# Into the unknown
active monitoring of neural networks (extended version)

Kueffner, Konstantin; Lukina, Anna; Schilling, Christian; Henzinger, Thomas A.

**DOI**
10.1007/s10009-023-00711-4

**Publication date**
2023

**Document Version**
Final published version

**Published in**
International Journal on Software Tools for Technology Transfer

International Journal on Software Tools for Technology Transfer
https://doi.org/10.1007/s10009-023-00711-4

GENERAL

Special Issue: RV 2021

# Into the unknown: active monitoring of neural networks (extended version)

Konstantin Kueffner[1] · Anna Lukina[2] · Christian Schilling[3] · Thomas A. Henzinger[1]

Accepted: 30 May 2023

**Abstract**

Neural-network classifiers achieve high accuracy when predicting the class of an input that they were trained to identify. Maintaining this accuracy in dynamic environments, where inputs frequently fall outside the fixed set of initially known classes, remains a challenge. We consider the problem of monitoring the classification decisions of neural networks in the presence of novel classes. For this purpose, we generalize our recently proposed abstraction-based monitor from binary output to real-valued quantitative output. This quantitative output enables new applications, two of which we investigate in the paper. As our first application, we introduce an algorithmic framework for active monitoring of a neural network, which allows us to learn new classes dynamically and yet maintain high monitoring performance. As our second application, we present an offline procedure to retrain the neural network to improve the monitor's detection performance without deteriorating the network's classification accuracy. Our experimental evaluation demonstrates both the benefits of our active monitoring framework in dynamic scenarios and the effectiveness of the retraining procedure.

**Keywords** Monitoring · Neural networks · Novelty detection

## 1 Introduction

Automated classification is an essential part of numerous modern technologies and one of the most popular applications of deep neural networks [25]. Neural-network image classifiers have fast-forwarded technological development in many research areas, e.g., automated object localization as a stepping stone to successful real-world robotic applications [46]. Such applications require a high level of reliability.

However, when deployed in the real world, neural networks face a common problem of novel input classes appearing at prediction time, leading to inherent misclassifications, which can stay undetected, accumulate over time, eventually reduce the overall accuracy, and possibly lead to system failures. The likelihood of severe system damage increases with the frequency and diversity of novel input classes. Typically, this risk is addressed by detecting novel inputs, augmenting the training dataset, and retraining the classifier from scratch [34]. This procedure is not only inefficient, but also leaves the system vulnerable until such a dataset has been collected. Techniques to incrementally adapt classifiers at prediction time are beneficial for improving accuracy in real-world applications [37, 39]. They, however, do not provide desired interpretability for humans.

Therefore approaches to run-time monitoring of neural networks were introduced [36]. In particular, approaches based on abstractions [5, 6, 19, 48] proved to be effective at detecting novel input classes and provide transparency of neural-network monitoring. Crucially, these monitors are constructed offline and remain static at prediction time. Functionalities they are still lacking are distinguishing between "known" and "unknown" novelties and selectively adapting at prediction time.

In this work, we propose a new monitor designed for the adaptive setting. In contrast to traditional qualitative monitoring, which judges whether or not an observed input–output pair of the network is reliable, our new *quantita-*

✉ A. Lukina
a.lukina@tudelft.nl

✉ C. Schilling
christianms@cs.aau.dk

K. Kueffner
konstantin.kueffner@ist.ac.at

T.A. Henzinger
tah@ist.ac.at

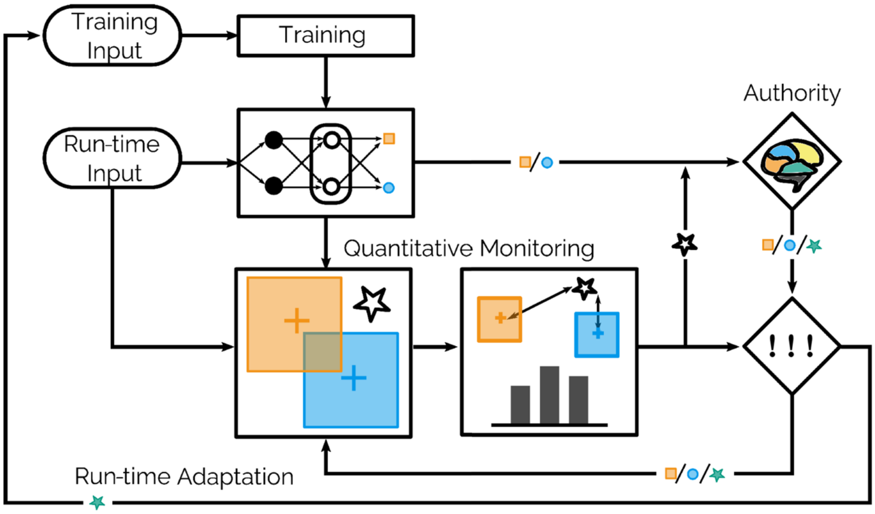
[1] IST Austria, Am Campus 1, Klosterneuburg, 3400, Austria

[2] TU Delft, Van Mourik Broekmanweg 6, Delft, 2628 XE, The Netherlands

[3] Aalborg University, Selma Lagerlöfs Vej 300, Aalborg, 9220, Denmark

Published online: 30 June 2023

**Fig. 1** High-level overview of the framework



*tive* monitor computes a numerical "reliability metric" for each observed input–output pair. For this purpose, we extend the abstraction-based monitor from our previous work [19], which was only able to give qualitative output. Very briefly, the idea was to detect unusual patterns in the feature space (i.e., the hidden layers) of the neural network. The metric of our new quantitative monitor is a distance in this feature space.

Based on the quantitative feedback of the monitor, we present an active monitoring framework for neural networks that detects novel input classes, obtains the correct labels from a human authority, and adapts the neural network and the monitor to the novel classes, all at prediction time. The framework contains a mechanism for automatic switching between monitoring and adaptation based on run-time statistics.

Figure 1 visualizes the chronology of the different steps. The neural network receives an input at run-time. This input is classified while the monitor watches the classification process. In this work, we abstract from what should happen if the monitor reports a misclassification; in the figure, this is reported to an authority, but we do not assume this authority to be continuously available. The authority's sole task is to assess the reported input and assign the correct label to it. When enough samples have been labeled by the authority in such a way, the adaptation is triggered.

Each reported input falls into one of three possible cases: it can be either a novelty, a misclassification of a known class, or a correct classification. From the monitor's perspective, the first two cases are true positives, whereas the latter is a false positive. Correspondingly, adaptation consists of either retraining the neural network to learning new classes (case 1), retraining the neural network to improve its accuracy (case 2), or adapting the monitor to improve its accuracy (case 3) whenever enough data of the corresponding case has been collected. Retraining is applied to the network and the monitor independently.

The quantitative metric allows for easy adaptation to newly introduced labels at prediction time and maintains overall classification accuracy on inputs of known and previously novel classes combined. As such, our framework is an interactive and interpretable tool for informed decision making in neural-network based applications.

This paper is an extended version of Lukina et al. [28]. In addition, we present a new procedure to improve the quantitative monitor's detection performance. The goal is to detect more novelties while not raising more false warnings. The challenge for this task is the nature of novelties: We cannot anticipate what a novel class will look like and therefore cannot optimize for a particular novel class. Instead, we retrain the neural network on the proxy task of reducing the quantitative metric of the monitor on the *known* classes. Recall that the metric describes a distance in the feature space. The intuition is that, by reducing this distance, the network learns to pack specific features closer together, which allows the monitor to better detect novel features. To achieve this effect via retraining, we formulate a new training objective that incorporates both the original network's task and the task of reducing the quantitative metric. By carefully weighing these two tasks during retraining we avoid a decline of the network performance on its original objective.

We summarize the contributions of this paper:

1. We propose a quantitative monitor to measure the confidence of the novelty detection (Sect. 4).
2. We propose an automatic framework with two modes, monitoring and adaptation, that operates in parallel with the original neural network and adapts the monitor to novel input classes at prediction time (Sect. 5).
3. We propose a procedure based on the quantitative feedback of the monitor to retrain the neural network for im-

proving the monitor's performance and avoiding decline in the network's accuracy (Sect. 6).

4. We provide an experimental evaluation (Sect. 7) on a diverse set of image-classification benchmarks. The evaluation demonstrates the effectiveness of the framework for achieving high monitor performance over time. Given a fixed budget of times the monitor can query the authority for a label, our monitoring approach adapts to the available classes and consumes the budget more effectively. Our evaluation also shows that the retraining approach yields neural networks with equal classification performance but significantly improved monitor performance.

## 2 Related work

**Novelty detection** Gupta and Carlone [16] consider neural networks that estimate human poses, for which they propose a domain-specific monitoring algorithm trained on perturbed inputs. Our framework is not limited to any specific domain of images. Common novelty-detection approaches [35] examine the input-sample distribution [22], which is computationally heavier at run-time than our monitor. Several approaches monitor the neuron valuations and compare to a "normal" representation of those valuations per class, obtained for a training dataset: the patterns of neuron indices with highest values [41] or positive/nonpositive values [6], and a box abstraction [19, 20, 48]. These monitors are purely qualitative and hence not adaptive, in contrast to our metric-based monitor. Recent work takes a statistical approach to latent-space approximation [17], where the intervals, calculated based on fitted Gaussians, serve as an alternative to the box abstraction.

**Anomaly detection** There are other directions for detecting more general anomalous behavior, not necessarily only novel classes. In *selective classification*, an input is rejected based on a (quantitative) confidence score, already at training time [12]. The probably best-known approach classifies based on the *softmax* score [15, 18], which is shown to be limited in effect [11]. A recent approach looks at the neurons' relative activation and deactivation patterns to detect *out-of-distribution* inputs [53]. Similarly to ours, this approach builds an abstraction of the hidden layers. Approaches to *failure prediction* identify misclassifications of *known* classes [51]. *Domain adaptation* techniques detect when the underlying data distribution changes, which is needed for reliable statistical methods [38]. Notably, Royer and Lampert [39] show that correlations in the data distribution can be exploited to increase a classifier's accuracy; although that approach applies to arbitrary classifiers in an unsupervised setting, it cannot deal with unknown classes. Sun and Lampert [44] study the detection of *out-of-spec situations* where classes do not occur with the expected frequency. An important aspect of domain adaptation, *transfer learning* [33, 45], is challenging online [54].

**Continuous/incremental learning** A central obstacle in incremental learning is *catastrophic forgetting*: the classifier's accuracy for known classes decreases over time [31]. We mitigate that obstacle by maintaining a sample of the training data and tuning the model on demand. Mensink et al. [32] find that a simple nearest-class-mean (NCM) classifier (mapping an input to feature space and choosing the closest centroid of all known classes) is effective; they also consider multiple centroids per class, as we do, but they use the Mahalanobis distance in contrast to our more lightweight distance. Guerriero et al. [14] extend that idea to nonlinear deep models, where the focus is on efficiency to avoid constant retraining; we also delay retraining (network and monitor) until accuracy deteriorates. Rebuffi et al. [37] extend the NCM classifier for *class-incremental learning* with fixed memory requirements. That learning approach, working in a completely supervised scenario, retrains the neural network using sample selection/herding and rehearsal. These ideas could also be integrated in our framework, but a representative sampling for our monitor is harder to obtain. Similar to the NCM approach is the proposal by Mandelbaum and Weinshall [30] to obtain a confidence score using a $k$-nearest-neighbor distance based on the Euclidean distance with respect to the training dataset, for which they require to modify the training procedure; we do not need access to the training procedure, and we experimentally found that the Euclidean distance is not suitable for networks with different scales at different neurons.

**Active learning** Our approach for active monitoring is inspired by active learning. Active learning aims to maximize prediction accuracy even on unseen data by detecting the most representative novel inputs to label and incrementally retraining the neural network on a selected sample of labeled novelties [42]. In contrast, the performance of our framework is measured primarily by the run-time detection of the monitor. We therefore use the incrementally retrained neural network solely for the monitor adaptation in parallel with the original model. Our quantitative monitor also reasons about the feature space of the neural network (and not the input space).

An essential idea behind active learning is that when selecting the training data systematically, fewer training samples are needed; this selection is usually taken at run-time by posing labeling queries to an authority [42]. Our approach follows the spirit of *selective sampling*, where data comes from a stream, from the *region of uncertainty* [8]. Das et al. [9] use a statistical outlier detection adapting to the reactions of the authority.

In an *open world* setting, novel classes have to be detected on the fly, and the classifier needs to be adapted accordingly. This setting is first approached by Bendale and Boult [1] using an NCM classifier and by Bendale and Boult [2] with a softmax score. More recently, Mancini et al. [29] propose a deep architecture for learning new classes dynamically. Wagstaff and Lu [47] argue that two main obstacles in this setting are the cold starts and the cost of having the classifier in the loop.

**Combining losses** Our proposed procedure for incorporating monitoring feedback into the neural network training is inspired by the recent work on provable robustness for neural networks [10]. Although this work is orthogonal to ours, we employ a similar approach to balancing two losses: the monitor distance loss and the classification loss, weighted accordingly with adaptive coefficients.

## 3 Background and assumptions

In this paper, we deal with neural networks, which we denote by $\mathcal{N}$. For simplicity, we present the concepts assuming a single feature layer $\ell$ of the network, but they generalize to multiple feature layers in a straightforward way. It is widely believed that the essential feature information is available in layers close to the final network output [50]. Note, however, that a single feature layer is typically the best choice [19].

For our purposes, a *monitor* is a function taking both an input and a classifier prediction, and then assesses whether that prediction is correct. The monitor raises a warning if it suspects that the prediction is incorrect. The assessment can be qualitative ("yes" or "no") or quantitative (expressing the confidence of the monitor). We write $\vec{x}$ for an unlabeled data point, $\mathcal{X}$ for a (possibly labeled) dataset, $y \in \mathcal{Y}$ for a class in a set of classes, and $(\vec{x}, y)$ for a labeled data point.

**Observing feature layers** We are given a trained neural network $\mathcal{N}$ and a labeled dataset $\mathcal{X}$ (which is not necessarily the dataset that $\mathcal{N}$ was trained on) with classes $\mathcal{Y}$. When we *observe a feature layer* $\ell$ for some input $\vec{x}$, we obtain the corresponding neuron valuations at layer $\ell$, which we view as a vector. We can thus compute the set of neuron valuations $\mathcal{V}_y$ for each labeled data point $(\vec{x}, y) \in \mathcal{X}$ of class $y \in \mathcal{Y}$.

**Performance metrics** As conventionally used for assessing the performance of classifiers and monitors, we compute different scores. Observe that a monitor is also a binary classifier, so classifier scores apply to monitors as well. The *accuracy* is the ratio of correct classifications over all classifications. The *precision* is the ratio of true positives $\texttt{TP}$ over the total number of positive predictions (including false positives $\texttt{FP}$): $\texttt{TP}/(\texttt{TP} + \texttt{FP})$. For example, for the monitor this is the ratio of correct warnings over the total number of warnings. At run-time we can only compute the precision based on samples that we know the ground truth for, i.e., samples reported by the monitor and subsequently labeled by an authority. Other metrics are the true positive rate $\texttt{TPR} = \texttt{TP}/(\texttt{TP} + \texttt{FN})$ and the false positive rate $\texttt{FPR} = \texttt{FP}/(\texttt{FP} + \texttt{TN})$.

**Assumptions** In this work, we make a number of assumptions. First, we assume the availability of an authority that assigns the correct label for any requested input. Although a human can play this role in many cases, in certain applications, like medical image processing, such an authority does not necessarily exist. Second, in our experimental setup, we assume that the authority is available in real time. We also occasionally adapt the monitor or retrain the neural network. While faster than building from scratch, this takes a non-negligible amount of time. In time-critical applications, we would need to delay these interactions and adaptations accordingly. In practice the authority can also be queried in batches, which only results in a delay of the adaptation process. Third, neural networks require a large amount of data points to learn new classes. In our evaluation, there is sufficient data available. Still, there are approaches that work with only few samples [3, 27].

## 4 A quantitative monitor in feature space

In this section, we propose a quantitative monitor for neural networks. At run-time, given an input $\vec{x}$ and a corresponding prediction $y$ of the neural network, the monitor observes the feature layer $\ell$ and compares its valuation to a model of "typical" behavior for the class $y$. This comparison is based on a distance, and if this distance exceeds a given threshold, then the input is flagged as a warning. Next, we describe the steps to initialize this monitor, i.e., to construct said behavioral model; these steps are also illustrated in Fig. 2. Given a labeled training dataset, we observe the neuron valuations for each class $y \in \mathcal{Y}$ (Fig. 2 (a)). Instead of working on the neuron valuations directly, we obtain $\mathcal{V}_y$ by applying a transformation matrix learned by principal component analysis (PCA) [21] to them. This reduces the dimensions and increases the relevance of each dimension (principal component, PC). We then apply a clustering algorithm to the sets $\mathcal{V}_y$ (Fig. 2 (b)). In our implementation, we use *k-means* [26] and find $k$ dynamically. Clustering is motivated because inputs corresponding to the same class are typically mapped to one of several disjoint neighborhoods in feature space, as observed by Henzinger et al. [19].

So far the initialization is shared with the qualitative monitor by Henzinger et al. [19], which would next compute the *box abstraction* for each cluster, i.e., the smallest box that

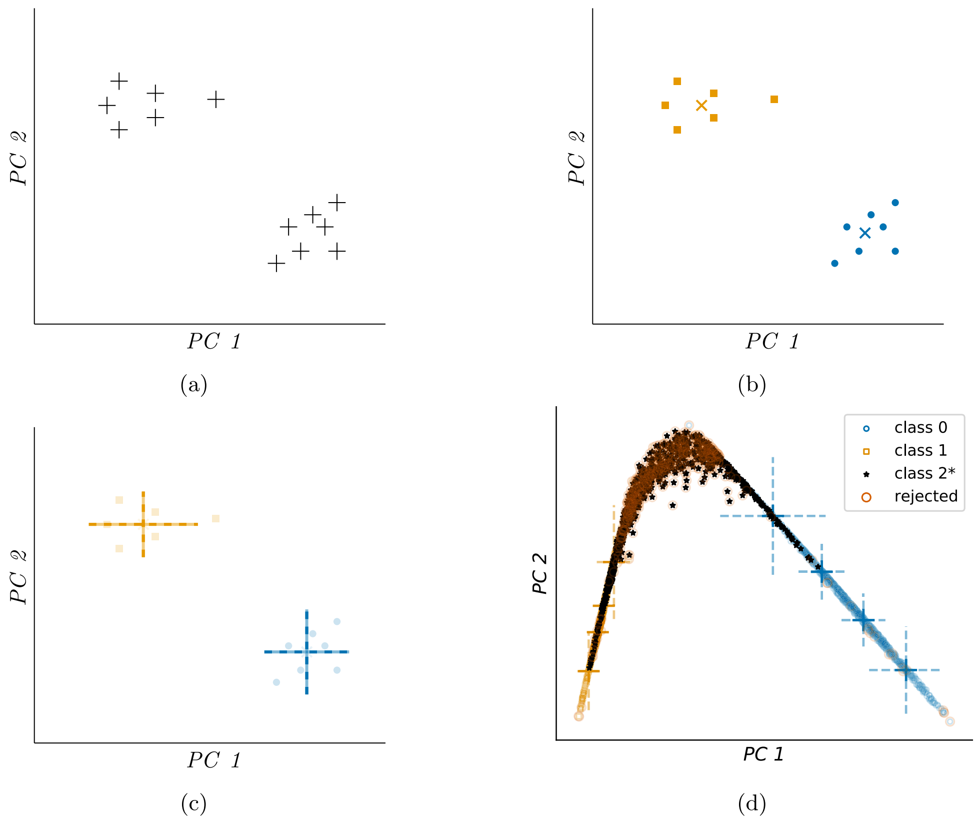


**Fig. 2** Illustration of the steps for initializing the quantitative monitor on a fixed class in a two-dimensional projection on the first two principal components *PC* 1 and *PC* 2 of the feature layer $\ell$. (a) Sampling of data points. (b) Result of clustering (here two clusters ● and ■) where × and ×, respectively, mark the cluster centers. (c) Quantitative metric for each cluster, visualized as dashed lines. (d) Projection of a quantitative monitor and its detection results for a network trained on the first two classes of the MNIST dataset

contains all points in the cluster. A qualitative abstraction-based monitor can only determine whether a point lies inside the abstraction (here a box) or not. Since we are interested in a quantitative monitor, we instead define a *distance function* below.

**Distance function** We set reference points for computing the distance function at the cluster centers. This way the majority of points have low distance. Below we describe the particular distance function, which we found effective in our evaluation, also depicted in Fig. 2 (c) and (d).

Let us fix a class $y \in \mathcal{Y}$ and a corresponding cluster $B^y$ with center $\vec{c} = (c_1, \ldots, c_n)^\mathrm{T}$ of dimension $n$. Let $\vec{r} = (r_1, \ldots, r_n)^\mathrm{T}$ be the radius of the bounding box around the cluster. We define the distance of a point $\vec{p} = (p_1, \ldots, p_n)^\mathrm{T}$ to $B^y$ as the maximum absolute difference to $\vec{c}$ in any projected dimension $i$, normalized by the radius $r_i$:

$$d_+(\vec{p}, B^y) = \max_i |c_i - p_i| \cdot r_i^{-1}.$$

The distance generalizes to a set $\mathcal{B}^y$ of clusters for the same class $y$ by taking the minimum distance in the set:

$$d_+(\vec{p}, y) = \min_{B^y \in \mathcal{B}^y} d_+(\vec{p}, B^y).$$

Computing the distance is linear in the dimension (i.e., the number of neurons in the feature layer). We note that we can in principle also generalize the distance to a set of classes $\mathcal{Y}$ to obtain a new classifier. In this paper, for the purpose of monitoring, we just compare the distance for a fixed class to some class-specific threshold. By default this threshold is 1, but we can also adapt it during active monitoring, which we explain in the next section.

## 5 Active monitoring algorithm

We design our monitoring framework to achieve high precision in detecting novel classes without depressing the learned model's run-time performance. To address this trade-off, our framework operates in stages, switching between monitoring and adaptation. This procedure is based on parallel composition of two components: a dynamically adapted copy of the original neural network and a monitor that originally knows the same classes as the network. During monitoring, inputs to the network that are flagged by the monitor are passed to an authority for assigning the correct label. From that, performance scores for both the monitor and the neural network are assessed for whether adaptation is required. During adaptation, depending on the assessment, the neural network or the monitor is incrementally adjusted, or they are retrained to learn an unknown class.

**Hyperparameters** Before we explain the algorithm, we introduce several parameters for the neural network, the monitor, and the online procedure. We assume that the user sets the upper bound on the loss of the prediction accuracy during deployment. We define the *model performance threshold* $s^*_{network}$ as 95% of the accuracy score of the original neural-network model $\mathcal{N}$ on a test dataset (with classes known to $\mathcal{N}$), which we use for making decisions about model adaptation. The parameter $s^*_{samples}$ is the number of collected and labeled data samples of a novel class sufficient for incremental adaptation of the model to this class, which we set to $s^*_{samples} = 0.05|\mathcal{X}|/|\mathcal{Y}|$ for an initially given dataset. Naturally, the higher $s^*_{network}$ is, the more data would need to be collected and labeled to perform online adaptation. We therefore expect that initially more than 5% of the average size of training data per class is required to adapt to a novel class of inputs. The parameter $s^*_{monitor}$ is the desired precision threshold of the monitor at run-time, which we set to 0.9. This value can be established based on the initial average performance of the monitor constructed for a particular neural network and training dataset. The distance parameters $d^*_y$ (for each class $y \in \mathcal{Y}$) are thresholds for refining the inputs detected by the monitor, initialized to 1.

We now explain our active monitoring algorithm, summarized in Algorithm 1 and also illustrated in Fig. 1.

**Algorithm 1** Active monitoring

```
Input  $\mathcal{N}$: trained model; $\mathcal{X}$: training data; $\mathcal{X}_{run}$: online input stream
 1: while True do
 2:     $\mathcal{M}, \mathcal{Y} \leftarrow$ buildMonitor($\mathcal{N}$, $\mathcal{X}$, $\ell$) // build monitor $\mathcal{M}$ and extract known classes $\mathcal{Y}$ from $\mathcal{X}$
 3:     while True do
 4:         // monitoring mode
 5:         $\vec{x} \leftarrow$ get($\mathcal{X}_{run}$) // get next input $\vec{x}$
 6:         $y \leftarrow$ classify($\mathcal{N}$, $\vec{x}$) // predict class of $\vec{x}$
 7:         $\vec{p} \leftarrow$ observe($\mathcal{N}$, $\vec{x}$, $\ell$) // observe output at layer $\ell$
 8:         warning, $\vec{s} \leftarrow$ monitor($\vec{p}$, $y$, $\mathcal{M}$) // monitor and compute statistics $\vec{s}$
 9:         if warning then
10:             $y^\star \leftarrow$ askAuthority($\vec{x}, y, d_+(\vec{p}, y)$)
11:             $\mathcal{X} \leftarrow$ collect($\vec{x}, y^\star, \mathcal{X}$) // add labeled pair $(\vec{x}, y^\star)$ to $\mathcal{X}$
12:             adapt_model $\leftarrow$ evaluate($\vec{s}$, $\mathcal{X}$, $\mathcal{Y}$)
13:             // adaptation mode
14:             if adapt_model then
15:                 $\mathcal{N}$, $\mathcal{M}$, $\mathcal{X} \leftarrow$ adaptModel($\mathcal{N}$,$\mathcal{X}$) Ⓑ
16:                 break
17:             end if
18:             $\mathcal{M} \leftarrow$ adaptMonitor($\vec{s}$, $\mathcal{M}$, $\mathcal{N}$, $\mathcal{X}$) Ⓐ
19:         end if
20:     end while
21: end while
```

**Initialization** We start with a trained neural network $\mathcal{N}$ with a feature layer $\ell$ and a dataset $\mathcal{X}$ with a number of classes (the "known" classes) as inputs. The first step in line 2 is to initialize a monitor $\mathcal{M}$ for this network, for example, as described in Sect. 4 for our quantitative monitor. Recall that instead of working on the feature layer's neurons directly, we learn a transformation matrix by applying principal component analysis (PCA) [21] or Kernel PCA [40] to the neuron valuations $\mathcal{V}_y$. This transformation is not a requirement of our framework, and hence we omit it in the pseudocode; as we noticed experimentally, this step tends to further separate the valuations $\mathcal{V}_y$ and $\mathcal{V}_{y'}$ for different classes $y' \neq y$, which improves the overall monitor precision.

**Monitoring stage (lines 5–12)** At run-time, we apply our framework to a stream of inputs. For each input $\vec{x}$, we perform the following steps. We first apply the neural network to obtain both the class prediction $y$ and the (principal components of the) neuron valuations $\vec{p}$ at the feature layer $\ell$. We then query the monitor $\mathcal{M}$ about the prediction. In the case of the quantitative monitor, $\mathcal{M}$ computes the distance $d_+(\vec{p}, y)$ with respect to the predicted class $y$. Then $\mathcal{M}$ compares this distance to a class-specific threshold $d^*_y$; initially, this threshold is set to 1, but we increase this value during the course of the algorithm later.

In the simple case that $d_+(\vec{p}, y) \leq d^*_y$, the monitor does not raise a warning, and the framework just returns the predicted class $y$ for input $\vec{x}$ (not shown in the pseudocode). Otherwise, the monitor rejects the network prediction as unknown. In this case, we query the authority to provide the ground truth $y^*$ for input $\vec{x}$ and add the pair $(\vec{x}, y^*)$ to our training dataset $\mathcal{X}$. Our quantitative monitor additionally provides the authority with the distance $d_+(\vec{p}, y)$ as a confidence measure, whereas for qualitative monitors, this argument is missing. The procedure **evaluate**($\vec{s}$,$\mathcal{X}$,$\mathcal{Y}$), where $\vec{s} = \{s_{network}, s_{samples}, s_{monitor}\}$, decides between the following two scenarios, which we describe later.

Ⓐ The ground truth matches the prediction ($y^* = y$). In this case, it was not correct to raise a warning, and we continue with the monitor adaptation.

Ⓑ The ground truth does not match the prediction ($y^* \neq y$), possibly because $y^*$ is unknown to $\mathcal{N}$. In this case, it was correct to raise a warning, and we continue with the model adaptation.

**Monitor adaptation (line 18)** Procedure **adaptMonitor**($\vec{s}$, $\mathcal{M}$, $\mathcal{N}$, $\mathcal{X}$) for monitor adaptation in Ⓐ is triggered if a wrong warning was raised and only applies to our quantitative monitor. Recall that the reason for raising a warning is that the distance of $\vec{p}$ exceeds the threshold for class $y$. We do not immediately adapt the monitor every time it raises a wrong warning. Instead, we keep track of the monitor's performance over time in terms of a score $s_{monitor}$. We only adapt the monitor if $s_{monitor}$ drops below a user-defined threshold $s^*_{monitor}$. The adaptation performs two simple steps. First, we adapt the cluster centers to the new collected data in $\mathcal{X}$. Second, we adapt the distance threshold $d^*_y$ as follows. Let $s_{samples}$ be the number of samples of class $y$ that we have already collected in $\mathcal{X}$, and let $s^*_{samples}$ be a learning threshold as defined before. We define the new threshold $d^*_y$ as

$$d^*_y + (d_+(\vec{p}, y) - d^*_y) \cdot \frac{s^*_{samples}}{s_{samples}}.$$

The first term is the old threshold, which ensures that the threshold increases. The second term consists of a difference (the amount by which we would have to increase the distance to accept the given sample) and a discount factor relative to the number of samples of class $y$ (to reduce the influence of outliers).

**Model adaptation (lines 14–16)** In contrast to monitor adaptation, *model* adaptation in Ⓑ involves retraining the neural-network model to learn novel classes of inputs. Procedure **adaptModel**($\mathcal{N}$, $\mathcal{X}$) performs this adaptation only if one of the following conditions is satisfied:

- B.1 The number of collected samples labeled by the authority reaches a predefined threshold $s^*_{samples}$.
- B.2 The accuracy score of the current model $s_{network}$ falls below the desired value $s^*_{network}$.

In B.1, using the dataset $\mathcal{X}$ replenished with the data points reported by the monitor and labeled by the authority, we identify which class (or multiple classes) should be learned, based on the collected statistics $\vec{s}$. We then employ transfer learning [33] to train a new model that recognizes this class (classes) in addition to the ones already known. Specifically, we remove the output layer and all trailing layers until the last fully connected one and then add a new output layer corresponding to the desired number of classes present in $\mathcal{X}$. From the newly compiled model we also augment the monitor. In the case of our quantitative monitor, we apply the steps from Sect. 4 for the new class(es) and set the corresponding distance threshold(s) to 1.

In B.2, we rely on regular run-time measurements of the accuracy score for the current model. Algorithmically, this is achieved by keeping a separate (not used for retraining) test dataset after each successful transfer learning. We collect only the inputs reported by our monitor and subsequently labeled by the authority. This is in line with our main objective for the human in the loop to remain the ultimate trustee for the framework.

*Remark 1*
The model obtained from transfer learning on the accumulated labeled samples is not meant as a replacement for the original model provided at the initialization stage but rather as an assistant to ongoing active monitoring.

This concludes all possible cases for one iteration of the algorithm. This process is repeated for each input in the stream.

## 6 Making a neural network monitor-friendly

We are now coming back to the quantitative monitor proposed in Sect. 4. For the sake of discussion, we quickly recapitulate some relevant aspects. We are given a trained neural network and a training dataset, from which we construct an individual monitor. The construction finds a set of boxes for each known class that covers the values the network assumes at the feature layer $\ell$ on the training data (resp., after applying a transformation matrix). At run-time, when the network classifies an unknown input, the monitor computes the distance for the proposed class and, based on that distance, determines whether to consider the input a novelty. The assumption underlying the monitor is that the network maps inputs of the same class to clusters in feature space.

However, in practice the clusters are not necessarily packed together but may rather be spread out. In that case, it is more likely that the monitor assigns a low distance to a novel input and thus misses it (low true-negative rate). Missing such novelties can be easily avoided by decreasing the distance threshold for rejection, but this generally also increases the number of warnings for the outliers of known classes (high false-positive rate).

In this section, we try to break this vicious circle by retraining the neural network. The idea is to teach the network to shrink the spread within the clusters of the known classes (i.e., reduce the distance of outliers), as illustrated in Fig. 3. This allows us to decrease the distance threshold without increasing the number of false positives.

### 6.1 Distance loss

The standard way to train a neural network is to define a *loss function* to be minimized. Here we encode the distance as a new distance-loss function $\mathcal{L}_d$ and use a standard training procedure to minimize this loss. First, we construct the monitor (i.e., the cluster centers and distance parameters) as described in Sect. 4. Now for each labeled training input $(\vec{x}, y)$, we compute the values $\vec{p}$ at layer $\ell$ as usual (including the application of the transformation matrix) and then obtain the distance loss

$$\mathcal{L}_d(\vec{x}, y) = d_+(\vec{p}, y) = \min_{B \in \mathcal{B}^y} d_+(\vec{p}, B)$$

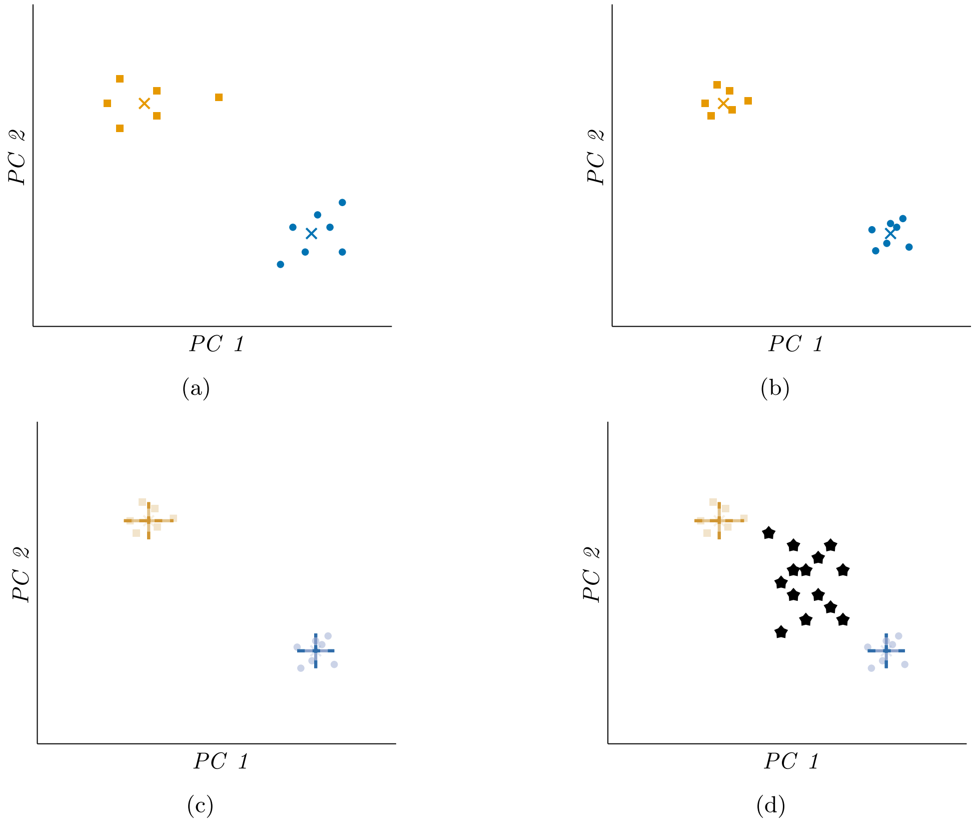


**Fig. 3** Illustration of the effect of retraining to improve the monitor precision. (a) Copy of Fig. 2 (b). (b) Retraining of the neural network packs the data points. (c) Quantitative metric for each cluster after retraining, visualized as dashed lines. (d) The retrained network rejects the novelties in black

$$= \min_{B \in \mathcal{B}^y} \max_i \frac{|c_i^B - p_i|}{r_i^B},$$

where $c^B$ and $r^B$ are the center and radius associated with cluster $B$, respectively.

### 6.2 Combining two objectives in one loss

Optimizing solely for the distance loss would render the neural network unusable, since the training procedure would forget the original classification task and instead only focus on packing all points in the feature space together. We therefore consider joint optimization of 1) the classification loss function $\mathcal{L}_N$ used during the original training of the neural network, which quantifies classification performance, and 2) the distance loss $\mathcal{L}_d(\vec{x}, y)$, responsible for shrinking latent clusters. Following common practice in multiobjective neural-network training, we optimize a weighted sum of the two losses:

$$\mathcal{L}(\vec{x}, y) = \alpha\beta\mathcal{L}_d(\vec{x}, y) + (1 - \alpha)\mathcal{L}_N(\vec{x}, y).$$

This function has two parameters $\alpha$ and $\beta$, which are recomputed before each learning iteration (epoch) and explained below.

Since we retrain an already accurate neural network, the distance loss $\mathcal{L}_d$ tends to assume values that overshadow the original loss $\mathcal{L}_N$. The parameter $\beta$ normalizes the distance loss to even out the importance of the two objectives:

$$\beta = \frac{\sum_{(\vec{x},y)\in\mathcal{X}} \mathcal{L}_N(\vec{x}, y)}{\sum_{(\vec{x},y)\in\mathcal{X}} \mathcal{L}_d(\vec{x}, y)}.$$

The parameter $\alpha$ adapts to the current accuracy $s_{network}$ of the neural network to carefully balance the two objectives during training. The value is determined based on a desired lower bound $s^*_{lba} \in (0, 1)$ (user defined) on the network accuracy (e.g., $s^*_{lba} = 0.9$ means that we do not want the accuracy to fall below 90%) in exponential fashion:

$$\alpha = 1 - 10^{-\gamma}, \quad \gamma = \frac{\max(s_{network}, s^*_{lba}) - s^*_{lba}}{1 - s^*_{lba}}.$$

### 6.3 Discussion

We end this section with some discussion and details on our implementation of the above retraining algorithm.

In a standard training scenario such as for a classification task, the objective is constant, and we can easily run training in consecutive epochs. In our case, the objective shifts all the time: Since the monitor is computed for a given neural network, it needs to be recomputed after retraining the network parameters. Furthermore, the data is processed twice: first to compute a monitor and then for the actual retraining;

**Table 1** Dataset and model description. The columns show the number of samples for training and testing, the number of classes in total and initially known to the network, the ID of the network architecture, and the number of neurons in the monitored layer $\ell$

| Dataset | Dataset size train/test | Classes all/init | Net ID | Number of neurons in $\ell$ |
|---|---|---|---|---|
| MNIST | 60,000/10,000 | 10/5 | 1 | 40 |
| FMNIST | 60,000/10,000 | 10/5 | 1 | 40 |
| CIFAR10 | 50,000/10,000 | 10/5 | 3 | 256 |
| GTSRB | 39,209/12,630 | 43/22 | 2 | 84 |
| EMNIST | 112,800/18,800 | 47/24 | 1 | 40 |

this can only be done sequentially. These two reasons make the retraining procedure computationally heavy.

Training datasets are usually split into smaller *batches*, and the network parameters change after each batch. Technically, we could recompute the monitor after each batch, but to save time, we only do that after each epoch (i.e., when all batches have been processed).

In our implementation, we assume that the number of retraining epochs is fixed in advance. We only retrain the monitor during the first half of the training epochs and then freeze it. This has two effects. First, training is faster after that point, and second, the objective does not change anymore, which in our experience helps the network parameters to converge.

As a technical note, the standard training algorithm for neural networks is stochastic gradient descent [4], which requires to compute the gradients of the network parameters. The gradients are computed using automatic differentiation [13], which requires to implement the involved operations in a compatible way. In our implementation, we had to reimplement the distance computation to enable the retraining procedure.

## 7 Experiments

In this section, we evaluate the approaches proposed in this paper. In Sect. 7.1, we shortly introduce five image-classification datasets that we use in the evaluation. In Sect. 7.2, we show experimental results on active monitoring from Sect. 5. We compare our quantitative monitor to three other (static) monitoring strategies: a box-abstraction monitor [19], a monitor based on the softmax score [18], and a monitor that warns with uniform random rate. We also investigate the influence of different parameters on our quantitative monitor specifically. In Sect. 7.3, we show the performance of the retraining procedure proposed in Sect. 6.

We implemented our framework in Python 3.6 with Tensorflow 2.2 and scikit-learn. We ran all experiments on an i7-8550U@1.80 GHz CPU with 32 GB RAM. The source code and scripts that we used are available online.[1]

### 7.1 Benchmark datasets

We consider the following publicly available datasets, summarized in Table 1: MNIST [24], Fashion MNIST (FMNIST) [49], and Extended MNIST (EMNIST) [7] consist of 28 × 28 grayscale images; CIFAR10 [23] and the German Traffic Sign Recognition Benchmark (GTSRB) [43] consist of 32 × 32 color images. As network architectures, we use VGG16 [52] pretrained on ImageNet for CIFAR10, and the architectures from Cheng et al. [6] for MNIST (which we also use for FMNIST and EMNIST) and GTSRB.

### 7.2 Active monitoring

**Experimental setup** For each of the benchmarks (combinations of a network and a dataset) we trained two neural-network models: one model trained on all classes, which we refer to as the "static full" model, and one model trained on half of the classes, which we refer to as the "static half" model. We let the framework process inputs in batches of size 128. For each dataset, we ran our active monitoring framework on reshuffled data five times. We evaluate our active monitoring framework with four different monitoring strategies, each of which uses the same overall processing within the framework, e.g., the same sequence of samples in the input stream and the same policy for model adaptation. The strategy based on the softmax score rejects inputs when the score falls below 0.9. The random strategy rejects inputs with probability $p = 5\%$ (resp., $p = 10\%$ in the EMNIST experiment). To make the comparison fair, we limit the number of available authority queries for each strategy to a budget of $p$ (the random rejection probability) percent of the full dataset. For most benchmarks, we use PCA and $s^*_{samples}$ as explained in Sect. 3. For CIFAR10, we use Kernel PCA and $s^*_{samples} = 0.01|\mathcal{X}|/|\mathcal{Y}|$.

[1] https://github.com/VeriXAI/Into-the-Unknown-extended.

**Table 2** Monitor comparison. We compare four different monitoring strategies: quantitative (this paper), box abstraction, softmax score, and random warning. For each benchmark, we report the interaction limit with the authority, the highest number of learned classes, and the average monitoring precision of five runs. The best results per benchmark are marked in **bold**

| Dataset | Interaction limit | Quantitative classes/prec | Abstraction classes/prec | Softmax classes/prec | Random classes/prec |
|---|---|---|---|---|---|
| MNIST | 3,000 | **10** | **10** | **10** | 6 |
| | | **0.81** ± 0.01 | 0.6 ± 0.02 | 0.71 ± 0.01 | 0.48 ± 0.01 |
| FMNIST | 3,000 | 9 | 9 | **10** | 8 |
| | | **0.74** ± 0.02 | 0.54 ± 0.02 | 0.7 ± 0.01 | 0.5 ± 0.01 |
| CIFAR10 | 2,500 | **10** | **10** | **10** | **10** |
| | | **0.75** ± 0.02 | 0.61 ± 0.02 | 0.53 ± 0.01 | 0.41 ± 0.01 |
| GTSRB | 1,960 | 37 | **38** | 34 | 25 |
| | | 0.67 ± 0.02 | 0.7 ± 0.01 | **0.75** ± 0.03 | 0.29 ± 0.01 |
| EMNIST | 11,280 | 42 | **47** | **47** | **47** |
| | | **0.81** ± 0.01 | 0.71 ± 0.02 | 0.69 ± 0.01 | 0.39 ± 0.01 |

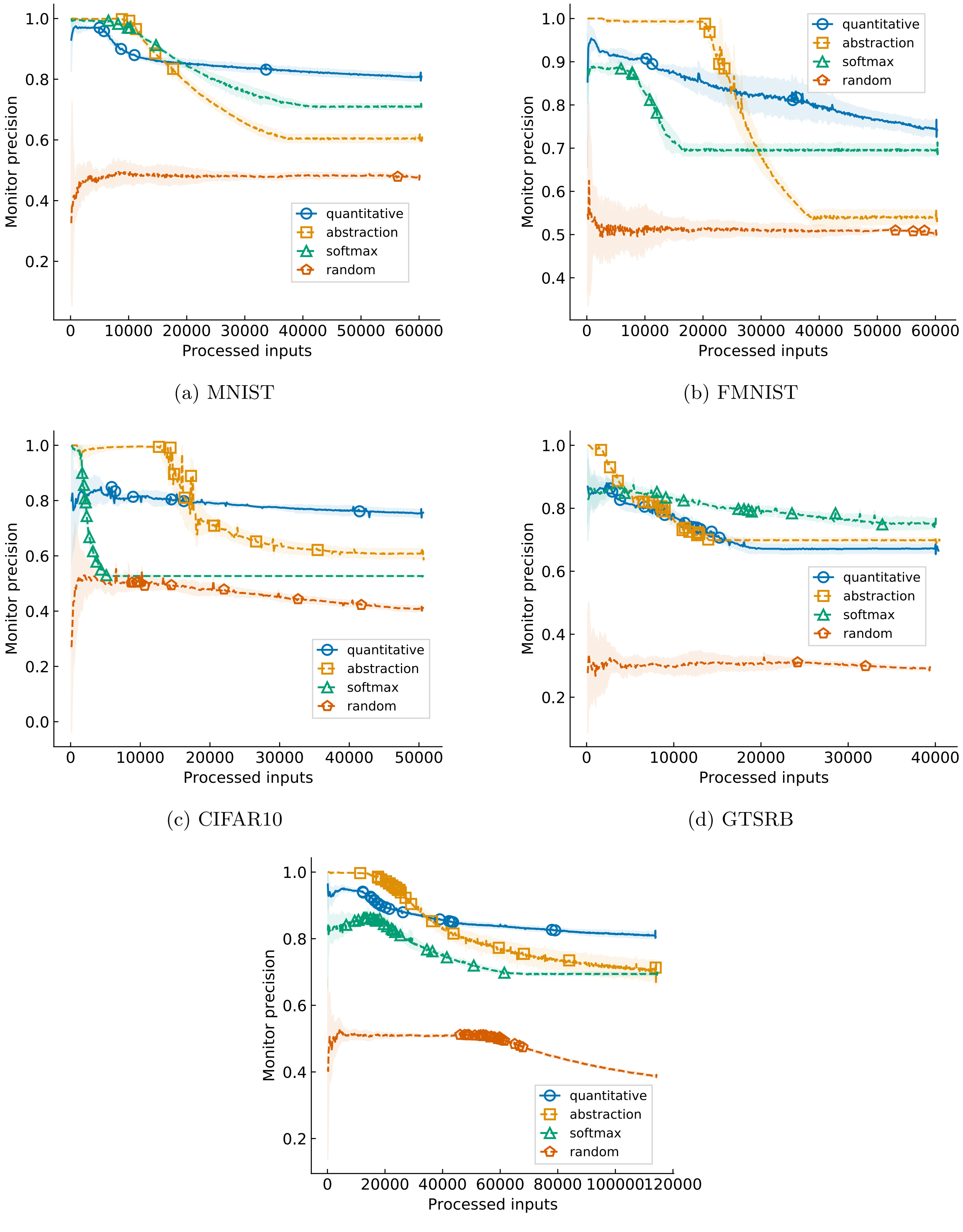
**Table 3** Model adaptation. We compare the accuracy of the neural networks (training and test accuracy) between the static model trained on 50% of the classes, the static model trained on all classes, and the model obtained from our framework (using the quantitative monitor), averaged over five runs. In the static cases the test accuracy is measured on the filtered test set (not including novelties for the 50% model), and hence the accuracy is generally higher for the 50% model. The second column shows the epochs used for the initial training resp. the retraining/transfer learning at run-time

| Dataset | Epochs init/run | Static half train/test | Static full train/test | Adaptive test |
|---|---|---|---|---|
| MNIST | 10/10 | 0.99/0.99 | 0.99/0.99 | 0.97 ± 0.01 |
| FMNIST | 10/10 | 0.99/0.92 | 0.97/0.91 | 0.79 ± 0.05 |
| CIFAR10 | 50/30 | 0.99/0.83 | 0.99/0.79 | 0.54 ± 0.02 |
| GTSRB | 30/30 | 0.99/0.95 | 0.99/0.88 | 0.87 ± 0.01 |
| EMNIST | 30/30 | 0.97/0.92 | 0.92/0.86 | 0.71 ± 0.04 |

**General performance** The performance of the different monitoring strategies in terms of monitoring precision is averaged over five runs and summarized in Table 2. For all but one benchmark, our monitor achieves the highest precision, and for GTSRB, the precision is comparable with other monitors. GTSRB consists of traffic signs, which are easier for the neural network to learn, owing to overlapping features, but can be difficult for an abstraction-based monitor to distinguish. Our distance quantification does not improve the monitor precision for GTSRB since the distance from novel images to the known ones can be quite small after the monitor has learned the majority of classes of traffic signs. This is also reflected in the online performance of the monitor discussed next, where most of the classes are learned early. Figure 4 shows the evolution of the monitor precision over time as more classes are learned. Recall that the network is dynamically retrained (using transfer learning) for new classes. Clearly, the number of new samples for this training procedure is lower than in a normal, full-fledged training. Consequently, the adapted network is less accurate for these new classes (cf. Table 3) than a network trained on the full training dataset. Hence it is expected that the general trend in the monitoring precision is decreasing for all strategies.

We report the test accuracy of the neural networks in Table 3, averaged over five runs per benchmark. The accuracy is generally lower than what could be achieved by training the network with a full and balanced dataset from scratch (the "static full" model), but for some benchmarks, we achieve almost the same accuracy. This shows the framework's ability to adapt to new situations.

**Cost analysis** In Fig. 5, we show the frequency of authority queries. Recall that there is a budget of queries (cf. Table 2). Our quantitative monitor queries the authority more frequently at the beginning, but as it adapts to more novel classes, the rate of requests is steadily decreasing. Hence the monitor has the fewest queries in four of the five benchmarks (except for GTSRB). The other monitors do not have an adaptation mechanism and therefore are prone to querying the authority more often. For some monitors, we even observe an increase in warnings over time, in particular, the monitor that uses the softmax score. As we argued above, we suspect that the network tends to be less confident for newly



(a) MNIST

(b) FMNIST

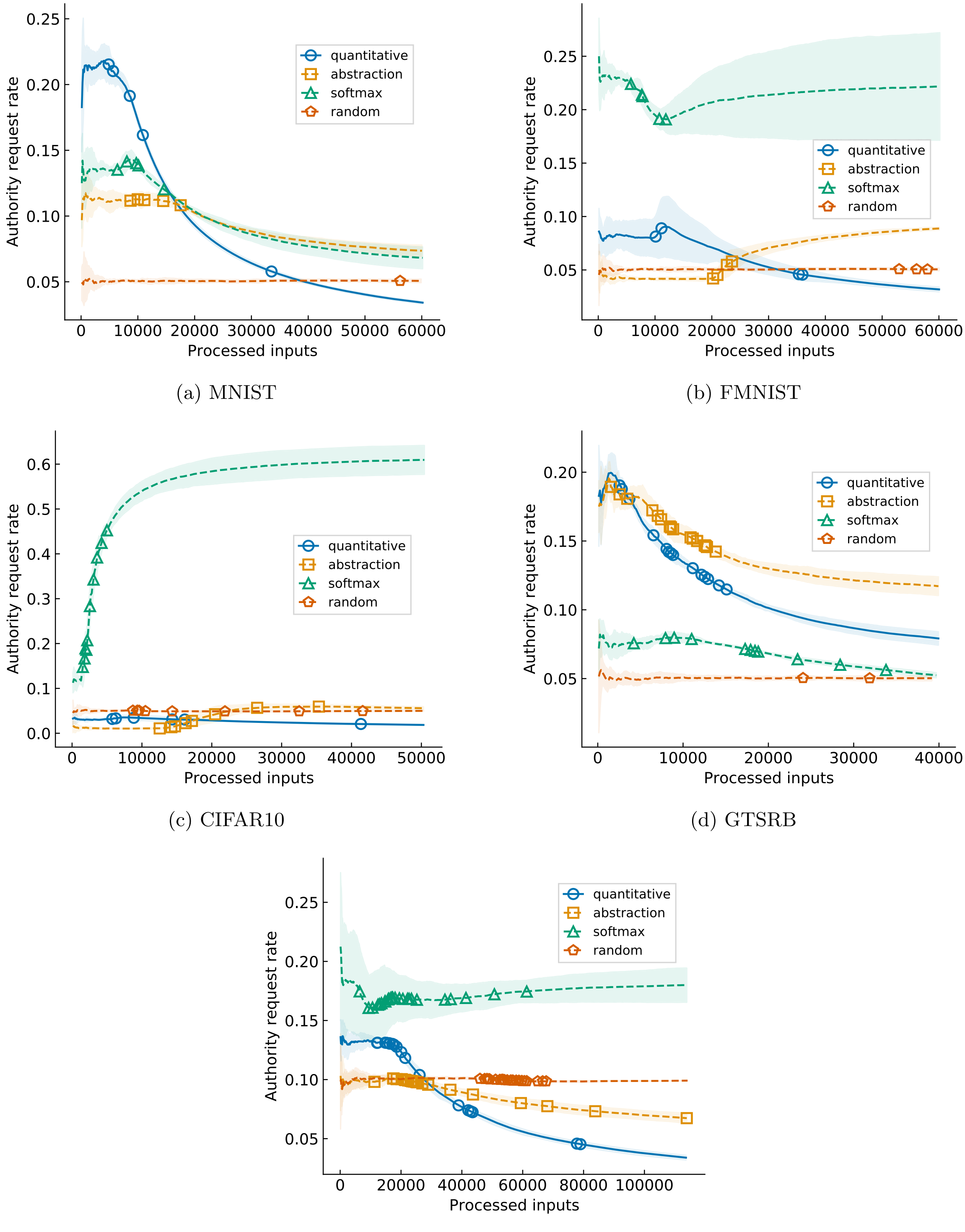
(c) CIFAR10

(d) GTSRB

(e) EMNIST

**Fig. 4** Comparison of the monitor precision between four monitoring strategies, averaged over five runs and including 95%-confidence bands. The markers correspond to points in time when a model adaptation takes place

learned classes, which results in lower softmax scores. Also, the strategies often learn new classes at roughly the same point in time. This is because the novelties appear with uniform distribution in the input stream, so the points in time when a fixed number per class has been seen are close to each other.



(a) MNIST

(b) FMNIST

(c) CIFAR10

(d) GTSRB

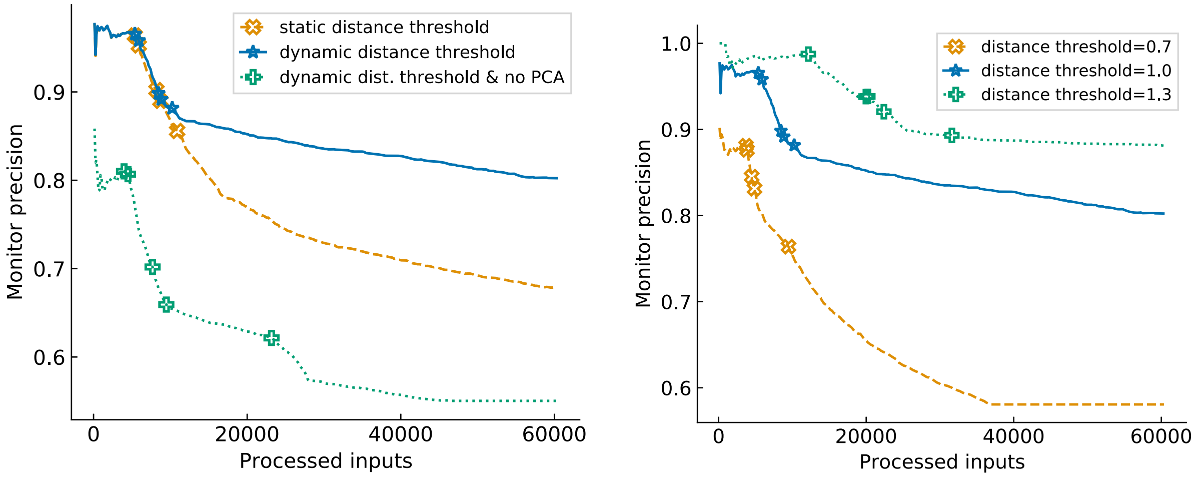
(e) EMNIST

**Fig. 5** Comparison of the rate of authority queries between four monitoring strategies, averaged over five runs and including 95%-confidence bands. The markers correspond to points in time when a model adaptation takes place

Overall, the plots do not reveal a clear trend which monitor is fastest at learning new classes. There is generally a trade-off between the rate at which a warning is raised and the rate at which new classes are learned. In our scenario, raising a warning is initially correct in 50% of the cases (note that none of the monitors is in that range); taken to the extreme, a

**Table 4** Average run times in seconds. For each active-monitoring benchmark, we average (five runs) the time for retraining the neural network (after collecting enough samples), for retraining the monitor (after retraining the network), and for adapting the monitor (when the precision drops too much)

| Dataset | Retrain network | Retrain monitor | Adapt monitor |
|---|---|---|---|
| MNIST | 26 ± 1 | 59 ± 3 | 39 ± 5 |
| FMNIST | 19 ± 6 | 45 ± 10 | 59 ± 5 |
| CIFAR10 | 257 ± 57 | 2,477 ± 282 | 40 ± 3 |
| GTSRB | 228 ± 12 | 194 ± 24 | 19 ± 1 |
| EMNIST | 360 ± 191 | 347 ± 71 | 82 ± 16 |



(a) Static and dynamic distance threshold.

(b) Different initial threshold values.

**Fig. 6** Influence of the dynamic distance threshold $d_y^*$ for each class $y$ on the quantitative-monitoring precision for the MNIST benchmark. The markers correspond to points in time when a model adaptation takes place. (a) Comparison between a static value and a dynamically changing value (as proposed in this paper); we also show a comparison with a run where we omit the preprocessing with PCA. (b) Influence of the initial value of the threshold

monitor that always raises a warning would be the fastest in learning new classes. On the other hand, a monitor that generally raises fewer warnings to the authority may also miss novelties and thus learn slower. However, in our experience, it is more preferred to provide a low false-positive rate, i.e., warnings raised by the monitor should be genuine. In this sense the quantitative monitor works best.

**Ablation and sensitivity study** All components of our framework contribute to its performance. In Fig. 4, we illustrate how incremental retraining of the model improves the monitor precision for all monitoring strategies. In principle, other active-learning strategies can be plugged into our framework to further increase this effect. In addition, Fig. 4 demonstrates that the monitor-adaptation stage (where the monitor is incrementally adjusted without *model* adaptation), which only applies to our quantitative monitor, helps maintaining a better precision than the other monitoring strategies.

Figure 6 (a) shows that dynamically changing the distance threshold $d_y^*$ (for each class $y$) contributes to the precision of our monitor, as does the use of PCA for dimensionality reduction. Figure 6 (b) shows that the starting value of the (dynamic) threshold also influences the monitor precision.

**Timing analysis** Table 4 shows a timing comparison for the individual adaptation stages of the framework, taken from the runs for the quantitative monitor strategy. (Comparing different strategies is generally difficult because they interact with the authority and adapt the model and/or the monitor in different orders and frequencies.) The time grows with the size of the dataset but on average is on the order of milliseconds per input; hence the framework can be run in real time. For CIFAR10, the time is dominated by the use of Kernel PCA.

**Table 5** Retraining with distance loss. We study the effect of retraining the network with the distance loss. For that, we compare the monitoring precision to the results from Table 2. As before, for each benchmark, we report the highest number of learned classes, and the average monitoring precision of 7 runs. The best results per benchmark are marked in **bold**

| Dataset | Distance loss classes/prec | Quantitative classes/prec | Abstraction classes/prec | Softmax classes/prec | Random classes/prec |
|---|---|---|---|---|---|
| MNIST | **10** | **10** | **10** | **10** | 6 |
| | **0.82** ± 0.0 | 0.81 ± 0.01 | 0.6 ± 0.02 | 0.71 ± 0.01 | 0.48 ± 0.01 |
| FMNIST | 9 | 9 | 9 | **10** | 8 |
| | **0.75** ± 0.0 | 0.74 ± 0.02 | 0.54 ± 0.02 | 0.7 ± 0.01 | 0.5 ± 0.01 |
| GTSRB | **42** | 37 | 38 | 34 | 25 |
| | 0.67 ± 0.0 | 0.67 ± 0.02 | 0.7 ± 0.01 | **0.75** ± 0.03 | 0.29 ± 0.01 |

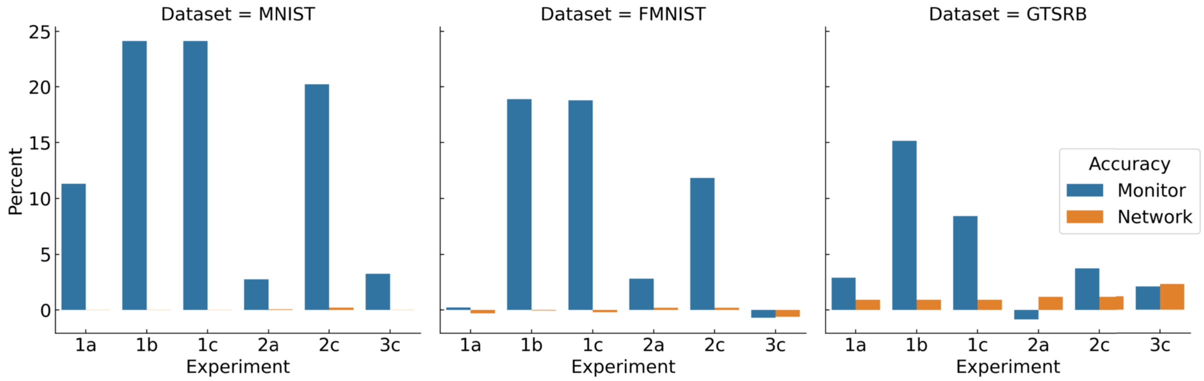


**Fig. 7** Relative change in accuracy of the monitor and the network (color figure online)

### 7.3 Network retraining to improve monitor performance

To evaluate the effectiveness of the procedure proposed in Sect. 6, we run two experiments on several datasets (MNIST, FMNIST, GTSRB). In the first experiment, we assess the online learning with a monitor that uses the network retraining based on the distance loss. Table 5 puts the monitoring precision into perspective compared to the results reported in Table 2. As we can see, the monitoring precision is at least as good as for the quantitative monitor without the distance-loss retraining, and usually slightly better. Although the improvement does not seem significant, we see a strong improvement in detecting novelties in the GTSRB dataset, where 42 classes are learned; this is five classes more than the second-best approach.

In the second experiment, we consider the offline setting and investigate the accuracy, the TPR, and the FPR of the monitor, as well as the accuracy of the network, and compare those metrics to the appropriate base cases. That is, for each experiment, we train two networks for an equivalent number of epochs. The first network is trained using only the standard loss $\mathcal{L}_N$ for all epochs and serves as the base line for our comparisons. The second network is trained as follows: For the first $\lfloor 20\% \rfloor$ of the total epochs, we use only the standard loss $\mathcal{L}_N$, then we use the custom loss $\mathcal{L}$ on the resulting pretrained network, and we freeze the monitor after $\lfloor 50\% \rfloor$ of the remaining epochs. We start with the standard loss $\mathcal{L}_N$ for the network to stabilize first. Initially, the network parameters are chosen randomly, and we cannot expect the monitor to yield any reasonable clusters and distances. Thus we want to first let the network learn how to classify correctly before we incorporate the other goal of monitor performance. We freeze the monitor half-way to help the network converge. Otherwise, the optimum is a moving target because the loss depends on the monitor. We trained each network for a total of 16 epochs using a network accuracy threshold of 90%, i.e., $s^*_{lba} = 0.9$. Specifically, we consider six different settings for each dataset. Namely, the network is trained on two classes (1), on 50% of the classes (2), or on all but one class (3) and, in addition to the known classes, encounters a single unknown class ($a$), 50% of all classes ($b$), or all classes ($c$). These settings control how often an unknown class is encountered at run-time. Some of the combinations coincide ($2b$ and $2c$, $3a$ and $3c$) or are not possible ($3b$).

Across almost all experiments, we observe an improvement of the monitor accuracy, whereas the network accuracy remains comparable to the base case. The monitor accuracy improves mainly when there are multiple novel classes ($1b$, $1c$, $2c$), i.e., a single novel class can typically also be

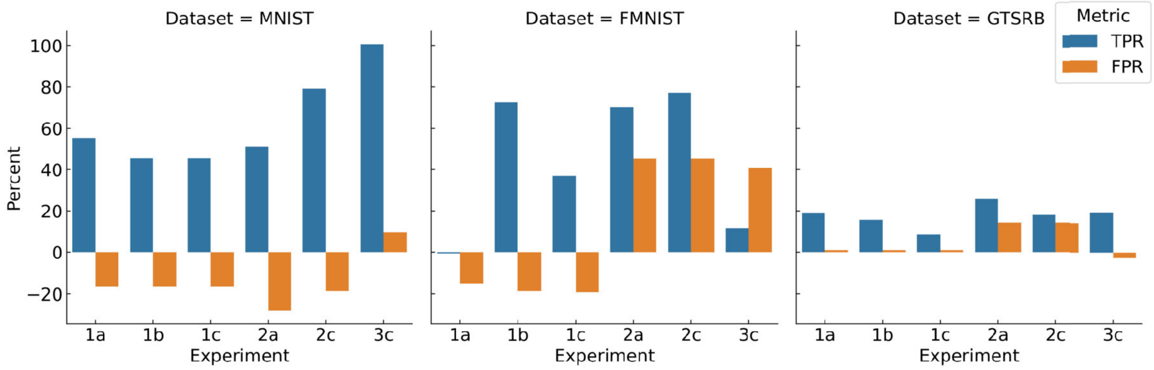
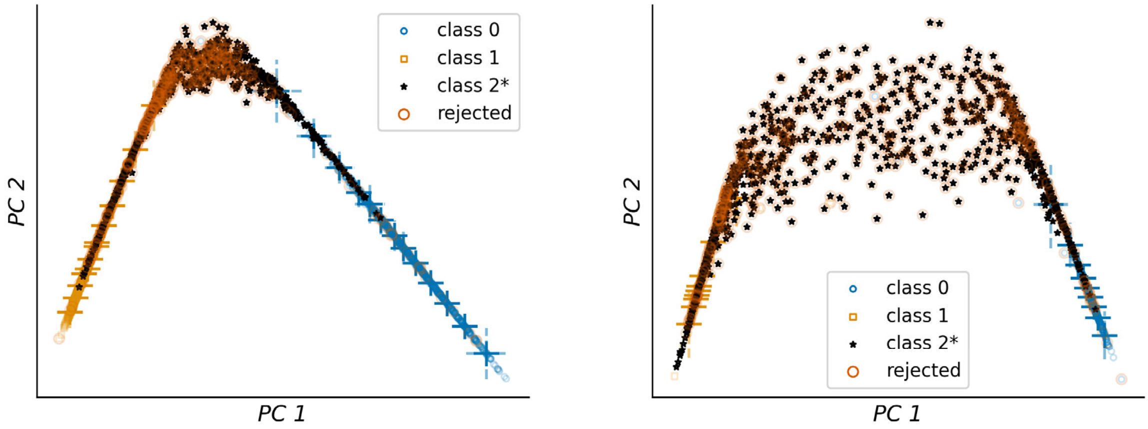


**Fig. 8** Relative change in the TPR and FPR of the monitor (color figure online)



(a) Trained with standard loss $\mathcal{L}_{\mathcal{N}}$.

(b) Trained with custom loss $\mathcal{L}$.

**Fig. 9** Projection of the quantitative monitor and its detection results for a network trained on the first two classes of the MNIST dataset (see also Fig. 11)

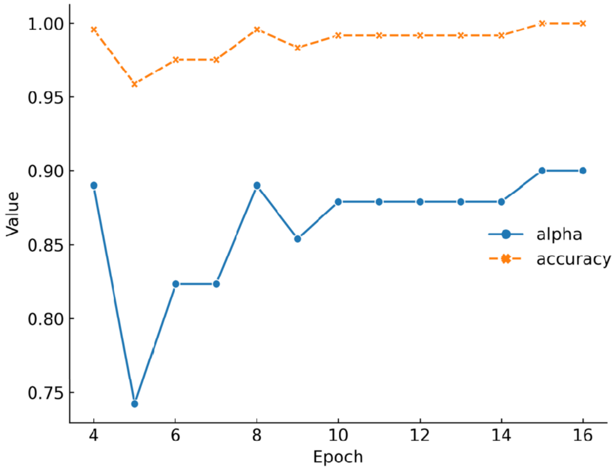
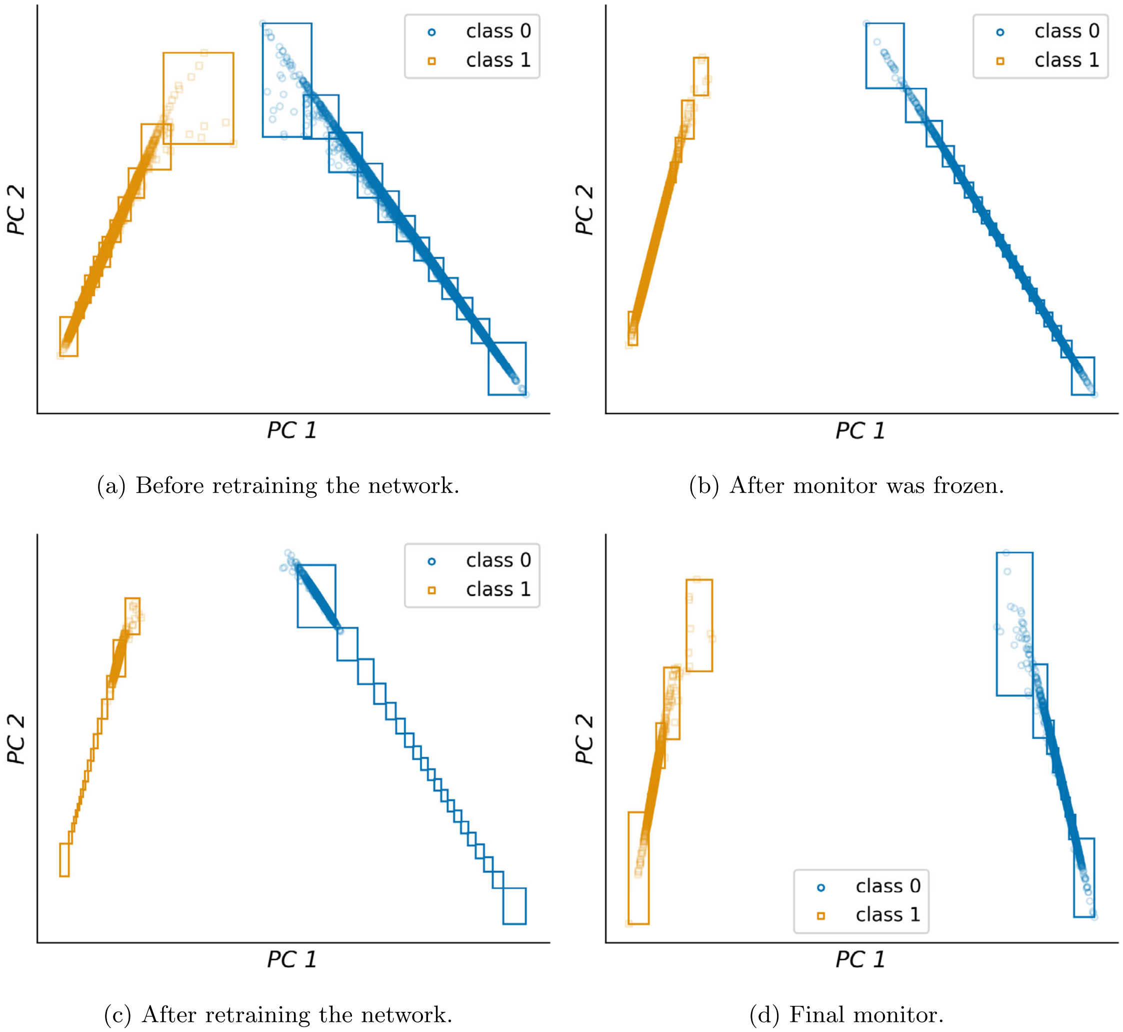


**Fig. 10** Adjustment of the weight-balancing parameter $\alpha$ during retraining on the GTSRB dataset with two known and one unknown classes

handled equally well without additional monitor training. In some cases, e.g., GTSRB, the network accuracy even improves compared to only using the custom loss (see Fig. 7). Moreover, we observe that the improvements in the monitor's accuracy are mostly attributed to an increase in the TPR. For MNIST in particular, the procedure also reduces the FPR, leading to an exceptional improvement in performance (see Fig. 8). For the other datasets, the FPR sometimes increases, which is generally expected. The new loss encourages tighter box distances per cluster, which may generally lead to more issued warnings. Still, with the exception of FMNIST in setting $3c$, the FPR increase is below the TPR increase.

We are able to retain a high network accuracy by automatically scaling $\mathcal{L}_d$ to the same magnitude of $\mathcal{L}_{\mathcal{N}}$ and carefully controlling the impact of $\mathcal{L}_d$ on $\mathcal{L}$. This is accomplished by adjusting the parameter $\alpha$ to compensate for drops in the network accuracy. Figure 10 demonstrates how the parameter $\alpha$ responds to fluctuations in the accuracy. As can be seen, the accuracy drops from nearly 100% to 95%, which triggers a reduction of $\alpha$ and thus a shift toward $\mathcal{L}_{\mathcal{N}}$ in $\mathcal{L}$.



(a) Before retraining the network.

(b) After monitor was frozen.

(c) After retraining the network.

(d) Final monitor.

**Fig. 11** Illustration of the monitor development (depicted with boxes) during training on MNIST with two known and one unknown classes, on a projection on the first two principal components $PC$ 1 and $PC$ 2 of the feature layer $\ell$

After three epochs, the network accuracy recovers, and $\alpha$ is increased again. Afterward, the network accuracy remains close to 100%, and correspondingly $\alpha$ keeps a high value of around 90%.

The purpose of retraining with our custom loss was to teach the network to shrink the spread within the clusters of the known classes in the feature space. In our experiments, we observe that the networks trained with our custom loss not only cluster inputs of the same class closer together, but also position those clusters further apart. An example is shown in Fig. 9.

Moreover, in Fig. 11, we illustrate that even after the monitor is frozen, the network keeps pushing inputs of the same class closer together in the feature space, resulting in several empty boxes (former data clusters). In the last plot, we show the monitor constructed for the final neural network, for which new clusters are computed.

## 8 Conclusion and future work

In this work, we introduced a new quantitative monitor that expresses its own confidence about the reported warnings based on a distance to the predicted class in feature space. We showed the benefits of the monitor in two different settings. In the first setting, we presented an active monitoring framework for accompanying a neural-network classifier during deployment. The framework adapts to unknown input classes via interaction with a human authority. Experiments on a diverse set of image-classification benchmarks showed that active monitoring is effective in improving accuracy over time in the setting when inputs of novel classes are frequently encountered. In comparison to alternative monitoring strategies, our monitor demonstrated superior performance in detection and adaptation at run-time. In the second setting, we used the quantitative feedback to retrain the neural network for reducing the distance on the training dataset. We showed experimentally that this significantly improves the monitor's performance in novelty detection.

Our approach is independent of the choice of the dataset and the neural-network architecture. The only requirement for applicability is access to the output of the feature layer(s). We plan to extend our procedure toward real-world applications with particular need of active monitoring, e.g., in robotics for the trained controller to gradually adapt to the behavior of the authority. Other interesting directions are time-critical applications where the adaptation of the monitor or the neural network need to be delayed to uncritical phases, and scenarios where novel inputs occur rarely. In addition, the underlying method of our framework can serve as a suitable tool for explainability of a neural network's predictions.

**Author Contribution** Konstantin Kueffner, Anna Lukina and Christian Schilling contributed equally to this work.

**Funding** This work was supported in part by the ERC-2020-AdG 101020093, by DIREC - Digital Research Centre Denmark, and by the Villum Investigator Grant S4OS.

## References

1. Bendale, A., Boult, T.E.: Towards open world recognition. In: CVPR, pp. 1893–1902. IEEE Comput. Soc., Los Alamitos (2015). https://doi.org/10.1109/CVPR.2015.7298799
2. Bendale, A., Boult, T.E.: Towards open set deep networks. In: CVPR, pp. 1563–1572. IEEE Comput. Soc., Los Alamitos (2016). https://doi.org/10.1109/CVPR.2016.173
3. Bendre, N., Terashima-Marín, H., Najafirad, P.: Learning from few samples: a survey (2020). https://arxiv.org/abs/2007.15484. arXiv:2007.15484. CoRR
4. Bottou, L.: Large-scale machine learning with stochastic gradient descent. In: COMPSTAT, pp. 177–186. Physica-Verlag, Heidelberg (2010). https://doi.org/10.1007/978-3-7908-2604-3_16
5. Chen, Y., Cheng, C., Yan, J., et al.: Monitoring object detection abnormalities via data-label and post-algorithm abstractions (2021). https://arxiv.org/abs/2103.15456. arXiv:2103.15456. CoRR
6. Cheng, C., Nührenberg, G., Yasuoka, H.: Runtime monitoring neuron activation patterns. In: DATE, IEEE, Florence, Italy, pp. 300–303 (2019). https://doi.org/10.23919/DATE.2019.8714971
7. Cohen, G., Afshar, S., Tapson, J., et al.: EMNIST: extending MNIST to handwritten letters. In: IJCNN, pp. 2921–2926. IEEE, Anchorage, AK, USA (2017). https://doi.org/10.1109/IJCNN.2017.7966217
8. Cohn, D.A., Atlas, L.E., Ladner, R.E.: Improving generalization with active learning. Mach. Learn. **15**(2), 201–221 (1994). https://doi.org/10.1007/BF00993277
9. Das, S., Wong, W., Dietterich, T.G., et al.: Incorporating expert feedback into active anomaly discovery. In: ICDM, pp. 853–858. IEEE Comput. Soc., Los Alamitos (2016). https://doi.org/10.1109/ICDM.2016.0102
10. Fan, J., Li, W.: Adversarial training and provable robustness: a tale of two objectives. In: AAAI, pp. 7367–7376. AAAI Press, Menlo Park (2021). https://ojs.aaai.org/index.php/AAAI/article/view/16904
11. Gal, Y., Ghahramani, Z.: Dropout as a Bayesian approximation: representing model uncertainty in deep learning. In: ICML, JMLR Workshop and Conference Proceedings, vol. 48. JMLR.org, New York, NY, USA, pp. 1050–1059 (2016). http://proceedings.mlr.press/v48/gal16.html
12. Geifman, Y., El-Yaniv, R.: Selective classification for deep neural networks. In: NeurIPS, pp. 4878–4887 (2017). http://papers.nips.cc/paper/7073-selective-classification-for-deep-neural-networks
13. Griewank, A., Walther, A.: Evaluating Derivatives - Principles and Techniques of Algorithmic Differentiation. SIAM, Philadelphia (2008). https://doi.org/10.1137/1.9780898717761
14. Guerriero, S., Caputo, B., Mensink, T.: DeepNCM: deep nearest class mean classifiers. In: ICLR. OpenReview.net (2018). https://openreview.net/forum?id=rkPLZ4JPM
15. Guo, C., Pleiss, G., Sun, Y., et al.: On calibration of modern neural networks. In: ICML, PMLR, vol. 70. PMLR, Sydney, Australia, pp. 1321–1330 (2017). http://proceedings.mlr.press/v70/guo17a.html
16. Gupta, A., Carlone, L.: Online monitoring for neural network based monocular pedestrian pose estimation. In: ITSC, pp. 1–8. IEEE, Rhodes, Greece (2020). https://doi.org/10.1109/ITSC45102.2020.9294609
17. Hashemi, V., Kretínský, J., Mohr, S., et al.: Gaussian-based runtime detection of out-of-distribution inputs for neural networks. In: RV, LNCS, vol. 12974, pp. 254–264. Springer, Berlin (2021). https://doi.org/10.1007/978-3-030-88494-9_14
18. Hendrycks, D., Gimpel, K.: A baseline for detecting misclassified and out-of-distribution examples in neural networks. In: ICLR. OpenReview.net (2017). https://openreview.net/forum?id=Hkg4TI9xl
19. Henzinger, T.A., Lukina, A., Schilling, C.: Outside the box: abstraction-based monitoring of neural networks. In: ECAI, Frontiers in Artificial Intelligence and Applications, vol. 325, pp. 2433–2440. IOS Press, Amsterdam (2020). https://doi.org/10.3233/FAIA200375
20. Ibrahim, S.H., Nassar, M.: Hack the box: Fooling deep learning abstraction-based monitors (2021). https://arxiv.org/abs/2107.04764. arXiv:2107.04764. CoRR
21. Jolliffe, I.T.: Principal Component Analysis. Springer Series in Statistics. Springer, Berlin (1986). https://doi.org/10.1007/978-1-4757-1904-8
22. Knorr, E.M., Ng, R.T.: A unified notion of outliers: properties and computation. In: KDD, pp. 219–222. AAAI Press, Menlo Park (1997). http://www.aaai.org/Library/KDD/1997/kdd97-044.php
23. Krizhevsky, A.: Learning multiple layers of features from tiny images (2009). Tech. Rep. http://www.cs.Toronto.edu/~kriz/learning-features-2009-TR.pdf
24. LeCun, Y., Bottou, L., Bengio, Y., et al.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)

25. Liu, W., Wang, Z., Liu, X., et al.: A survey of deep neural network architectures and their applications. Neurocomputing **234**, 11–26 (2017). https://doi.org/10.1016/j.neucom.2016.12.038
26. Lloyd, S.P.: Least squares quantization in PCM. IEEE Trans. Inf. Theory **28**(2), 129–136 (1982). https://doi.org/10.1109/TIT.1982.1056489
27. Lu, J., Gong, P., Ye, J., et al.: Learning from very few samples: a survey (2020). https://arxiv.org/abs/2009.02653. arXiv:2009.02653. CoRR
28. Lukina, A., Schilling, C., Henzinger, T.A.: Into the unknown: active monitoring of neural networks. In: RV, LNCS, vol. 12974, pp. 42–61. Springer, Berlin (2021). https://doi.org/10.1007/978-3-030-88494-9_3
29. Mancini, M., Karaoguz, H., Ricci, E., et al.: Knowledge is never enough: towards web aided deep open world recognition. In: ICRA, pp. 9537–9543. IEEE, Montreal, QC, Canada (2019). https://doi.org/10.1109/ICRA.2019.8793803
30. Mandelbaum, A., Weinshall, D.: Distance-based confidence score for neural network classifiers (2017). http://arxiv.org/abs/1709.09844. arXiv:1709.09844. CoRR
31. McCloskey, M., Cohen, N.J.: Catastrophic interference in connectionist networks: the sequential learning problem. In: Psychology of Learning and Motivation, vol. 24, pp. 109–165. Elsevier, Amsterdam (1989). http://www.sciencedirect.com/science/article/pii/S0079742108605368
32. Mensink, T., Verbeek, J.J., Perronnin, F., et al.: Distance-based image classification: generalizing to new classes at near-zero cost. IEEE Trans. Pattern Anal. Mach. Intell. **35**(11), 2624–2637 (2013). https://doi.org/10.1109/TPAMI.2013.83
33. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Trans. Knowl. Data Eng. **22**(10), 1345–1359 (2010). https://doi.org/10.1109/TKDE.2009.191
34. Parisi, G.I., Kemker, R., Part, J.L., et al.: Continual lifelong learning with neural networks: a review. Neural Netw. **113**, 54–71 (2019). https://doi.org/10.1016/j.neunet.2019.01.012
35. Pimentel, M.A.F., Clifton, D.A., Clifton, L.A., et al.: A review of novelty detection. Signal Process. **99**, 215–249 (2014). https://doi.org/10.1016/j.sigpro.2013.12.026
36. Rahman, Q.M., Corke, P., Dayoub, F.: Run-time monitoring of machine learning for robotic perception: a survey of emerging trends. IEEE Access **9**, 20,067–20,075 (2021). https://doi.org/10.1109/ACCESS.2021.3055015
37. Rebuffi, S., Kolesnikov, A., Sperl, G., et al.: iCaRL: incremental classifier and representation learning. In: CVPR, pp. 5533–5542. IEEE Comput. Soc., Los Alamitos (2017). https://doi.org/10.1109/CVPR.2017.587
38. Redko, I., Morvant, E., Habrard, A., et al.: Advances in Domain Adaptation Theory. Elsevier, Amsterdam (2019). https://doi.org/10.1016/C2016-0-05108-2
39. Royer, A., Lampert, C.H.: Classifier adaptation at prediction time. In: CVPR, IEEE Comput. Soc., Los Alamitos, pp. 1401–1409 (2015). https://doi.org/10.1109/CVPR.2015.7298746
40. Schölkopf, B., Smola, A.J., Müller, K.: Kernel principal component analysis. In: ICANN, LNCS, vol. 1327, pp. 583–588. Springer, Berlin (1997). https://doi.org/10.1007/BFb0020217
41. Schultheiss, A., Käding, C., Freytag, A., et al.: Finding the unknown: novelty detection with extreme value signatures of deep neural activations. In: GCPR, LNCS, vol. 10496, pp. 226–238. Springer, Berlin (2017). https://doi.org/10.1007/978-3-319-66709-6_19
42. Settles, B.: Active Learning, Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan Kaufmann, San Mateo (2012). https://doi.org/10.2200/S00429ED1V01Y201207AIM018
43. Stallkamp, J., Schlipsing, M., Salmen, J., et al.: The German traffic sign recognition benchmark: a multi-class classification competition. In: IJCNN, pp. 1453–1460. IEEE, San Jose, CA, USA (2011). https://doi.org/10.1109/IJCNN.2011.6033395
44. Sun, R., Lampert, C.H.: Ks(conf): a light-weight test if a multiclass classifier operates outside of its specifications. Int. J. Comput. Vis. **128**(4), 970–995 (2020). https://doi.org/10.1007/s11263-019-01232-x
45. Tan, C., Sun, F., Kong, T., et al.: A survey on deep transfer learning. In: ICANN, LNCS, vol. 11141, pp. 270–279. Springer, Berlin (2018). https://doi.org/10.1007/978-3-030-01424-7_27
46. Tobin, J., Fong, R., Ray, A., et al.: Domain randomization for transferring deep neural networks from simulation to the real world. In: IROS, pp. 23–30. IEEE, Vancouver, BC, Canada (2017). https://doi.org/10.1109/IROS.2017.8202133
47. Wagstaff, K.L., Lu, S.: Efficient active learning for new domains (2020). In: Workshop on real world experiment design and active learning
48. Wu, C., Falcone, Y., Bensalem, S.: Customizable reference runtime monitoring of neural networks using resolution boxes (2021). https://arxiv.org/abs/2104.14435. arXiv:2104.14435. CoRR
49. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms (2017). http://arxiv.org/abs/1708.07747. arXiv:1708.07747. CoRR
50. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: ECCV, LNCS, vol. 8689, pp. 818–833. Springer, Berlin (2014). https://doi.org/10.1007/978-3-319-10590-1_53
51. Zhang, P., Wang, J., Farhadi, A., et al.: Predicting failures of vision systems. In: CVPR, pp. 3566–3573. IEEE Comput. Soc., Los Alamitos (2014). https://doi.org/10.1109/CVPR.2014.456
52. Zhang, X., Zou, J., He, K., et al.: Accelerating very deep convolutional networks for classification and detection. IEEE Trans. Pattern Anal. Mach. Intell. **38**(10), 1943–1955 (2016). https://doi.org/10.1109/TPAMI.2015.2502579
53. Zhang, Z., Wu, P., Chen, Y., et al.: Out-of-distribution detection through relative activation-deactivation abstractions. In: ISSRE, pp. 150–161. IEEE, Wuhan, China (2021). https://doi.org/10.1109/ISSRE52982.2021.00027
54. Zhao, P., Hoi, S.C.H.: OTL: a framework of online transfer learning. In: ICML, Omnipress, Haifa, Israel pp. 1231–1238 (2010). https://icml.cc/Conferences/2010/papers/219.pdf

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.