Information Fusion 72 (2021) 48–69

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

Full length article

On learning effective ensembles of deep neural networks for intrusion detection

F. Folino, G. Folino *, M. Guarascio, F.S. Pisani, L. Pontieri

Institute of High Performance Computing and Networking (ICAR-CNR), Via P. Bucci, 87036 Rende (CS), Italy

ARTICLE INFO

Keywords:
Ensemble learning
Deep learning
Intrusion Detection Systems

ABSTRACT

Classification-oriented Machine Learning methods are a precious tool, in modern Intrusion Detection Systems (IDSs), for discriminating between suspected intrusion attacks and normal behaviors. Many recent proposals in this field leveraged Deep Neural Network (DNN) methods, capable of learning effective hierarchical data representations automatically. However, many of these solutions were validated on data featuring stationary distributions and/or large amounts of training examples. By contrast, in real IDS applications different kinds of attack tend to occur over time, and only a small fraction of the data instances is labeled (usually with far fewer examples of attacks than of normal behavior). A novel ensemble-based Deep Learning framework is proposed here that tries to face the challenging issues above. Basically, the non-stationary nature of IDS log data is faced by maintaining an ensemble consisting of a number of specialized base DNN classifiers, trained on disjoint chunks of the data instances' stream, plus a combiner model (reasoning on both the base classifiers predictions and original instance features). In order to learn deep base classifiers effectively from small training samples, an ad-hoc shared DNN architecture is adopted, featuring a combination of dropout capabilities, skip-connections, along with a cost-sensitive loss (for dealing with unbalanced data). Tests results, conducted on two benchmark IDS datasets and involving several competitors, confirmed the effectiveness of our proposal (in terms of both classification accuracy and robustness to data scarcity), and allowed us to evaluate different ensemble combination schemes.

1. Introduction

Timely recognizing security breaches and attacks is a task of utmost importance these days, owing to the high risk of cyber-security threats to which most modern companies and organizations are exposed daily. *Intrusion Detection Systems* (*IDSs*), i.e., hardware and software systems able to identify malicious behaviors are widely reckoned as a precious tool in this scenario, in order to leverage the large amounts of streaming log data concerning network connections and/or computer/application usage.

In the last few decades, there has been a growing interest towards classification-based *Machine Learning* (*ML*) techniques, owing to their capability to learn general models of malicious behaviors, while curbing the number of false alarms. As discussed in [1], the research in this area has been dominated for years by the usage of shallow learning techniques, which however typically need high levels of human-expert interventions, especially in the preparation of the training data through suitable feature-engineering activities. These activities are clearly intensive and expensive, and are strongly determined by the levels of data-analytics skill and of domain knowledge possessed by these experts.

This explains why recent advances in the ML field have paid increasing attention to *Deep Neural Networks* (*DNNs*), which have been shown able, in diverse application domains, to learn effective hierarchical feature representations of the data automatically. This capability to deal adequately with "raw" high-dimensional representations of the data becomes, indeed, particularly useful when working with the logs of complex system networks, as noticed in [1]. In fact, in a recent comparative analysis conducted in [2] over several learning-based IDS approaches, DNN-based methods were empirically shown capable of ensuring better detection accuracy than traditional supervised classification methods, across a wide range of attack types. On the other hand, thanks to the availability of high-performance computing resources and of software libraries providing various template DNN components, the analyst is allowed to develop and test easily more complex (and deeper) neural classification models than in the past.

* Corresponding author.
E-mail addresses: francesco.folino@icar.cnr.it (F. Folino), gianluigi.folino@icar.cnr.it (G. Folino), massimo.guarascio@icar.cnr.it (M. Guarascio), francesco.pisani@icar.cnr.it (F.S. Pisani), luigi.pontieri@icar.cnr.it (L. Pontieri).

https://doi.org/10.1016/j.inffus.2021.02.007
Received 13 May 2020; Received in revised form 17 December 2020; Accepted 7 February 2021
Available online 12 February 2021

1.1. Challenges and limitations of state-of-the-art solutions

Unfortunately, most of the DNN-based approaches to the discovery of intrusion–detection classifiers (implicitly) assume that the behaviors under analysis are stationary, and hence they cannot deal with changes in the distribution underlying data. Moreover, many of the solutions proposed in the literature were only validated on datasets featuring large enough amounts of labeled training examples.

By contrast, in most real-life intrusion–detection scenarios, only a small fraction of the log data instances is labeled (with far fewer examples of attacks than normal behavior) and, more importantly, different kinds of concept drifts are very likely to occur over the time, mainly owing to the surge of totally new attack types or of variants of old attack types that had not been occurring for some time. In particular, the latter kind of (recurrent) drift makes the idea of just resorting to an incremental learning scheme unsuitable — see Section 2.2 for a deeper discussion of critical issues affecting the discovery on an intrusion-oriented classifier.

A possible way to deal with these challenging issues (and in particular with non-stationary data) is to adopt an ensemble-learning strategy, as discussed in [3]. As a matter of fact, ensemble-based classifiers (a.k.a. Multiple Classifier Systems) are a powerful kind of hybrid intelligent systems for information fusion that enjoy several valuable features [4,5]: robustness to noise, capability to neatly improve weak classifiers, scalability (amenable to parallel/distributed implementations), incrementality. In particular, the latter feature has been recently exploited to deal with non-stationary data [6,7], by mainly making the ensemble include and harmonize models trained over different data chunks, in a way that different kinds of concept drifts (including recurrent ones) can be handled effectively.

However, hybridizing such an ensemble learning scheme with DNN base models is not straightforward, since dividing the input data stream into chunks further reduces the number of labeled samples available for training each DNN classifier — hence exacerbating the risk of discovering overfitting base classifiers, which is indeed high when training a model with a lot of parameters on few examples.

On the other hand, since the attack class mixes up different attack types/modes that change over time, each data chunk hardly covers this class as a whole, so that any classifier induced from this chunk will only play as a specialized predictor (which may return unreliable predictions on test instances that are "outside the scope" of the chunk). This makes the simple combination schemes unsuitable, such as weighted voting/averaging, adopted in most of the DNN-based ensemble approaches to IDS.

To the best of our knowledge, a limited number of works proposed the usage of Deep Ensemble Learning strategies for IDSs, and none of them addressed the problem of learning from non-stationary data and low amounts of labeled samples, and that of combining highly specialized base classifiers.

1.2. Proposed approach and contribution

In order to face the challenging issues mentioned above and overcome the limitation of state-of-the-art approaches to the induction of IDS-oriented classifiers, an incremental ensemble-based deep learning framework is proposed in this work, which leverages and extends the chunk-based induction strategy adopted in [6] and in other approaches (e.g., some of those overviewed in [7]).

The proposed framework essentially induces and combines a number of weak (specialized) base DNN classifiers on different data chunks, resulting from a temporal segmentation of the input stream, as a way to deal with the non-stationary nature of IDS log data.

In order to better face the small and imbalanced nature of the training data in IDS scenarios, the DNN architecture of the base classifiers has been designed to include both a combination of dropout layers and residual-like connections. Moreover, these classifiers are trained by using an ad-hoc cost-sensitive (imbalance-aware) loss function.

In addition, we define and study different combiner schemes for fusing the base classifiers in the ensemble:

- several alternative trainable models, including an adaptation of the classical Mixture-Of-Experts [8] (named *ensemble_moe*) and two variants (named *ensemble_feature* and *ensemble_stack*) of stacked generalization, all aimed at discovering some flexible and adaptive combination scheme for computing the final classification of any new instance, based on both the predictions of the (maybe very heterogeneous) base classifiers and on the raw data features of the instance itself;
- a cheaper non-trainable combination scheme (named *ensemble_max*) that simply assigns any new instance to the class that received the highest membership probability from all of the base models.

To the best of our knowledge, there has been no previous attempt to combine a chunk-based learning scheme with the discovery of DNN ensembles, while suitably dealing with the many challenging issues mentioned above. And yet, in our opinion, this was a promising research topic (as confirmed by our test results), considering the fact that deep learning methods have been proven very effective in classifying high-dimensional data (like those arising in network-traffic logs [9]), while chunk-wise ensemble learning was shown capable to effectively and efficiently support classification tasks over non-stationary data [6,7].

As a further contribution, we also discuss the results of an extensive experimentation conducted, on two benchmark IDS datasets, to assess the validity of the proposed framework, compare it with a number of competitors and study the effect of using different combination schemes. Besides allowing us to assess the feasibility and validity of our idea of hybridizing chunk-wise ensemble learning and deep learning methods, our experimental findings let us be confident in the fact that our work could provide a basis for developing effective, versatile, robust and efficient enough intelligent systems for the analysis of IDSs logs. Notably, our experimental study also includes an empirical analysis of the intrusion-oriented classifiers performances when reducing consistently the amount of labeled data used for training them; to the best of our knowledge, such analysis has not been given adequate attention in previous DNN-based approaches to IDS.

1.3. Organization of the paper

The rest of the paper is structured as follows. Section 2 introduces the specific intrusion-oriented classification problem addressed in our work in a more precise and formal way, and discusses the main technical issues that make it hard to solve. Some relevant related work in the field is overviewed in Section 3. The proposed chunk-wise ensemble learning scheme is presented in Section 4, which also discusses the technical solutions that have been exploited to allow it address effectively the challenging issues described in the former section. Section 5 describes, in a detailed way, both the DNN architecture that we are proposing for the base classifiers of the ensemble, and different possible (NN-based) schemes for combining the predictions of the former classifiers. The experimental setting and findings are described and analyzed in Section 6. Finally, Section 7 presents some concluding remarks (including a discussion on the novelty and significance of our work) and a number of directions for future research.

2. Problem statement and challenging issues

2.1. Intrusion detection as a binary classification task

According to the usual setting of *Network Intrusion Detection Systems* (*NIDSs*), our work aims at supporting the recognition of intrusion

attacks in a computer network, based on traffic-log data.[1] Following a common approach in the literature [10], we specifically tackle the (binary) classification problem of deciding whether a given network *connection* (or network *flow*) is associated with an intrusion attack or not, based on a fixed-length representation of the connection itself, which may encode different parameters of the connection (e.g., its endpoints and communication protocol) and aggregate features summarizing the flow of data exchanged in the connection. In fact, even though fine-grained classification schemes with multiple attack classes could be used as well, such a holistic binary scheme somewhat curbs the main limitation of supervised-learning IDS approaches that descends from their need to be provided with a sufficient amount of training examples for each of the classes (see paragraph *Rationale* below).

Regarding the space X of network connections as conceptually partitioned into the two security-related classes of *normal connections* and *attack connections* (or simply *attacks*), we specifically want to discover a (fuzzy) *classification model* (or *classifier*) for discriminating between these two classes, from a given collection of example data instances (i.e. connection tuples), labeled each with its respective ground-truth class. Precisely, this discovery task amounts to inducing, from a given sample of labeled connections, a classification model encoding a function of the form $m : X \rightarrow [0, 1]$ which maps any (possibly non labeled) connection $x \in X$ to an anomaly-oriented score $m(x)$ representing a sort of (possibly not calibrated) estimate of the probability that x is (linked to) an intrusion attack.

Hereinafter, we assume that the reference hypothesis space of this inductive learning task consists of a specific kind of *Deep Neural Networks* (*DNNs*) and of suitable ensemble-like combinations of them, the architectures of which are discussed in full detail in Section 5.

Rationale. The general problem setting sketched above is similar to those considered in previous ML-based NIDS approaches, if abstracting from the peculiar choices that have been made here relatively to the representation of network traffic data and the number of attack classes to predict. In fact, our framework could be easily extended to work in multi-class classification settings and with finer-grain representations of network connections (see Section 7) —the latter extension mainly entails equipping each classifier with a suitable embedding layer.

However, trying to discriminate between different attack classes may be impractical in many real-life intrusion detection scenarios, as it would require experts (or automatic labeling mechanisms) capable of providing a sufficiently large and updated number of correctly labeled examples for each of the different attack classes and for the normal one. This requirement is far stronger (and hardly feasible in many real-life NIDS settings) than just having training examples classified in a binary fashion (i.e. as either normal or attack), seeing as most connections associated with security attacks can be recognized on the basis of the damages that they have originated over the time — even though it is not always easy to assign such insecure connections to one among a given collection of attack (sub-)classes.

On the other hand, we believe that the choice of adopting a binary "attack-vs-normal" classification setting (hence renouncing to a fine-grain intrusion detection scheme that can directly reveal the type of attack associated with an insecure connection) helps soften the well-known incapability of multi-class classification approaches to intrusion detection to recognize a novel attack class until retraining the classifier with a sufficient number of examples of this class [10]. Indeed, a binary classifier has some chance of assigning the instances of an emerging attack type to the global class of attacks, to the extent that these novel instances share distinguishable patterns of deviance (w.r.t. normal behaviors) with the attack examples that were used to train the classifier.

[1] In fact, our approach can be also applied to other kinds of log data (such as the system/application logs usually considered Host-Based and Hybrid IDS). However, we here prefer to focus on network traffic data for the sake of concreteness and better readability.

2.2. Challenging issues

In many real-life intrusion–detection scenarios, the task of inducing a DNN-based classification model of the kind discussed above herein (i.e. a model for discriminating between normal connections and attacks) is hardened by a number of challenging issues, which are summarized below:

(I1) *Attack patterns change over the time.* It is natural to expect that novel types/modalities of attacks tend to continuously emerge over the time, and that the distribution of the attack class is non-stationary. This calls for devising a learning scheme that is robust to major changes of this distribution, a.k.a. *concept drifts*, which may severely undermine the accuracy of a classifier induced from historical training data. Moreover, attack patterns that have occurred in the past often come back, possibly with minor variations, after a relatively long period of absence. Such a form of *recurrent concept drift* makes purely incremental learning schemes unsuitable, owing to their limited memory capabilities. This limitation is particularly severe in the case of DNN models, which are very likely to incur in catastrophic forgetting phenomena [11] when trained in a purely incremental fashion — see Section 3.2 for more details in this respect. By contrast, chunk-based ensemble learning schemes [7] have proven to be an effective, efficient and versatile way to deal with non-stationary data and recurrent concept drifts. The core idea underlying these schemes is to segment the input stream of training examples into chunks, induce a separate base classifier, say BM_i, from each of the chunks, say D_i, and use a dynamical ensemble strategy to select and combine a subset of the discovered base classifiers. This processing flow, exploited and extended in our framework, is sketched in Fig. 1 and discussed in Section 4.1.

(I2) *Scarcity of training data* Even though labeling sample connections as either *normal* or *attack* is easier than labeling them according to a multi-class classification scheme (with multiple attack classes), this annotation task may well be rather expensive in terms of required time and user skills. As a consequence, the fraction of novel connection instances that can be associated with a reliable class label (and hence regarded as fresh training examples) may be rather small. This clearly calls for devising a DNN architecture and training scheme for the base classifiers that can work well with small amounts of training data, while reducing the risk of incurring into overfitting, which is indeed very high when using few data to train a neural network with millions of parameters.

(I3) *Class imbalance* The difficulty of having (labeled) examples exacerbates for the class of the attacks, seeing as malicious network connections tend to occur relatively rarely, together with a far larger number of non-malicious connections. The imbalance between the classes of the normal connections and of the attacks is likely to reflect in the training dataset, featuring a wide majority of the examples of the former class.

(I4) *Chunk-derived classifiers likely focus on sub-regions of the attack class* Clearly, the data distribution of the attack class is expected to be both very heterogeneous and non-stationary, owing to the fact that it mixes up different attack types/modes, and that the latter change over the time. In particular, as noticed in [12] and reflected in many NIDS datasets [13], many types of intrusion attacks tend to manifest in the form of bursts of malicious network connections covering limited periods of time — consider, for example, the case of Brute-Force (SSH) attacks and (Distributed) Denial-of-Service attacks. Thus, each data chunk is very likely only to provide evidence for specific attack modes, and hardly covers the variety of all possible attack patterns. Consequently, the discovered base classifiers act as specialized predictors, which are good at recognizing the attack patterns covered by the respective data chunk, but may return unreliable predictions on

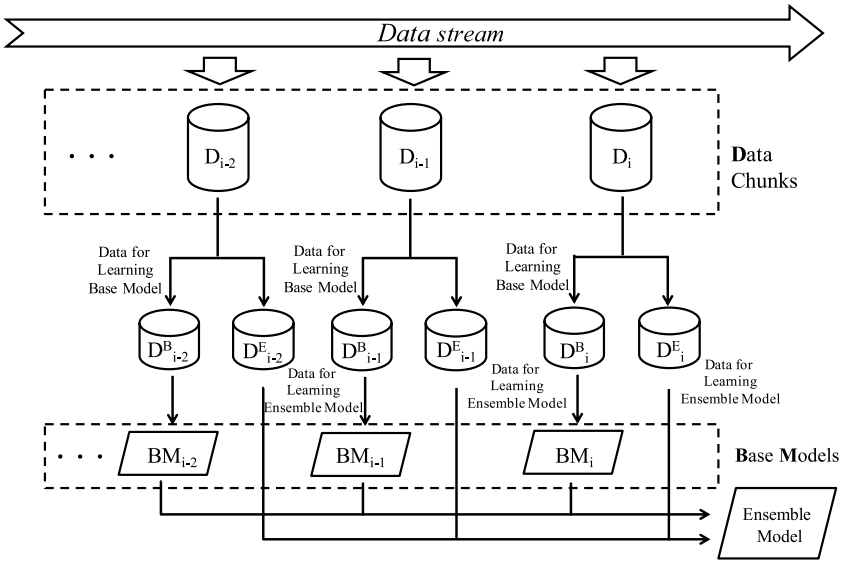


Fig. 1. Reference chunk-wise ensemble learning scheme: information flow.

instances that "are outside the scope" of the chunk on which they were trained. This issue calls for equipping the ensemble of chunk-derived classifiers with a *combiner* that can possibly account for the regions of competence of the classifiers when fusing the predictions that they return for a new data instance. Notably, such a capability of the combiner would become particularly useful in distributed IDS scenarios [5], where the specialized nature of the base classifiers also comes from the locality of the data used to train them.

3. Related work

Major ensemble-based approaches to development of Intrusion Detection Systems in the literature are discussed in what follows, in two separate subsections. The first subsection is devoted to illustrating those approaches combining multiple traditional (shallow) models and training algorithms developed in the field of Machine Learning (ML), whereas the latter subsection focuses on the usage of DNN's ensembles. Section 3.3 then presents a summary discussion on the gap among these state-of-the-art solutions and certain core challenging requirements (condensed in the four issues of Section 2.2) of real-life IDS applications.

3.1. Ensembles of ML models for IDS

In general, ensemble-based classifiers (a.k.a. Multiple Classifier Systems) constitute a form of hybrid intelligent systems for information fusion that enjoy several valuable features [4,5]: robustness to noise, capability to improve weak classifiers neatly, and scalability (amenable to parallel/distributed implementations). Moreover, when dealing with non stationary data, one can think of building the ensemble incrementally, as proposed in [6,7], by making it include and harmonize models trained over different data chunks, in such a way that different kinds of concept drifts (including recurrent ones) can be handled effectively.

In general, IDSs can benefit from using information fusion and, in particular, ensemble-based methods for a number of reasons [4,5]. First, such methods work well both even when a few data are available and when a huge number of data has to be analyzed; furthermore, they can be easily implemented so as to exploit the advantages of distributed environments such as parallel, GPGPU architectures, and P2P and Cloud computing architectures. In addition, they can model different abstractions or parts of the instance space and then combined together, to ensure better predictions.

The rest of this subsection briefly describes some major ensemble-based approaches to intrusion detection, which all rely on using shallow Machine Learning (ML) models and algorithms. Table 1 offers a summarized view of these approaches (and some more details on their implementation and experimental validation). More specifically, for each reviewed approach, the table reports: the kind of models employed and how they are combined (under columns *Combined Models* and *Combination Strategy*, respectively), the dataset(s) and evaluation metrics used for experimentally validating the approach, and three columns corresponding to the general issues I1, I2 and I3 defined in Section 2.2, respectively. The latter three columns are devoted to summarizing the features (if any) of the approach that allows it to cope with these challenging issues, which our research work is ultimately meant to address. Notice that issue I4 (concerning the need to combine specialized/local base models) has not been considered in this table, because it only concerns the (very few) approaches that train the base models with temporally/spatially defined partitions of the dataset — so that almost all the cells in the column would give no information.

Standard ensemble learning approaches rely on training multiple base classifiers and using an either fixed or trainable combination function to merge the predictions of these classifiers, on any new data instance x, into a final decision for x (this strategy is named *Predictions Fusion* in the table).[2]

Such a standard ensemble learning strategy is used in all the approaches reviewed here but the one by Costa et al. [14], where a hierarchical classification framework is defined that combines associative rule learning and probabilistic models according to a sort of *Cascade* strategy. Specifically, the approach in [14] first induces an ordered list of association-based classification rules (CARs) for splitting the instance space in (possibly overlapping) regions with class-homogeneous data instances. A probabilistic classifier is then induced for each rule, out of the training instances covered by the rule. The list of CARs and their associated probabilistic classifiers constitute a hierarchical classification

[2] Two very popular combination criteria are *majority voting* (x is assigned to the majority class among those predicted by the base classifiers), and associated weighted variants; *(Bayesian) averaging* (x is assigned to the class with the maximum average posterior probability, assuming that all the base classifiers can estimate the posterior class probabilities). *belief* (x is assigned to the data class with the maximum belief value, where per-class belief values are computed by estimating the probability that an instance assigned to a given class actually belongs to the class).

Table 1
Ensembles of ML models for IDS: main properties and mechanisms adopted to cope with the issues defined in Section 2.2. For the sake of comparison, the same kinds of information are also reported for our approach.

Approach	Combined models	Combined strategy	Dataset(s)	Evaluation metric(s)	Pre-processing	Changing & recurring behaviors (I1)	Scarcity of labeled data (I2)	Class imbalance (I3)
Costa et al. [14]	(Ordered list of) CAR classification rules + a probabilistic classifier per rule	Cascade training/classification: select the probabilistic model associated with the first rule covering the test instance	KDD'99	ACC, AUC	Standard preprocessing	Not addressed	Not addressed	CARs are found with a sequential covering algorithm and correlation-based selection criterion
Borji [15]	ANN, SVM, DT and k-NN classifiers	Predictions fusion via majority voting, Bayesian averaging or belief rule	DARPA'98	DR, FAR	Standard preprocessing	Not addressed	Not addressed	Not addressed
Sivatha Sindhu et al. [16]	Multiple shallow NNs (trained via bagging) + DT classifier	Hybrid: Predictions' fusion via majority voting for combining the NNs, and DT training on NN ensemble's output	KDD'99	DR, FAR, TP, TN, FP, FN	Dimensionality reduction via genetic algorithm	Not addressed	Data augmentation before training the final DT classifier (randomly generated examples, labeled with the NN ensemble)	A class-weighted error is used in the training procedure[a]
Folino et al. [17]	J48, JRIP, NBTree, NB, 1R, LMT, LogR, DS and 1BK classifiers, all trained on temporal data chunks	Predictions fusion through a "meta-learnt" GP-tree combiner	ISCX IDS	AUC, AUC-PR	Standard preprocessing	Temporal ensemble (base models trained on different data chunks)	Not addressed	Imbalance-aware fitness metric used in the search for an optimal combiner
Zhou et al. [18]	C4.5, RF and Forest PA (penalizing attributes) classifiers	Predictions fusion via average-of-probabilities rule	NSL-KDD, AWID, CICIDS 2017	ACC, P, DR, F1, FAR	Bat-inspired procedure for selecting data features automatically	Not addressed	Not addressed	Not addressed
Our approach	Multiple DNNs trained on temporal data chunks	Predictions fusion via a trainable (MOE or FF) sub-net or max-probability rule	CICIDS 2017, ISCX IDS	AUC, AUC-PR, F1	Standard preprocessing	Temporal ensemble (base models trained on different data chunks)	Dropout layers and skip-through connections within each base model (providing additional ensemble-like capabilities)	Data resampling, imbalance-aware loss function

[a]In principle, the bagging procedure for learning the NN ensemble could be easily combined with a resampling mechanism, but this is not explored in the paper.

model, which allows for classifying each new instance x as follows: (i) first all the CARs are applied to x sequentially, until a rule r_x covering x is found; (ii) the class-membership probabilities of x are then computed by using the probabilistic classifier associated with r_x. Notably, the CAR rules are built up by using a variant (devised to retain only positively correlated patterns) of A-priori algorithm, combined with a sequential covering rule induction scheme. Both mechanisms should make the discovery of these rules pretty robust to class imbalance issues. Indeed, experiments on several datasets, including KDD'99, confirmed the ability of the approach to effectively recognize rare classes. In a sense, similarly to our setting, the probabilistic classifiers play as local models, trained each on a specific "region" of the instance space; however, there is no need to combine the predictions of these models, since any new instance is classified by only using one of them (chosen according to the CAR list).

All the works described in the following adopt the standard predictions fusion strategy of classical ensemble learning methods.

In particular, an ensemble of ANN, SVM, Decision Tree (DT) and k-NN (1 Nearest Neighbor) classifiers is defined in Borji et al. [15], for detecting attack instances in the DARPA dataset. Three alternative combination schemes are considered for integrating the base classifiers' output: Bayesian averaging, majority voting and belief rule.

Sivatha Sindhu et al. [16] adopt an ensemble of shallow neural networks, combined according to a voting mechanism. In order to select the optimal subset of the features from the dataset, a genetic-algorithm procedure is exploited. The trained ensemble is used to produce a refined version of the training set, which includes the original instances and a number of randomly generated ones (labeled all with the class predicted by the ensemble). This augmented training set is eventually employed to train a Decision Tree (DT) classifier, by using the gain-ratio variant of algorithm C4.5. data augmentation before training the final DT classifier (randomly generated examples, labeled with the NN ensemble). "A class-weighted error is used in the training procedure; in principle, the bagging procedure for learning the NN ensemble could be easily combined with a resampling mechanism, but this is not explored in the paper"

A further heterogeneous ensemble method for NIDSs is proposed by Zhou et al. [18]. The method consists in training different kinds of shallow classifiers, learnt by using algorithms C4.5, Random Forests (RF) and Forest Penalizing Attributes, and then employing an average-of-probabilities (AOP) rule for combining their predictions. It is worth noticing that, before training the ensemble, a heuristics (bat-inspired) feature selection procedure, named *CFS-BA*, is exploited to pre-process the given dataset optimally. This specialized pre-processing capability seems to boost the performances of this method, which is empirically shown to obtain very good results over different benchmark datasets (cf. Table 1).

Folino et al. [19] describe a temporal ensemble approach to NIDS, where multiple heterogeneous base classifiers (see Table 1 for details on the base learners employed) are trained on different data chunks, similarly to our approach. An optimal combiner function (specifically, a GP-tree model assembling non-trainable aggregation functions) is computed, for fusing the base classifiers predictions, though a Genetic Programming procedure, which adopts a fitness measure defined in terms of the accuracy scores that a candidate solution achieves over the classes. Notably, these per-class accuracy scores are weighted differently in order to deal effectively with imbalanced classes (cf. issue I3, in Section 2.2) –as confirmed by experimental results obtained

on dataset ISCX IDS. An extended version of this work is presented in [17], which devises a continuous learning-and-classify method. As highlighted in Table 1, the use of a temporal segmentation strategy for dividing the training data among different base models, allows this approach possibly to deal with non-stationary data/class distributions. However, it is worth noticing that, when applied to a test instance x, the GP-tree meta-classifier discovered in [17] returns a classification decision that only depends on the predictions made for x by the base classifiers, without taking account of the intrinsic features of x itself. This does not allow the approach possibly to learn instance-adaptive combination functions, which may become very useful when using specialized/local base classifiers, which are very likely to occur when using temporally-segmented chunks of NIDS data (cf. issue I4, in Section 2.2).

3.2. Ensembles of DNN models for IDS

An emerging research line consists in leveraging Deep Learning (DL) architectures for intrusion detection. Basically, a Deep Neural Network (DNN) works in a hierarchical way: several layers of non-linear processing units are stacked one onto the other, so that each layer in the model generates features at a higher level of abstraction than the previous layer. Compared to classic ML techniques, these models enjoy several benefits [20], two of which look particularly interesting for the analysis of network traffic: (i) they can learn from raw traffic data with little/no manual feature engineering; (ii) they can be trained incrementally, by periodically using new batches of training data to update their weights [21].

Quite a complete and updated survey on existing DL-based approaches to IDSs can be found in [31]. In the rest of this section, we concentrate on those leveraging some kind of model ensemble/combination strategy, which look more closely related to the problem setting and proposal described in this manuscript.

A summary of the selected works is reported in Table 2, sharing the same structure as Table 1, which reports, for each approach, the kinds of DNN models and combination strategy employed, the dataset(s) and metrics used to validate the approach, and information on the features related to the issues I1, I2 and I3 of Section 2.2.

As concerns the kinds of base models exploited, it seems that the following neural architectures have been used the most in these IDS methods, in addition to feed-forward (FF) deep neural networks (simply referred to, in the table, as DNNs): Deep Belief Networks (DBN), Auto-Encoders (AEs), and Long Short Term Memory (LSTM) networks [32] and Extreme Learning Machines [33].

Three broad categories of combination strategies seem to have been used in the DNN-ensemble solutions that have been proposed in the field of IDS so far:

A. *Cascade training/classification* (used in [22,23,34]);
B. *Knowledge transfer and fine tuning* (used in [24–27]); and
C. *Predictions Fusion*, i.e. the standard approach to ensemble learning (used in [28–30]).

A. cascade-like approaches. The first category includes methods that exploit an unsupervised (or a self-supervised) neural model (typically a DBN or an AE) as a sort of feature extractor/learner, and train a shallow classification model by reusing some classic machine learning algorithm.

A pioneering approach of this kind in the field of IDSs is described in [22], which combines a DBN and an SVM, according to such a hybrid machine/deep learning scheme. The DBN is specifically meant to perform a dimensionality reduction task — which is empirically shown in the paper to improve classic PCA, gain ratio and chi-square methods — and eventually provide the SVM induction algorithm with effective (more general and meaningful) representations of the training data at hand. The SVM classifier discovered from the data transformed this way is expected, indeed, to obtain an SVM model capable to classify new data instances (once mapped onto the transformed space produced by DBN).

A conceptually similar hybrid approach to NIDSs is shown in [34], with a deep AE and a Random Forest (RF) classifier (in the places of the DBN and SVM classifier, respectively). More specifically, the deep AE takes the form of a stack of non symmetric deep AEs (i.e. auto-encoders featuring multiple non-symmetrical hidden layers), and it is still trained in an unsupervised way in order to capture useful (and general) data transformations. The representations of the training instances produced by this model (more precisely, by its final encoding layer) are then used (along with their associated class labels) to induce the RF component.

Mohammadi et al. [23] adopt a rather similar hybrid solution, by specifically using a stacked AE (featuring four hidden layers for both the encoder and decoder) to perform the representation learning (and dimensionality reduction) task. The transformed data are then passed to a Linear Classifier (LC), trained with an ad hoc Genetic Programming (precisely Memetic) algorithm.

B. transfer-and-finetune approaches. The proposals falling into this category still rely on exploiting an unsupervised NN model for learning low-dimensional and general enough data representations. However, the knowledge acquired by such an auxiliary model (typically an AE or DBN) is directly transferred into a DNN classifier, before fine tuning the latter, in a supervised way, over the labeled training instances available.

In particular, in [26], the auxiliary unsupervised learning task is delegated to a deep AE model, which consists of four stacked auto-encoders. The main classification task is performed instead with a DNN model, which includes a replica of the encoding structure of the deep AE model, topped with a softmax regression layer. Once trained, the DNN classifier can be directly applied to the original features of any new data instance x, in order to predict the vector of class membership probabilities for x.

Very similar architectures (a deep AE and a DNN with a number of shared layers) and training/classification schemes are used in [25] (which specifically adopts a 2-layer AE) and in [27], in order to detect network intrusions.

By contrast, a DBN and a paired DNN (including a final softmax layer) is employed in [24]. The former model (consisting of four Restricted Boltzmann Machines [32]) still plays as a feature learning (and dimensionality reduction) component, trained in an unsupervised way. The hidden weights of the DBN are used to initialize (pretrain) the DNN, which is fine-tuned in two steps (first only in its final classification layer, and then as a whole).

C. predictions fusion approaches. As highlighted in [35], although promising, DNN ensemble architectures have been under-exploited in the literature.

In [28] the output of several DNN approaches are combined to detect anomalous behavior occurring in computer networks. The final heterogeneous ensemble includes different kinds of base models: an AE-based DBN-based classifier (built according to a transfer-and-finetune strategy like those presented previously), a DNN classifier and an ELM classifier. An extensive experimentation proves that employing an ensemble-based strategy permits higher intrusion detection accuracy to be achieved than single-DNN classifiers.

A hybrid (shallow+deep) ensemble is proposed in [29], which includes a DNN and traditional ML models — namely, Decision Tree (DT), SVM classifier, Logistic Regression (LogR) and k-NN classifiers. An ad hoc algorithm (based on CART), named *MultiTree*, is proposed in the work as an additional base learner, which exploits data resampling mechanisms (helping cope with imbalanced classes) and eventually returns a DT classifier. An adaptive weighted voting scheme is used to derive an overall classification, for any new data instance, from the predictions made for the instance by these base models.

Table 2
Ensembles of DNN models for IDS: main properties and mechanisms adopted to cope with the issues defined in Section 2.2. For the sake of comparison, the same kinds of information are also reported for our approach.

Approach	Combined models	Combined strategy	Dataset(s)	Evaluation metric(s)	Pre-processing	Changing & recurring behaviors (I1)	Scarcity of labeled data (I2)	Class imbalance (I3)
Salama et al. [22]	DBN + SVM classifier	Cascade training/classification (DBN used as feature extractor)	NSL-KDD	ACC	Dimensionality reduction via DBNs	Not addressed explicitly and in full[a]	Not addressed explicitly and in full[b]	Not addressed
Shone et al. [1]	Deep AE + Random Forest (RF) classifier	Cascade training/classification (AE used as feature extractor)	KDD'99, NSL-KDD	ACC, P, R, F1, FAR	Standard preprocessing	Not addressed explicitly and in full[a]	Not addressed explicitly and in full[b]	Not addressed
Mohammadi and Namadchian [23]	Deep AE-based anomaly predictor + a Linear Classifier (LC)	Cascade training/classification: AE's predictions are taken as input by the LC	KDD'99, NSL-KDD	ACC	Dimensionality reduction via stacked AE	Not addressed explicitly and in full[a]	Not addressed explicitly and in full[b]	Not addressed
Alrawashdeh and Purdy [24]	DBN + DNN classifier (replicating the DBN's structure)	Knowledge transfer (DBN to DNN) and fine tuning	KDD'99	ACC, TP, TN, FP, FN	Standard preprocessing	Not addressed explicitly and in full[a]	Not addressed explicitly and in full[b]	Not addressed
Potluri and Diedrich [25]	Deep AE + DNN classifier (replicating the AE's encoder part)	Knowledge transfer (AE to DNN) and fine tuning	NLS-KDD	ACC	Standard preprocessing	Not addressed explicitly and in full[a]	Not addressed explicitly and in full[b]	Not addressed
Farahnakian and Heikkonen [26]	Deep AE + DNN classifier (replicating the AE's encoder part)	Knowledge transfer (AE to DNN) and fine tuning	KDD'99	ACC, DR, FAR	Dimensionality reduction via AE	Not addressed explicitly and in full[a]	Not addressed explicitly and in full[b]	Not addressed
Javaid et al. [27]	Deep AE + DNN classifier (replicating the AE's encoder part)	Knowledge transfer (AE to DNN) and fine tuning	NSL-KDD	ACC, P, R, F1	Standard preprocessing	Not addressed explicitly and in full[a]	Not addressed explicitly and in full[b]	Not addressed
Ludwig [28]	AE-based, DBN-based, DNN and ELM classifiers	Predictions Fusion via (weighted) majority voting	NSL-KDD	ACC, DR, FAR, AUC, F1	Dimensionality reduction via AE; Feature extraction via DBN	Not addressed explicitly and in full[a]	Not addressed explicitly and in full[b]	Not addressed
Gao [29]	DT, RF, kNN, DNN and "MultiTree" classifiers	Predictions Fusion via (class-wise) weighted majority voting	NSL-KDD	ACC, P, R, F1	Dimensionality reduction via PCA	Not addressed	Not addressed	Data resampling (only when learning the MultiTree model)
Zhong et al. [30]	DBN + AE-based anomaly predictor + LSTM anomaly predictor	Hybrid: Cascade training + Predictions Fusion via a trainable weighted averaging	MAWILab, CICIDS 2017	P, R, F1, FPR	Damped Incremental Statistics for feature extraction	Ensemble retrained periodically; however, only instances from last chunk are maintained	Partially addressed, labeled data used only to compute combination weights	Not addressed
Our approach	Multiple DNNs trained on temporal data chunks	Predictions fusion via a trainable (MOE or FF) sub-net or max-probability rule	CICIDS 2017, ISCX IDS	AUC, AUC-PR, F1	Standard preprocessing	Temporal ensemble (base models trained on different data chunks)	Dropout layers and skip-through connections within each base model (providing additional ensemble-like capabilities)	Data resampling, imbalance-aware loss function

[a]In principle, the models could be trained incrementally, but knowledge of past attack types may be lost.
[b]In principle, additional unlabeled data could be exploited in the unsupervised feature-learning step, to reduce overfitting risks, but such a semi-supervised learning scheme is not explored in the work.

Unsupervised DBN and AE models are combined with an LSTM predictor in the *HELAD* (*Heterogeneous Ensemble Learning Anomaly Detection*) NIDS system proposed in [30]. First, a DBN "feature extractor" and the AE are trained on the given network data, regarded each as a sequence of packet tuples. The LSTM model is then trained in a supervised way, after labeling each packet tuple with the (RMSE-based) anomaly score predicted by AE. A new instance x is classified as either anomalous or normal by comparing a fixed threshold with a weighted combination of the anomaly scores assigned to x by the AE and LSTM (so playing as two base models). Notably, the combination weights are learnt via a ("supervised") simulated annealing procedure, guided by manually labeled examples — this makes the approach fall in the realm of (semi-) supervised IDS solutions. The ensemble model is retrained periodically according to a fixed-window policy, where both manually labeled examples and automatically labeled ones (coming from the previous data window) are exploited to update the predictors combination weights. More details in this respect are provided in the following paragraph, which discusses some limitations of current DL-based approaches to IDS, relatively to the challenging issues I1, I2 and I3 (cf. Section 2.2).

3.3. Gap analysis

As summarized in Table 2, none of the approaches deals with the problem of learning from imbalanced data (issue I3), apart the one proposed in [29] –which just resorts to the use of a data resampling procedure, prior to training one of the base classifiers.

Moreover, the need of coping with changing/recurring behaviors (issue I1) and with the scarcity of labeled data (issue I2) is not solved explicitly and properly in the reviewed works. Indeed, as for issue I1, it is worth noticing that, in principle, all the NNs used in these approaches could be trained incrementally, by periodically exploiting novel batches of training data to update the weights of these NNs [21]. In other words the approaches falling in categories B and C can be easily turned into incremental learning ones, and the same might even be done with the approaches of category A —provided that an incremental algorithm is used to induce the shallow classifier (maybe in an approximated way).

However, performing online classification with incrementally trained NNs requires special attention in many non-stationary settings, seeing as common back-propagation algorithms (such as SGD and variations of it) were originally conceived to work on i.i.d. data instances (i.e. instances sampled uniformly from a given fixed distribution). In particular, as observed in [36], these algorithms tend to reinforce the data modalities (e.g., attack sub-types, in our setting) covered by the current batch, so that the NN is likely to forget those that are not (maybe temporarily) well represented in the last batch (or the last few batches). This phenomenon, known in the literature as *catastrophic forgetting* and well studied in Continual/Lifelong Learning systems [11]), exacerbates in IDS settings, where different attack subtypes tend to appear and disappear over time, as well as come back (maybe in a disguised/renewed form) after a relatively long period of absence. This difficulty to retain naturally relevant knowledge on past attack modes may explain the fact that almost none of the reviewed DL-based approaches to IDS have been conceived to work on streaming data.

The above observations provide a possible explanation to the fact that all the state-of-the-art DNN ensembles for IDS did not addressed explicitly issue I1, despite it being definitely crucial in real IDS scenarios. As discussed before, the only exception to this general pattern is the HELAD system [30], owing to its capability to periodically retrain the ensemble detector (or, at least, the combiner module of it), over new data chunks according to a variant of classical *landmark windowing* methods [37], which also exploits older labeled data, coming from the previous chunk and from the initial set of examples provided by the expert. However, we are afraid that this expedient does prevent losing relevant knowledge on behaviors that featured some chunks before, seeing as the underlying base models may well forget this knowledge in a few retraining steps.

Issue I2 as well is not addressed appropriately in the approaches of Table 2. And yet, the ones relying on a feature-learning step (i.e. the approaches of categories A and B) could be easily rephrased in a semi-supervised fashion, as to exploit larger amounts of unlabeled data (in this preliminary unsupervised step), when only few labeled data are available for the supervised training phase. Anyway, none of these approaches explores this possibility, nor other mechanisms for mitigating overfitting risks in the latter phase. We suppose that issue I2 should not impact severely on the HELAD method [30], since this method only requires labeled data to find optimal combination weights for its base anomaly predictors, while training these on either unlabeled data (the AE-based one) or artificially labeled data (the LSTM-based one). However, we cannot draw a definite conclusion in this respect, for no analysis of the sensitiveness of the method toward the scarcity of labeled data is conducted in [30].

As summarized in both Tables 1 and 2, the framework proposed in this paper is meant to address all of these issues, while also includes a number of ad hoc schemes for fusing the predictions of local/specialized classifiers (which is a specific issue of chunk-based ensembles like ours). In fact, the following technical aspects of the framework (devised rightly to cope with these issues) make it quite different from existing solutions in the field: the chunk-wise ensemble learning scheme used in the training of an IDS model, the peculiar DNN architecture (including both residual-like and dropout components) employed for the base classifiers, in addition to the above-mentioned combination schemes (in the place of usual averaging/voting mechanisms).

Details on these features are provided in the next section, while further remarks on the novelty of our proposal can be found in Section 7.2.

4. Proposed framework: ensemble learning scheme and design choices

The fundamental research question underlying our work can be summarized as follows: is it possible to devise suitable DNN architectures for implementing the learning infrastructure of a chunk-wise ensemble classification approach, in a way that the challenging issues stated in the previous subsection are addressed satisfactorily?

Armed with the desire to shed some light on this question, we here propose an ensemble-based framework for the binary classification problem above, where both the base classifiers and the combination logics rely on specific neural-network architectures defined here ad hoc.

Before illustrating the proposed DNN architecture in detail in a separate section (namely, Section 5), in what follows we first illustrate (in Section 4.1) a chunk-wise ensemble learning scheme for which our proposed framework is meant to provide a core methodological background, and then summarize (in Section 4.2) the main technical features of our framework that allow it deal appropriately with challenging issues presented in Section 2.2.

4.1. Chunk-wise ensemble learning scheme

As sketched in Fig. 1, our approach to the detection of network intrusions relies on building up a series of base DNN classifiers (denoted as $BM_1, BM_2, \ldots$) sharing the same DNN architecture (see Section 5). These classifiers are induced from disjoint *data chunks* (denoted as $D_1, D_2, \ldots$, respectively), which result from suitably partitioning the given training instances.

Different strategies have been proposed in the literature (see, e.g., [7]) for deriving the data chunks from temporally-marked data originated by a data stream. A very simple strategy, used to implement and test our approach, consists in applying a sliding window of a fixed temporal span (clearly, one may well define the window in terms of instances' number), and then discarding the segments that do not contain a sufficient number of examples of the attack class. This way, as depicted in the figure, the data chunks $D_1, D_2, \ldots$ are the result of first segmenting the stream of network-connection instances, and then filtering out those that are not useful for training a discriminative model.

Clearly, to ensure that the base classifiers are sufficiently reliable and diverse from one another, the size of the sliding window should be devised in a way that: (i) an adequate number of labeled instances can be extracted from all the chunks, and (ii) the resulting chunks are likely to feature different data distributions.[3]

The discovery of the ensemble follows a supervised learning strategy where the labeled connection instances are used as example data to

[3] For the sake of greater diversity, one could employ statistical change-detection tests to filter out redundant chunks and/or identify the chunks' borders more effectively; however, studying the combination of our learning approach with such an advanced feature is beyond the scope of this work, and it is left for future investigation. Anyway, it is worth noticing that fixed-window chunking approaches have been proven effective in dealing with concept drifts on their own, without using change-detection mechanisms [6,7].

induce the base classifiers and possibly a combiner sub-model for merging the outputs of the base classifiers into a final prediction.

Specifically, as shown in Algorithm 1, the labeled data, gathered in each data chunk D_i, are split in a (random) stratified fashion into two subsets, named respectively D_i^B and D_i^E. Therefore, each resulting dataset will include both positive and negative instances, i.e. attack and normal traffic flows, with the same relative frequency. Data sample D_i^E is a fraction *comb_train_perc* of D_i, which is kept aside for eventually training the combiner (line 11 of Algorithm 1) — notice that, in the case the user wants to adopt a non-trainable combination scheme, *comb_train_perc* should be set to 0. D_i^B, containing the remaining instances, is used instead to train the base classifier BM_i associated with the data chunk D_i.

Algorithm 1: Pseudo-code of the chunk-wise ensemble learning scheme

1 function *BuildEM* (D, k, s,*comb_train_perc*)
Input : A list of k labeled data chunks $D = [D_1, D_2, \ldots, D_k]$ temporally sampled from a datastream $\mathscr{D}$;
A combination strategy s;
The percentage of tuples *comb_train_perc* drawn for learning the combiner sub-net.
Output: Ensemble Model EM
2 $D^E = []$ // *samples drawn from each data chunk* D_i
3 $BM = []$ // *list of base models*
4 **foreach** $D_i \in D$ **do**
5 Randomly split D_i in a stratified fashion into two disjoint subsets D_i^E and D_i^B according to the *comb_train_perc* value
6 $D^E \overset{+}{\leftarrow} D_i^E$ // *append the current data sample to* D^E
7 $BM_i = \text{InduceNN}(D_i^B)$ // *induce a DNN from* D_i^B *(see Section 5.1)*
8 $BM \overset{+}{\leftarrow} BM_i$ // *append the current model to the list of base models*
9 **end**
10 $CM = \text{InitCombiner}(s)$ // *initialize a combiner sub-net (see Section 5.2)*
11 $EM = \text{BuildEnsemble}(D^E, BM, CM)$ // *assembly and finalize the ensemble model, possibly training the combiner sub-net on* D^E
12 **return** EM

In order to really apply such a computation scheme to streaming data, according to a continuous learning perspective, one should decide: (i) in which moments a novel version of the ensemble model must be built; (ii) which of the discovered base models must be included in the ensemble. In this work, we assume that, as soon as a novel base classifier BM_i is built, the ensemble is updated to contain the latest k classifiers $BM_{(i-k+1)}, \ldots, BM_i$ as its base models, and a combiner possibly trained on $D^E_{(i-k+1)} \cup \cdots \cup D_i^E$. The assumption underlying such a choice is that k is large enough to allow the ensemble to keep memory of all relevant attack patterns — i.e., so that older data chunks (for which there is no corresponding base classifier in the ensemble) either represent obsolete attack patterns or concern regions of the instances space that are also covered by the selected data chunks. Should it not be the case, one could think of resorting to other model selection strategies (see [3,37] for a survey). The discovered ensemble can be used to classify unlabeled data instances, possibly in an online fashion, until a novel updated ensemble becomes available.

The above learning scheme was partly simulated in the experimentation described in Section 6, where we evaluated the performances of different ensemble models induced from only $k = 7$ data chunks (resulting from segmenting by using non-overlapping windows with the same temporal span of one day).

It is worth noticing, however, that implementing the above described online stream-processing scheme in full is beyond the scope of this work. The classification framework proposed in this work is meant, indeed, to only offer a methodological basis for assessing the feasibility and validity of our idea of extending a chunk-wise ensemble learning approach with deep learning capabilities. And, in fact, experiments conducted with a prototype implementation of the proposed framework allowed us to give a positive empirical answer to the fundamental research question that originated our work, as discussed in Section 6.

4.2. Design choices: how the framework is meant to deal with issues I1 –I4

Let us now discuss the ad-hoc technical solutions that we propose to embed in the incremental intrusion-oriented classification framework introduced above, in order to empower it with the capability to address the challenging issues mentioned in Section 2.2. Notice that the link between these technical features and the general issues I1, I2 and I3 is also summarized in Tables 1 and 2 which allow for quickly comparing our approach to the existing ones, as far as the ability to cope with these issues is concerned.

How issue I1 (non-stationary data) is addressed. Based on previous research studies [7], we naturally expect that the adoption of the chunk-wise learning scheme described above herein should allow our framework to effectively cope with the skewed non-stationary nature of NIDS data, and prevent the risk of catastrophic forgetting that affects deep-learning models (and also many kinds of shallow one) when trained incrementally. In fact, compared to recent DL extensions for mitigating such a forgetting phenomenon [11], chunk-based ensembles appears to be a more consolidated solution, offering a very good balance along the stability–plasticity spectrum, owing to their ability to both: (i) integrate novel information by simply adding new base classifiers to the ensemble; and (ii) forget obsolete knowledge, by removing/underweighting the corresponding old classifier(s) [7]. This feature of chunk-based ensembles proves particularly useful in IDS environments, where concept drifts often concern only some parts of the knowledge discovered (e.g., linked to emerging modalities of intrusion attack or of normal behavior), while leaving the others parts of this knowledge (e.g., concerning ever-recurring attack/normality modalities) still relevant.

Interestingly, the advantage of using this category of ensemble-based learning systems also has some theoretical grounding, seeing as these systems have been proven to provide more stable results than single-classifier approaches in non-stationary settings [38,39], independently of the base learner adopted.

How issue I2 (example scarcity) is addressed. Dividing network -connection instances into temporal chunks exacerbates the risk of having insufficient training data for inducing deep base classifiers, since only a small fraction of these instances can be reasonably assumed to be equipped with a ground-truth class —despite combining multiple (diverse-enough) weak classifiers into an ensemble being an effective way to improve their individual performance, provided that all of them are better than a random classifier.

Thus, in order to prevent the discovery of low-quality per-chunk classifiers, we devise an ad-hoc DNN architecture for these classifiers that features a synergistic combination of *skip* (a.k.a. shortcut) connections and of dropout capabilities. Skip connections allow the base DNN classifier to play similarly to *Residual Networks* [40], which have been proven really robust to the notorious *degradation problem* (neural networks performing worse when increasing their depth), and capable of ensuring a good trade-off between convergence rapidity and expressivity/accuracy. It is worth noting that these benefits seem to be linked to the fact that a residual network plays as an ensemble of alternative transformation paths (as shown in Fig. 2), where most contributions to the final result (and to the gradient) come from medium-length paths, which are just a tiny portion of the total (combinatorial) number of such paths. Thus, no matter whether the network contains long

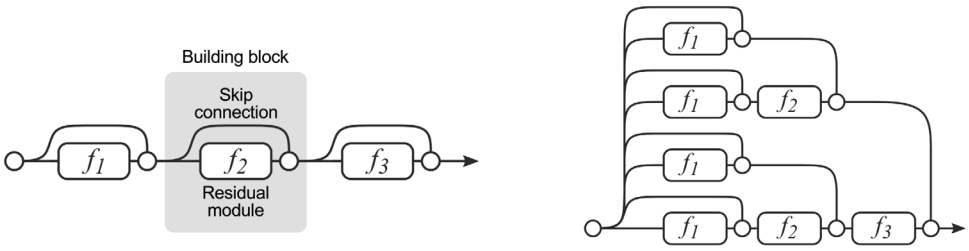


Fig. 2. Ensemble-like interpretation of a residual network (courtesy of [41]): conventional 3-block residual network (left) and its unraveled view (right).

transformation paths (which would be hard to train, owing to vanishing gradients), these "effective" paths can be trained effectively (and quickly enough).

In addition, each base DNN classifier is equipped with dropout layers [42], each of which is meant to mitigate the risk of overfitting by basically forcing the neurons following the dropout layer to depend less on those that precede the layer itself. Interestingly, as noticed in [42], dropout mechanisms make a neural net behave as an ensemble of multiple sub-nets resulting each from randomly masking some of its neurons.

Therefore, in a sense, our approach benefits from different levels of model ensembling: (i) it discovers separately and then combines multiple base classifiers, by training the same DNN architecture over disjoint training samples (resulting from an explicit "horizontal partitioning" of the training instances into data chunks); (ii) each base classifier is a neural network that works as an (implicit) ensemble of sub-nets, owing to the presence of both skip connections and of dropout layers.

A detailed description of the architecture proposed for the base classifiers is provided in Section 5.1.

How issue I3 (class imbalance) is addressed. The use of classic classification-oriented loss functions (such as the popular MSE and binary cross-entropy [43] functions), in the training of our DNN-based base classifiers and combiners, may well lead to models biased towards the majority class of normal connections. In order to prevent this, we adopt a simple (but effective enough, as shown in our experimentation) *cost-sensitive* loss function $L^{(c_{fn},c_{fp})}$, which is defined as follows, parametrically to the costs of false positives (c_{fp}) and of false-negative (c_{fn}):

$$L^{(c_{fn},c_{fp})} = \frac{1}{|T|}\sum_{x\in T} |y_x - \tilde{y}_x| \cdot weight(\tilde{y}_x, y_x, c_{fn}, c_{fp}) \quad (1)$$

where, for any training instance x in some given training/validation set T, y_x is the actual anomaly indicator of x (i.e. a Boolean-like flag in $\{0,1\}$ such that $y_x = 1$ iff x is an instance of an attack class); $\tilde{y}_x \in [0,1]$ is the anomaly score predicted for x; and $weight(\tilde{y}_x, y_x, c_{fn}, c_{fp})$ is an auxiliary error weighting function defined as follows: *(i)* $weight(\tilde{y}_x, y_x, c_{fn}, c_{fp}) = c_{fn}$ iff x is a false negative (i.e. $y_x = 1$ and $\tilde{y}_x \leq 0.5$); *(ii)* $weight(\tilde{y}_x, y_x, c_{fn}, c_{fp}) = c_{fp}$ iff x is a false positive (i.e. $y_x = 0$ and $\tilde{y}_x > 0.5$); and *(iii)* $weight(\tilde{y}_x, y_x, c_{fn}, c_{fp}) = 1$ otherwise.

How issue I4 (combining specialized classifiers) is addressed. To possibly of finding a "data-adaptive" mechanism for combining the base classifiers, we devise three alternative trainable neural-network architectures (described in detail in Section 5.2) for implementing the combiner of the ensemble: two of these architectures compute the final prediction for an instance x by suitably fusing results (namely the final predictions and the topmost hidden representations, respectively) returned for x by the base classifiers, while also taking as input the original features of x (in order possibly to adapt the combination logics to the region of the instance space to which x belongs).

A further "data-adaptive" combiner architecture introduced in our framework, inspired to Mixture-of-Experts models [8], is meant to return, for any data instance x, a weighted average of the attack probabilities predicted for x by the base classifiers, where the classifiers weights are computed through a trainable sub-net that still takes x itself as input.

These three different kinds of trainable combiners can give the ensemble the ability to consider the regions of competence of the base classifiers, differently from usual combination schemes like (weighted) majority voting and prediction averaging.

In addition to these combiners, a simpler non-trainable combiner is considered, which returns the most confident prediction among those of the base classifiers. Even though this method is not fully capable of capturing the reliability regions of the base classifiers, empirically we found it more robust to issue I4 than traditional global combination schemes like (weighted) averaging/voting.

5. Proposed framework: DNN architectures devised for our ensemble

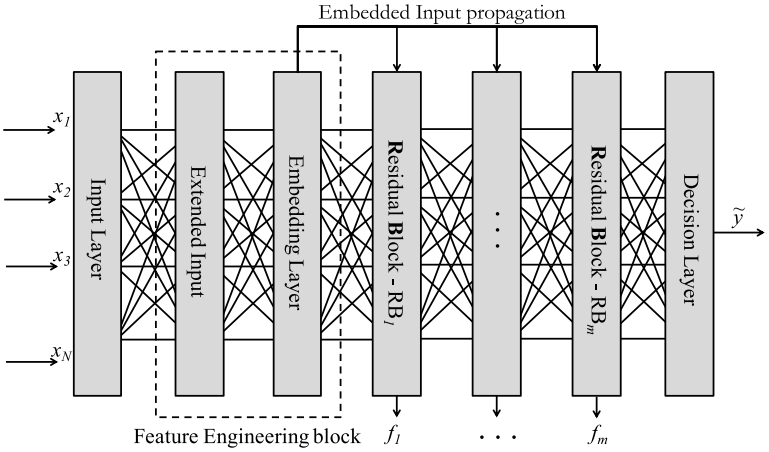
In our approach, an ensemble classifier can be regarded as an overall neural network that integrates two kinds of components: a number of base classifiers sharing the same architecture (but extracted from different data chunks), and a combiner sub-net for deriving an overall prediction from the outputs of the base classifiers. The source code of this framework, used to run the experimental results shown in Section 6, is available at: https://github.com/massimo-guarascio/dnn_ensemble_ids.

The rest of this section illustrates the DNN architectures that we devised for the base models (Section 5.1) and for the combiner sub-net (Section 5.2).

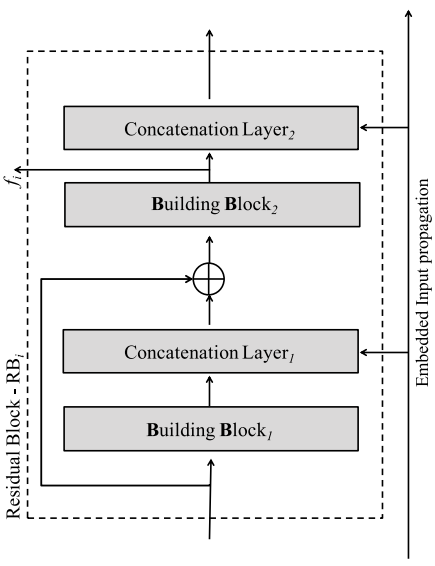
5.1. Base models' architecture (b-DNN)

As mentioned above, in our ensemble learning approach, the base intrusion–detection models share the same (feed-forward) DNN architecture, which is illustrated in pictorial form in Fig. 3(a). The architecture consists of a stack of the following different kinds of sub-nets (appearing in the figure from left to right, respectively):

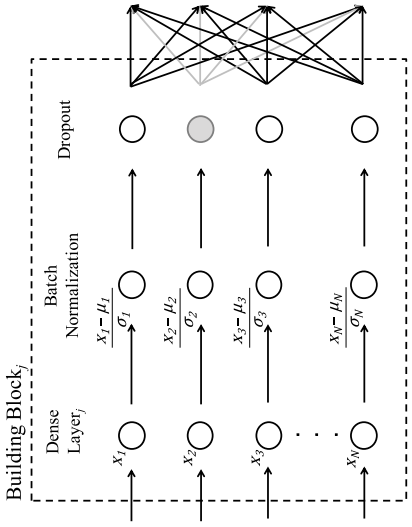
- an *input* layer that is just devoted to representing the log data in a numerical vectoral form;
- a *feature-engineering* block, composed of two layers staked one onto the other: the former layer, named *Extended Input layer*, is in charge of enriching each input vector with a fixed number of additional features, which are computed each by applying a distinguished non-linear functions to each original feature x in the vector; any extended data vector is then made to pass through an *Embedding layer*, which encodes a (trainable) transformation producing a compressed representation of the data vector;
- a variable number $m > 1$ of instantiations of a *Residual Block* sub-net (denoted in the figure as RB_1, …, RB_m), which is meant to produce a "versatile" hierarchy of abstract data features, useful for recognizing the targeted intrusion attacks accurately; essentially, each of these blocks consists of two instantiations of a *Building-block* sub-net, hinging on a fully-connected layer, linked one the other by a skip connection (more details are given later on);



(a) High-level architecture of a base *b-DNN* model.



(b) Details: internal structure (in the dotted rectangle) of the *i*-th *Residual Block*.



(c) Details: internal structure (in the dotted rectangle) of the *Building Block*.

Fig. 3. DNN architecture of all base classifiers.

- a final *Decision Layer*, equipped with a *sigmoid* activation function [43], which maps any given data instance $x = \langle x_1, \ldots, x_N \rangle$ to an anomaly score $\tilde{y}$, representing an estimate of the probability that x is an attack.

More details on the internal structures of the *Building Block* and *Residual Block* sub-nets are illustrated in what follows.

Building block. The architecture of this sub-network, depicted in Fig. 3(c), is composed by three components: *(i)* a Dense (i.e. fully-connected) layer, all the units of which are equipped with a *tanh* activation function [43]; *(ii)* a batch-normalization [44] layer, which is meant to improve the performances and stability of the dense layer in the block; and *(iii)* a dropout [42] layer, which helps strengthen the classifier robustness to overfitting.

Residual block. As illustrated in Fig. 3(b), in each *Residual Block* two replicas of the above-described building block are piled up one onto the other, and linked through a skip/shortcut connection. This connection allows the entire network to behave as a *Residual Network* [40] and reach a good trade-off between convergence rapidity and expressivity, as discussed in Section 4. To further reduce the risk of abstracting too much over the data features and thus losing the fine-grain ones that may possibly help directly in detecting intrusion attacks, the output of either building block's instance (inside each residual block) is concatenated with the output of the Feature Engineering block. The direct use of this low-level data as an additional input for the residual blocks is emphasized in Fig. 3(a) as an additional information flow, labeled as *Embedded Input propagation*.

5.2. Combiner sub-net: four alternative neural-network architectures

Choosing the method for fusing the base models into a collective decision is a key task in the design of a classification ensemble in general. As discussed previously, in our specific setting, this design choice is made even more critical by the fact that the base classifiers are trained on disjoint chunks of the training data that are likely to cover different regions of the data-instance distribution. This led us to discard traditional methods like majority voting and prediction averaging that have been used in the previous literature of IDS-oriented DNN ensembles, seeing as, in our setting, a novel test instance can receive unreliable predictions from the base classifiers that refer to unrelated regions of the instance space.

Before explaining the proposed combination schemes, let us notice that, in our approach, any ensemble classifier is conceived as an overall

neural network that integrates two kinds of components: (i) k base classifiers, denoted as $BM_1, BM_2, \ldots, BM_k$ (sharing all the architecture *b-DNN* shown in Section 5.1) extracted from different data chunks, and (ii) a *combiner* sub-net CM, which is in charge of fusing the different classification skills of the base classifiers.

Four alternative neural-network architectures are proposed here for the combiner CM: a simple non-trainable architecture, named *ensemble_max*, which simply returns the class predicted with the highest probability, and three trainable architectures, named *ensemble_stack*, *ensemble_feature* and *ensemble_moe* that are meant to learn automatically a sort of "context-aware" function that fuses the outputs of the base classifiers according to the characteristics of the data instance x at hand (taken as an explicit direct input of the combiner itself).

Before illustrating these architectures in more detail, let us introduce a small notation. Let FE denote a (trainable) Feature Engineering block, as those defined in Section 5.1, and MLP^q denote a probabilistic-classification block consisting of a one-layer dense (feed-forward) network returning a q-dimensional vector as output, topped by a (softmax, or sigmoid in the case $q = 1$) normalization layer that transforms the components of this vector in a way that they sum up to 1 (so as to represent a sort of discrete probability distribution). Let us also regard all these neural blocks as functions — e.g., for any given data instance x, the expanded representation of x produced by a block FE is indicated as $FE(x)$.

For any given data instance x, let $\tilde{y}^{(i)}$ be the final output (i.e. the anomaly score) returned for x by BM_i, and $\tilde{f}^{(i)}$ be the most abstract vectoral representation of x produced by BM_i (and used as input to the topmost layer of the latter to compute its final prediction), actually denoted as f_m in the schema of the reference *b-DNN* architecture of Fig. 3(a) —to be more precise, $\tilde{f}^{(i)}$ is the output returned by the second building block in the last residual block of BM_i, when providing the latter with x as input.

The behavior of the different combiner architectures considered in our framework can be then defined as follows:

- The *ensemble_max* (non-trainable) combiner architecture simply encodes a function that selects the highest class-membership probability among those estimated, for any test instance x, by the base classifiers — essentially, this combiner returns the most confident prediction as the final decision of the ensemble. More precisely, the prediction returned for x by this architecture is computed as follows: (i) $p^{atk}_{max}(x)$ if $p^{atk}_{max}(x) > p^{norm}_{max}(x)$, or (ii) $1 - p^{norm}_{max}(x)$ otherwise, where $p^{norm}_{max}(x) = \max_{i\in\{i,\ldots,n\}}(1 - \tilde{y}^{(i)})$ and $p^{atk}_{max}(x) = \max_{i\in\{i,\ldots,n\}} \tilde{y}^{(i)}$ represent the highest scores assigned (from all the base classifiers) to the probabilities that x belongs to the classes of normal instances and attacks, respectively.
- The *ensemble_stack* combiner architecture implements a variant of a classical stacked-generalization [45] scheme, which takes as input the "feature-engineered" version of x itself (obtained with a suitable FE block) in addition to the predictions $\tilde{y}_i$ returned for x by the base classifiers. Both kinds of information are given as input to a simple MLP^1 meta-classifier, consisting of just one fully-connected layer with one-dimensional output and *sigmoid* activation function. More formally, the overall prediction returned for x by this architecture is computed as: $MLP^1(FE(x) \oplus \tilde{y}^{(1)} \oplus \tilde{y}^{(2)} \ldots, \oplus \tilde{y}^{(k)})$, where $\oplus$ stands for vector concatenation.
- The *ensemble_feature* combiner architecture is meant to implement a sort of "feature-oriented" stacking, which takes as input both the basic "feature-engineered" version of x and the higher-level representations $\tilde{f}^{(1)}, \ldots, \tilde{f}^{(k)}$ that have been derived for x by the base classifiers — which are hence used only as diverse feature extractors while disregarding their respective final outputs. In other words, the base classifiers are reused as different ways to map a data instance to a high-level feature-vector representation that could be more informative than using the mere predictions $\tilde{y}^{(i)}$ and/or a low-level representation of x. In fact, the idea of merging alternative representations, of the same data, yielded by different neural networks is widely used in practice in many application contexts, which primarily include image/text classification [46]. From a structural point of view, this combiner architecture is identical to the former one, modulo the fact that each prediction $\tilde{y}^{(i)}$ of the ith base classifier is replaced with the feature vector $\tilde{f}^{(i)}$, for $i = 1..k$. Precisely, the overall prediction returned for x by this architecture is computed as: $MLP^1(FE(x) \oplus \tilde{f}^{(1)} \oplus \tilde{f}^{(2)} \ldots, \oplus \tilde{f}^{(k)})$, with $\oplus$ still denoting vector concatenation.
- The *ensemble_moe* combiner architecture is meant to implement a sort of *Mixture-of-Experts* (*MOE*) [8] scheme for fusing the predictions of the base classifiers, so that the final classification of x mainly depends on the predictions of a subset of the base classifiers (hopefully the classifiers that are more competent for the instance-space regions related to x). Specifically, as sketched in Fig. 4, the architecture employs a (trainable) *Gate* sub-net that is devoted to extracting a list $z_1, \ldots, z_k$ of weights for the base classifiers, allowing for computing the final anomaly score $\tilde{y}$ as an optimal convex combination of the predictions $\tilde{y}^{(1)}, \ldots, \tilde{y}^{(k)}$ returned for x by the base classifiers. For the sake of simplicity and of efficiency, the Gate network is made consist of just one fully-connected layer with softmax activations, hence returning a (normalized) k-dimensional vector. Formally, the overall prediction returned for x by this architecture is computed as: $\sum_{i=1}^{k} \tilde{y}_i \cdot z_i$, where $\langle z_1, \ldots, z_n \rangle = MLP^k(FE(x))$.

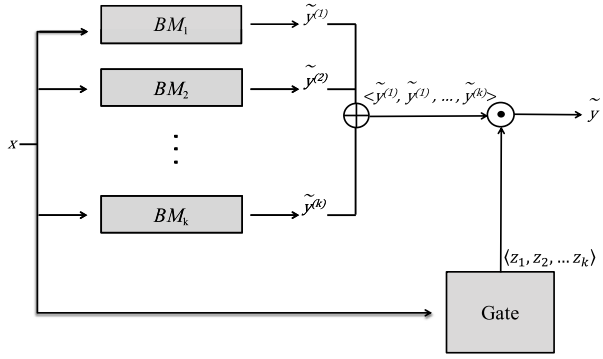


Fig. 4. Mixture-of-Expert (MOE) combiner: conceptual architecture.

As mentioned above, using (an expanded version of) the test instance x itself as an additional input allows the latter three (trainable) architectures possibly to equip the ensemble model with a "data-adaptive" scheme for fusing the partial information produced, for x, by the base models (i.e., abstract feature-vector representations of x in the case of *ensemble_feature*, or class-membership predictions for x in the cases of both *ensemble_stack* and *ensemble_moe*).

6. Experimental investigation

This section illustrates a series of experiments that we conducted to evaluate the performance of our approach on two realistic intrusion detection datasets, widely employed in literature as a benchmark for NIDS approaches. Different suites of tests were carried out for different analysis purposes: (i) to empirically evaluate and compare the alternative model fusion schemes proposed in our framework; (ii) compare our approach with state-of-the-art IDS solutions and well-known ensemble-learning methods; and (iii) analyze the behavior of our ensemble learning approach when a reduced portion of the dataset is labeled. The next subsection supplies more details on the datasets, parameters and competitors used in the experimentation.

6.1. Experimental setup: Datasets and parameters

Datasets. The tests were conducted on two datasets created recently as a better alternative to the classic ones (e.g., KDD, DARPA, NSL-KDD) that were widely used in the past to test IDS systems but shown to suffer from many drawbacks [47,48]: (i) the ISCX IDS dataset [49] from the Information Security Centre of Excellence of the University of New Brunswick, and (ii) the CICIDS2017 dataset,[4] provided by Canadian Institute of Cybersecurity, which contains the most up-to-date common attacks.

In more detail, ISCX IDS is composed of 2,230,620 records, divided into 7 days, containing different types of attack, i.e., HTTP Denial of Service, DDos, Brute Force SSH and attempts of infiltrating the sub-network from the inside (see Table 3). For each day, there are different sub-groups of attack types; therefore, this dataset is particularly suitable to our aims, as it represents the situation in which new types of attacks occur in different periods of time (days).

The CICIDS dataset is divided into five days, from Monday to Friday and includes six attack profiles based on the last updated list (up to 2017) of common attack families. In more detail, on Monday, only normal connections are present, while the other days include different types of attacks [50], i.e., Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet and DDoS (see Table 4).

The presence of different types of attack for each day make these two datasets suitable for testing our approach, by fixing a chunking window of a day. After excluding the days containing no attacks (namely Day 1 and Day 6 for ISCX, and Monday for CICIDS), we treated all the remaining days as distinguished data chunks, as described in Section 4.1.

Notice that the ISCX IDS dataset is quite unbalanced in terms of class distribution: indeed, the fraction of attack instances per day ranges from 0.0 to about 0.067. Despite CICIDS is a little more balanced in the average, in both the datasets, the fraction of attack instances is very small in most of the days — namely, it is less than 0.04 in three of the five days in CICIDS and five out of the seven days in ISCX. Moreover, the distribution of the attack sub-types over the days is very skewed in both datasets: most of the days feature only attack instances of just one or two attack sub-types, along with a far larger number of normal instances. For example, Day 7 of ISCX features only attacks of categories "HTTP DOS" and "Brute Force SSH", representing only 1.435% of that day's data, and there are no other day covering both categories. Similarly, in CICIDS, the attack instances of Thursday just cover 0.483% of the data of that day and belong only to the categories FTP/SSH Patator and Web/Infiltration (which do not occur in any other day). These characteristics make them suitable for evaluating the robustness of our approach to class imbalance in quite a stressing setting, as it is really hard to separate the attack class from the majority class (normal traffic instances).

Test setting. Analogously to what happens in real-life scenarios, we supposed that only a fraction of the instances in the dataset (and in each of its associated data chunks) is labeled. In the experiments of Section 6.5, we made this fraction range from 1% to 33% (in order to study robustness to training data scarcity), while we fixed it to 5% in all the other experiments.

In order to assess the performances of each discovered (ensemble) classification model, we adopted the following hold-out scheme: for each data chunk D_i (corresponding to the ith day of the dataset), a *test set* $Test_i$ was extracted from all the instances of the ith day that were not in D_i (i.e. fictitiously considered as unlabeled, in order to simulate scarcity of training data), by randomly sampling 33% of these instances. All of these day-related test sets were merged together into an overall one, devoted to estimating the (global) performances of a classifier over the entire instance space, and hence its capability to detect all the kinds of attacks present in the dataset. Since in the datasets both the classes (i.e. attacks vs normal connections/flows) and the day-related groups of attack types are rather unbalanced, we believe that it is important also to evaluate the performances of the models on each of these groups (each capturing a different portion of the class of the attacks). Thus, we employed each test set $Test_i$ to measure the ability of a model to classify accurately the specific group of attack types associated with the ith day.

As discussed in Section 4.1, 80% of each chunk (containing a fraction *train_perc* of the respective day's training instances) was used to induce the base models, while leaving the remaining *comb_train_perc* = 20% for training the combiner.

Evaluation metrics. Standard metrics for evaluating an IDS-oriented classifier are the Recall (or True Negative Rate) and Precision, which give an idea of its capability to individuate the attacks and to avoid false alarms, respectively. In more detail, the Recall is the fraction of real attack instances that were reckoned as such (a value of 100% for a classifier means that it detected all the attacks, but it might also yield false alarms), whereas the Precision is the proportion of attack instances detected correctly relative to the number of all the connection instances classified as attacks (a value of 100% means that no false alarms were raised, but there may be some real attacks missed).

These two measures are usually merged into a single one, named (classic) *F-Measure*, computed as their harmonic mean and offering a summarized performance score. For the sake of conciseness and readability, we will next show the F-Measure scores only. Notice, however, F-measure is not really suitable for evaluating classifiers trained on class-imbalanced data. For this reason, we resorted to two further metrics, namely *AUC* and *AUC-PR*, which have been widely employed in the case of imbalanced data (like NIDSs logs), and are described below.

The *AUC* metric quantifies the area under the ROC curve. The ROC curve is computed comparing the False Positive Rate (i.e., the ratio between the number of false alarms signaled and that of all the normal connections processed) and the True Positive Rate (i.e., the Recall).

For imbalanced classes, it was proposed to employ the *AUC-PR* metrics (i.e. the area under the Precision–Recall curve), which does not rely on the True Positive Rate and it is hence less risky in overestimating a model in settings where the number of normal connections is sensibly higher than the attacks. However, as a consequence of this aspect, AUC-PR is more benevolent for approaches raising many false alarms. For this reason, both these two latter metrics have been used in our experimental analysis.

Parameters' setting and competitors. As to the architecture of the base model, described in Section 5.1, the following configuration was adopted in all the tests: (i) the Extended Input layer produces $\sqrt{x}, x^2$, $\log(x+1)$ and $sin(x)$ for every original data feature X; (ii) the Embedding Layer maps its input onto a 96-dimensional vector space; (iii) 3 Residual Blocks are used, for a total of 6 Building Blocks; (iv) in every Building Block, the Dense layer consists of 32 neurons, and the dropout rate is set to 0.01.

This specific instantiation of the proposed DNN architecture was also employed as-is to implement a baseline competitor, denoted as *b-DNN*, allowing us to assess the advantage of training an ensemble of such DNN classifiers. In order to also assess the beneficial role played by the residual-like structures in *b-DNN*, we implemented a further non-ensemble DNN baseline, denoted as *simple-DNN*, which differs from *b-DNN* only for the absence of the skip connections (and of their associated summation components) in all of the 3 Residual Blocks; in other words, *simple-DNN* can be essentially regarded as a stack of 6 Building Blocks (plus the layers for concatenating the output of each of these blocks with that of the Embedding layer).

[4] https://www.unb.ca/cic/datasets/ids-2017.html.

Table 3
Main characteristics of the ISCX IDS dataset.

Day	Description	Size of the pcap file (GB)	Number of flows	Percentage of attacks
Day 1	Normal traffic without malicious activities	16.1	359,673	0.000%
Day 2	Normal traffic with some malicious activities	4.22	134,752	1.545%
Day 3	Infiltrating the network from the inside & Normal traffic	3.95	153,409	6.395%
Day 4	HTTP Denial of Service & Normal traffic	6.85	178,825	1.855%
Day 5	Distributed Denial of Service using an IRC Botnet	23.4	554,659	6.686%
Day 6	Normal traffic without malicious activities	17.6	505,057	0.000%
Day 7	Brute Force SSH + Normal activities	12.3	344,245	1.435%

Table 4
Main characteristics of the CICIDS dataset.

Day	Description	Size of the pcap file (GB)	Number of flows	Percentage of attacks
Monday	Normal traffic without malicious activities	10.0	529,918	0.000%
Tuesday	FTP-Patator, SSH-Patator	10.0	445,909	3.103%
Wednesday	Dos attacks, Heartbleed	12.0	692,703	36.476%
Thursday	Web attacks, Infiltration	7.7	458,968	0.483%
Friday	Bot, DDos, Port Scan	8.2	704,245	41.025%

Our framework and the DNN baselines *b-DNN* and *simple-DNN* were implemented in Python (using the Keras framework on top of Tensorflow), and trained with the RMSprop (Root Mean Square Propagation) optimizer.

For the sake of comparison, we also tested several major ensemble-based classification methods, leveraging the respective implementations available in the *scikit-multiflow* library.[5] These algorithms are described in detail in Section 6.3. If not specified differently, they were run using standard parameters, without performing any kind of parameters' tuning. All the experiments were performed on a Linux cluster with 16 Itanium2 1.4 GHz nodes, each having 2 GB of main memory and connected by a Myrinet high performance network.

6.2. Comparing different DNN ensembles: an analysis of the proposed combiners

Generally, in IDS scenarios, owing to the highly heterogeneous and changing nature of the intrusion attacks, most classifiers may well exhibit different performances in recognizing different attack subclasses. In this respect, let us recall that both datasets used in our experimentation presents different types of attacks on different days, each of which corresponds to a data chunk of our ensemble learning approach.

The objective of this first suite of experiments is to analyze comparatively the behavior of the different combining schemes described in Section 5.2, and to understand eventually which of them are more effective in general/global terms, and each of the sub-classes of attacks represented by the different days. Therefore, we run our approach with each of the proposed combining strategies (namely, *ensemble_moe*, *ensemble_stack*, *ensemble_max* and *ensemble_feature*), and evaluated the quality of the resulting ensemble models against both the entire collection of test instances and the subsets of test instances associated with the days/chunks.

The F-measure, the AUC and the AUC-PR scores obtained, for each day, by the considered ensembles, are shown in Tables 5 and 6, for the ISCX and the CICIDS datasets, respectively.

In addition, the section of the table named **All attacks** refers to the situation in which the discovered ensemble models are evaluated on all the test instances (i.e. the union of $Test_i$, for all days i), and provides an overall evaluation of the classifiers performances across the entire instance space (and all kinds of attacks).

We performed the Friedman test [51] for all the evaluation measures (columns) of Tables 5 and 6 and also for all the tables in the next subsection (Tables 7 and 8). The critical value of the Friedman test is obtained from a chi-square distribution with two degrees of freedom and a significance level of 5%. The Friedman test is a non-parametric statistical test and it is used to detect differences across multiple methods. The null hypothesis of this test is that the median value of all the populations is equal.

However, the Friedman test cannot be used alone to rank different methods, but it can be used only to detect whether the results are significantly different. Once the Friedman test rejects the null hypothesis, a post-hoc test is required in order to find the concrete pairwise comparison, which produces differences. To verify the differences between each couple of algorithms, the Nemenyi post-hoc test [51] is adopted. To better note the differences among the algorithms, when the Friedman test detects differences, we adopt the following strategy; we ranked the algorithms (on the basis of the results of the Nemenyi post-hoc test) by dividing them into groups (the algorithms in the same group are not significantly different) and we highlight in bold the first algorithm (or group of) and in gray the second algorithm (or group of) if it is significantly better than **at least two other approaches**. If the

Table 5
Comparison of the different ensemble strategies: F-measure, AUC and AUC-PR for the ISCX dataset using 5% of the dataset: the **first** and second algorithm (or group of algorithms) that is significantly better than at least three other approaches are reported in bold and gray, respectively.

Dataset	Algorithm	F_measure	AUC	AUC-PR
All attacks	*ensemble_feature*	0.948 ± 0.027	0.999 ± 0.001	0.981 ± 0.010
	ensemble_moe	0.960 ± 0.007	**1.000 ± 0.000**	**0.990 ± 0.005**
	ensemble_stack	0.922 ± 0.051	0.989 ± 0.022	0.953 ± 0.048
	ensemble_max	0.938 ± 0.033	0.999 ± 0.001	0.978 ± 0.011
Day 2	*ensemble_feature*	0.967 ± 0.011	**1.000 ± 0.000**	**0.986 ± 0.016**
	ensemble_moe	0.971 ± 0.010	**1.000 ± 0.000**	**0.995 ± 0.009**
	ensemble_stack	0.960 ± 0.022	0.996 ± 0.006	0.976 ± 0.023
	ensemble_max	0.964 ± 0.012	0.999 ± 0.001	0.981 ± 0.005
Day 3	*ensemble_feature*	0.693 ± 0.033	0.953 ± 0.012	0.614 ± 0.011
	ensemble_moe	**0.720 ± 0.035**	**0.962 ± 0.005**	**0.632 ± 0.003**
	ensemble_stack	0.644 ± 0.120	0.942 ± 0.039	0.591 ± 0.034
	ensemble_max	0.693 ± 0.050	**0.960 ± 0.004**	0.617 ± 0.012
Day 4	*ensemble_feature*	0.552 ± 0.031	0.977 ± 0.010	0.420 ± 0.017
	ensemble_moe	**0.569 ± 0.016**	**0.984 ± 0.002**	0.435 ± 0.019
	ensemble_stack	0.544 ± 0.033	0.944 ± 0.073	0.412 ± 0.064
	ensemble_max	0.533 ± 0.071	0.976 ± 0.008	0.418 ± 0.018
Day 5	*ensemble_feature*	0.915 ± 0.002	0.996 ± 0.002	0.946 ± 0.022
	ensemble_moe	**0.949 ± 0.018**	**0.999 ± 0.000**	**0.987 ± 0.003**
	ensemble_stack	0.918 ± 0.009	0.996 ± 0.002	0.940 ± 0.035
	ensemble_max	0.932 ± 0.020	**0.998 ± 0.002**	0.929 ± 0.053
Day 7	*ensemble_feature*	**0.946 ± 0.031**	0.999 ± 0.001	0.980 ± 0.012
	ensemble_moe	**0.959 ± 0.008**	**1.000 ± 0.000**	**0.989 ± 0.006**
	ensemble_stack	0.917 ± 0.055	0.988 ± 0.025	0.950 ± 0.051
	ensemble_max	0.925 ± 0.033	0.999 ± 0.001	0.973 ± 0.011

[5] https://scikit-multiflow.github.io/scikit-multiflow/index.html.

Table 6
Comparison of the different ensemble strategies: F-measure, AUC and AUC-PR for the CICIDS dataset using 5% of the dataset: the **first** and second algorithm (or group of algorithms) that is significantly better than at least three other approaches are reported in bold and gray, respectively.

Dataset	Algorithm	F_measure	AUC	AUC-PR
All attacks	*ensemble_feature*	0.977 ± 0.075	0.995 ± 0.018	0.994 ± 0.023
	ensemble_moe	0.995 ± 0.001	**1.000 ± 0.000**	**1.000 ± 0.000**
	ensemble_stack	0.988 ± 0.015	0.998 ± 0.002	0.998 ± 0.002
	ensemble_max	0.965 ± 0.006	0.993 ± 0.007	0.993 ± 0.012
Tuesday	*ensemble_feature*	0.943 ± 0.021	0.998 ± 0.001	0.978 ± 0.013
	ensemble_moe	0.948 ± 0.035	0.999 ± 0.001	0.978 ± 0.018
	ensemble_stack	0.940 ± 0.027	0.981 ± 0.049	0.969 ± 0.029
	ensemble_max	0.920 ± 0.086	0.987 ± 0.030	0.951 ± 0.062
Wednesday	*ensemble_feature*	**0.989 ± 0.006**	**0.999 ± 0.000**	**0.999 ± 0.000**
	ensemble_moe	**0.991 ± 0.004**	**0.999 ± 0.000**	**0.999 ± 0.000**
	ensemble_stack	0.984 ± 0.019	0.998 ± 0.005	0.997 ± 0.005
	ensemble_max	0.986 ± 0.010	0.996 ± 0.003	0.995 ± 0.003
Thursday	*ensemble_feature*	0.805 ± 0.094	0.982 ± 0.015	0.783 ± 0.076
	ensemble_moe	0.803 ± 0.110	0.983 ± 0.012	**0.802 ± 0.081**
	ensemble_stack	0.790 ± 0.099	0.975 ± 0.021	0.778 ± 0.090
	ensemble_max	0.772 ± 0.131	0.980 ± 0.009	0.736 ± 0.080
Friday	*ensemble_feature*	0.978 ± 0.077	0.996 ± 0.018	0.995 ± 0.019
	ensemble_moe	0.996 ± 0.001	**1.000 ± 0.000**	**1.000 ± 0.000**
	ensemble_stack	0.989 ± 0.014	0.999 ± 0.001	0.999 ± 0.001
	ensemble_max	0.972 ± 0.056	0.994 ± 0.006	0.993 ± 0.008

Friedman test fails, no algorithms are highlighted, as the differences are not significant.

By analyzing the table concerning the ISCX dataset, it is evident that the MOE strategy obtains the best results for all the measures, and also for all the days/chunks, representing different sub-groups of attacks. In more detail, the differences in terms of AUC are not substantial, while, as for the AUC-PR, the differences are significant in comparison with all the other techniques and also considering each day, with the exception of the day 4, in which the performance of the different algorithm are not significantly different. As for the other combiner schemes, *ensemble_feature* generally performs slightly better than the other algorithms, even if in many cases, there is no significant difference with the remaining techniques. As for the CICIDS dataset (Table 8), the MOE combination strategy performs better than the others on the overall dataset, and for the relevant metric of AUC-PR, it obtains better results for Thursday and Friday, while for the other two days, all the techniques perform equally well.

These experimental findings empirically demonstrate the validity of introducing the ad hoc schemes *ensemble_moe*, *ensemble_stack*, *ensemble_feature*) for fusing the predictions of specialized base classifiers (cf. issue i4 in Section 2.2) –trained, indeed, on data chunks (days) featuring quite different data distributions.

Having said this, it is worth noticing that the performances of the different combining strategies do not differ in a dramatic way. Thus, in application scenarios where there are stringent computation constraints (e.g., owing to very high streaming rates and very short chunking windows), one could even opt for the faster non-trainable *ensemble_max* combiner, while tolerating some degradation of the detection accuracy.

6.3. Comparing with baseline and ensemble-based algorithms

Based on the analysis conducted in the previous subsection, the MOE-based combination strategy appears to be the best performing one (across all the evaluation metrics considered). Therefore, in this subsection, we deeply compare the *ensemble_moe* variant of our approach with other well-known incremental ensemble-based algorithms, with the baseline DNN model used and with a simple DNN model. The choice of focusing on existing incremental learning methods in this analysis reflects the fundamental requirement of dealing with the definitely non-stationary nature of IDS data.

Table 7
Comparison of the MOE-based ensemble with the competitors and the baseline: F-measure, AUC and AUC-PR for the ISCX dataset using 5% of the dataset: the **first** and second algorithm (or group of algorithms) that is significantly better than at least three other approaches are reported in bold and gray, respectively.

Dataset	Algorithm	F_measure	AUC	AUC-PR
All attacks	*ensemble_moe*	0.960 ± 0.007	**1.000 ± 0.000**	**0.990 ± 0.005**
	b-DNN	0.930 ± 0.022	0.997 ± 0.001	0.981 ± 0.010
	simple-DNN	0.929 ± 0.023	0.995 ± 0.001	0.945 ± 0.035
	ozabag knn	0.940 ± 0.020	0.964 ± 0.016	0.941 ± 0.020
	ozabag htree	0.968 ± 0.015	0.974 ± 0.013	0.969 ± 0.014
	learn++.nse	0.924 ± 0.039	0.937 ± 0.028	0.929 ± 0.037
	online boosting	**0.976 ± 0.008**	0.986 ± 0.005	0.976 ± 0.007
Day 2	*ensemble_moe*	0.971 ± 0.010	**1.000 ± 0.000**	**0.995 ± 0.009**
	b-DNN	0.962 ± 0.008	**0.998 ± 0.002**	**0.992 ± 0.005**
	simple-DNN	0.921 ± 0.020	0.992 ± 0.002	0.901 ± 0.061
	ozabag knn	0.844 ± 0.042	0.909 ± 0.047	0.875 ± 0.056
	ozabag htree	0.964 ± 0.023	0.979 ± 0.018	0.965 ± 0.022
	learn++.nse	0.681 ± 0.053	0.760 ± 0.068	0.850 ± 0.062
	online boosting	0.963 ± 0.022	0.981 ± 0.018	0.964 ± 0.021
Day 3	*ensemble_moe*	**0.720 ± 0.035**	**0.962 ± 0.005**	0.632 ± 0.003
	b-DNN	0.654 ± 0.041	0.943 ± 0.015	0.613 ± 0.018
	simple-DNN	0.635 ± 0.085	0.905 ± 0.005	0.606 ± 0.038
	ozabag knn	0.402 ± 0.107	0.658 ± 0.056	0.457 ± 0.091
	ozabag htree	0.648 ± 0.069	0.864 ± 0.052	**0.708 ± 0.044**
	learn++.nse	0.539 ± 0.060	0.718 ± 0.043	0.585 ± 0.041
	online boosting	0.591 ± 0.066	0.766 ± 0.049	0.621 ± 0.048
Day 4	*ensemble_moe*	**0.569 ± 0.016**	**0.984 ± 0.002**	0.435 ± 0.019
	b-DNN	0.526 ± 0.030	0.969 ± 0.016	0.422 ± 0.028
	simple-DNN	0.445 ± 0.102	0.962 ± 0.005	0.404 ± 0.037
	ozabag knn	0.402 ± 0.065	0.685 ± 0.049	0.415 ± 0.061
	ozabag htree	0.367 ± 0.070	0.686 ± 0.077	0.425 ± 0.079
	learn++.nse	0.216 ± 0.088	0.573 ± 0.047	0.334 ± 0.048
	online boosting	0.397 ± 0.085	0.681 ± 0.062	0.418 ± 0.068
Day 5	*ensemble_moe*	0.949 ± 0.018	**0.999 ± 0.000**	**0.987 ± 0.003**
	b-DNN	0.931 ± 0.009	**0.998 ± 0.001**	0.978 ± 0.014
	simple-DNN	0.932 ± 0.016	**0.997 ± 0.001**	0.971 ± 0.013
	ozabag knn	0.970 ± 0.004	0.989 ± 0.004	0.975 ± 0.004
	ozabag htree	0.950 ± 0.013	0.986 ± 0.005	0.951 ± 0.012
	learn++.nse	0.966 ± 0.007	0.972 ± 0.007	0.968 ± 0.006
	online boosting	**0.976 ± 0.004**	0.986 ± 0.004	0.977 ± 0.004
Day 7	*ensemble_moe*	0.959 ± 0.008	**1.000 ± 0.000**	**0.989 ± 0.006**
	b-DNN	0.934 ± 0.024	**0.999 ± 0.001**	0.980 ± 0.011
	simple-DNN	0.920 ± 0.024	**0.998 ± 0.001**	0.951 ± 0.033
	ozabag knn	0.947 ± 0.020	0.969 ± 0.014	0.948 ± 0.020
	ozabag htree	0.968 ± 0.016	0.973 ± 0.014	0.969 ± 0.015
	learn++.nse	0.967 ± 0.030	0.974 ± 0.023	0.968 ± 0.029
	online boosting	**0.978 ± 0.008**	0.987 ± 0.005	0.977 ± 0.007

In more detail, our approach is compared with the two baseline neural-network models described in Section 6.1 and with four recent incremental learning approaches to the discovery of ensemble classifiers (all having a public implementation available in the scikit multi-flow library): two versions of the *OzaBagging* algorithm [52], respectively instantiated with a KNN and an Hoeffding-Tree base learner, the *Learn++.NSE* [6] and the *Online Boosting* algorithm [53].

Basically, the *OzaBagging* method can be considered as an adaptation of the well-known Bagging algorithm [54] to online classification settings that employs a Poisson distribution to decide whether an example will be used or not for training the chain of classifiers.

Online Boosting uses the same strategies as OzaBagging, but adapted to the Boosting algorithm [54].

Finally, *Learn++.NSE* first trains one classifier for each chunk of the data, and then combines these classifiers by using a dynamically weighted majority voting. The weights are automatically computed on the basis of the accuracy of each classifier (adjusted considering the time of the data) on current and past environments.

Tables 7 and 8 report the final result of the comparison among the MOE-based ensemble and the competitors, respectively for the ISCX and the CICIDS dataset. In all the experiments we considered as metrics,

Table 8
Comparison of the MOE-based ensemble with the competitors and the baseline: F-measure, AUC and AUC-PR for the CICIDS dataset using 5% of the dataset: the **first** and second algorithm (or group of algorithms) that is significantly better than at least three other approaches are reported in bold and gray, respectively.

Dataset	Algorithm	F_measure	AUC	AUC-PR
All attacks	*ensemble_moe*	0.995 ± 0.001	**1.000 ± 0.000**	**1.000 ± 0.000**
	b-DNN	0.987 ± 0.012	0.998 ± 0.001	0.997 ± 0.000
	simple-DNN	0.977 ± 0.023	0.996 ± 0.002	0.995 ± 0.002
	ozabag knn	0.992 ± 0.001	0.994 ± 0.001	0.993 ± 0.001
	ozabag htree	0.994 ± 0.001	0.995 ± 0.001	0.996 ± 0.001
	learn++.nse	0.994 ± 0.001	0.996 ± 0.001	0.995 ± 0.001
	online boosting	**0.996 ± 0.001**	0.997 ± 0.000	0.997 ± 0.000
Tuesday	*ensemble_moe*	0.948 ± 0.035	**0.999 ± 0.001**	**0.978 ± 0.018**
	b-DNN	0.873 ± 0.070	0.996 ± 0.002	0.942 ± 0.061
	simple-DNN	0.739 ± 0.173	0.985 ± 0.032	0.837 ± 0.169
	ozabag knn	0.879 ± 0.009	0.993 ± 0.000	0.891 ± 0.007
	ozabag htree	0.827 ± 0.093	0.863 ± 0.073	0.860 ± 0.067
	learn++.nse	0.961 ± 0.014	0.989 ± 0.005	0.962 ± 0.013
	online boosting	**0.981 ± 0.011**	0.989 ± 0.006	**0.982 ± 0.011**
Wednesday	*ensemble_moe*	0.991 ± 0.004	**0.999 ± 0.000**	**0.999 ± 0.000**
	b-DNN	0.979 ± 0.016	0.998 ± 0.003	0.998 ± 0.003
	simple-DNN	0.960 ± 0.029	0.995 ± 0.001	0.995 ± 0.002
	ozabag knn	0.989 ± 0.001	0.992 ± 0.000	0.990 ± 0.001
	ozabag htree	0.985 ± 0.010	0.987 ± 0.010	0.989 ± 0.006
	learn++.nse	**0.993 ± 0.001**	0.995 ± 0.001	0.994 ± 0.001
	online boosting	0.991 ± 0.004	0.992 ± 0.004	0.994 ± 0.002
Thursday	*ensemble_moe*	**0.803 ± 0.085**	**0.983 ± 0.012**	**0.802 ± 0.081**
	b-DNN	0.490 ± 0.305	0.941 ± 0.045	0.498 ± 0.285
	simple-DNN	0.364 ± 0.325	0.927 ± 0.063	0.365 ± 0.205
	ozabag knn	0.297 ± 0.018	0.724 ± 0.042	0.341 ± 0.042
	ozabag htree	0.134 ± 0.278	0.557 ± 0.126	0.179 ± 0.277
	learn++.nse	0.769 ± 0.055	0.930 ± 0.031	**0.781 ± 0.050**
	online boosting	**0.827 ± 0.046**	0.903 ± 0.029	**0.829 ± 0.045**
Friday	*ensemble_moe*	0.996 ± 0.001	**1.000 ± 0.000**	**1.000 ± 0.000**
	b-DNN	0.989 ± 0.013	0.999 ± 0.000	0.998 ± 0.000
	simple-DNN	0.980 ± 0.022	0.997 ± 0.002	0.998 ± 0.002
	ozabag knn	0.993 ± 0.001	0.994 ± 0.001	0.995 ± 0.001
	ozabag htree	0.996 ± 0.000	0.997 ± 0.000	0.998 ± 0.000
	learn++.nse	0.995 ± 0.001	0.995 ± 0.001	0.995 ± 0.001
	online boosting	0.996 ± 0.000	0.997 ± 0.000	0.997 ± 0.000

the F-measure, AUC and AUC-PR by considering a labeled set of 5% of the entire chunk/dataset.

For the ISCX dataset, our approach is significantly better than all the other approaches for the overall dataset both in terms of AUC and AUC-PR. By analyzing these measures on a per-day basis, the *ensemble_moe* performs better than the other strategies (or in the group of the best ones), for all the days, with the exception of the AUC-PR on day 3, in which the better choice is the OzaBagging (Hoeffding Tree version), and on day 4, in which there is no significant difference among the different approaches. As for the F-measure, Online Boosting obtains the highest value in the overall dataset and in 2 out of 5 days, while our approach is the best choice in 2 out of 5 days, but is quite close to the accuracy obtained by Online Boosting (i.e., 0.950 vs 0.960 for the overall dataset).

Observing the results reported in Table 8 for the CICIDS dataset, the trend is confirmed; indeed, the *ensemble_moe* is significantly better than the others (or in the group of the best ones), in the overall dataset and in each day, both in terms of AUC and AUC-PR, while for the F-measure, Online Boosting outperforms the other approaches in the overall dataset and on both Tuesday and Thursday (together with our approach); however the *ensemble_moe* is only slightly worse than the online boosting in all the cases (i.e., 0.995 vs 0.996 for the overall dataset).

Let us finally notice that, the baseline *b-DNN* performs generally better than *simple-DNN*, as expected. This confirms, indeed, the benefit of inserting residual-like components in the proposed base DNN architecture.

Table 9
Comparison of the MOE ensemble-based approach with three recent ensembles of IDS: Accuracy, DR and FAR for the Wednesday partition of the CICIDS dataset.

Algorithm	ACC	DR	FAR
XGBoost-IDS	0.9136	0.9838	0.1200
HELAD	0.9958	0.9958	0.0215
CFS-BA ensemble	0.9989	0.9990	0.0012
ensemble_moe	0.9979	0.9991	0.0028

Interestingly, the results in Tables 7 and 8 provide empirical evidence for the ability of our approach to deal effectively with the unbalanced nature of NIDS data (cf., issue I3 in Section 2.2). In particular, the performances reached on very unbalanced test subsets (e.g., like those related to Day 7 in ISCX IDS and Thursday in CICIDS) are remarkable, and show how the proposed approach outperforms the competitors (and its base DNN model) in classifying difficult subsets of the test data.

6.4. Comparison with ensemble-based IDS

In the previous subsection, we analyzed a number of popular ensemble-based algorithms that all rely on an incremental learning strategy. To better substantiate the significance of our approach, we also analyzed its behavior in comparison with other state-of-the-art ensembles for IDS. Differently from our approach, these works were designed appositely to work with intrusion detection datasets, and most of them include ad-hoc preprocessing modules. Therefore, in a sense, they can be considered orthogonal to our work, since we could integrate their preprocessing modules into our approach in order to improve its performances.

Specifically, as a first competitor we selected the approach proposed in [18] and denoted as *CFS-BA ensemble* hereinafter, which leverages a heuristic feature-selection algorithm, called CFS-BA (for selecting an optimal subset of features based on the correlation between them). An ensemble model is eventually built that combines C4.5, Random Forest (RF) and Forest PA (Penalizing Attributes) through an average-of-probabilities (AOP) rule. As a further competitor, we selected the *HELAD* system [30], which exploits an ad hoc ensemble of (Autoencoder-based and LSTM-based) anomaly detectors, and different modules for feature extraction and reduction. The last competitor considered in our analysis is *XGBoost-IDS* [55], which essentially reuses the popular eXtreme Gradient Boosting algorithm (as an off-the-shelf ensemble-like classifier), combined with ad hoc parameter tuning and preprocessing modules, specifically tailored to NIDS data. Notably, all of these methods were shown to work quite effectively on CICIDS data — see Section 3 for more details on the former two methods (which are more related to our approach, from a conceptual viewpoint).

Following the testing procedure of [18], for these experiments, we used the Wednesday partition of the CICIDS dataset (with 691,406 instances and 44,858 attacks divided into 5 different types) and we adopted the metrics of accuracy (ACC), detection rate (DR) and false alarm rate (FAR). We also performed the same simple preprocessing and data normalization steps as in [18], for the sake of fair comparison; however, we did not exploited the more complex and time consuming CFS-BA feature selection method introduced in that paper.

From Table 9, in which the test results are summarized, it is evident that *ensemble_moe* outperforms both *HELAD* and the *XGBoost-IDS* in terms of accuracy, detection rate and false alarm rate. On the contrary, the *CFS-BA ensemble* performs very well in terms of false alarm rate in comparison with all the approaches. In terms of detection rate and accuracy, the differences with our approach are not substantial. However, the really good behavior of *CFS-BA ensemble* is mainly owing to the preprocessing heuristic algorithm, which could be also used in our approach. Indeed, without using the CFS-BA preprocessing, it manages to obtain a FAR of 0.02, that is comparable with that achieved by our approach.

Table 10
Comparison of the training time of the base DNN models (average) and of the different ensemble-based approaches, for different percentages of the labeled training set (1%, 2%, 5% and 10% of the entire dataset).

Algorithm	1%	2%	5%	10%
DNN	10.95 ± 0.79	14.43 ± 0.66	29.29 ± 2.40	51.01 ± 4.43
ensemble_feature	16.19 ± 1.03	21.38 ± 1.04	45.04 ± 3.45	76.38 ± 10.30
ensemble_moe	14.57 ± 1.02	18.85 ± 1.18	38.10 ± 3.34	65.17 ± 7.72
ensemble_stack	11.97 ± 1.14	16.75 ± 0.63	33.40 ± 2.63	59.75 ± 5.50

6.5. Sensitiveness to the scarcity of training data

This subsection aims to analyze the behavior of our ensemble-based algorithm when a reduced portion of the tuples is labeled. In order to mimic real-world scenarios, we supposed that this percentage varies from 1% to 33%.

Figs. 5 and 6, respectively for the ISCX and the CICIDS dataset, show the two metrics of AUC and AUC-PR for the comparison between the MOE-based ensemble and the baseline method *b-DNN* (corresponding to inducing just one instance of the base DNN model from the entire training set), when the percentage of labeled training set is 1%, 2%, 5%, 10% and 33% of the dataset (chunk). Figs. 5a–5e (Figs. 6a–6d for CICIDS) consider respectively the single chunks/days, while Fig. 5f (6e for CICIDS) considers the situation in which test instances sampled from the whole dataset are given to the discovered models.

Considering the ISCX dataset, for the cases of the single days and even for all the attacks, it is evident that the ensemble obtains a good accuracy even for a small percentage of labeled data (down to 1% of the data) both in terms of AUC and AUC-PR. On the contrary, as expected, the DNN base model, while reaches results comparable (even if slightly worse) for high percentages of data (5%, 10% and 33%), fails to reach an acceptable level of accuracy, especially in terms of AUC-PR, which is a metric particularly significant for the case of unbalanced datasets. The same trend can be observed for the CICIDS dataset, even if the differences between the ensemble and the base DNN are less evident. In particular, for the hard case of the attacks detected on Thursday, the ensemble needs to work with 20% of the training data to reach about 0.8 against about 0.5 for the base DNN (in terms of AUC-PR).

In summary, the performance scores reported in Figs. 5 and 6, let us conclude that our approach copes effectively with the scarcity of labeled data (issue I2, cf. Section 2.2), and that combining multiple DNN classifiers according to the proposed ensemble scheme allows for neatly improving the base learner.

6.6. Efficiency analysis

An ideal IDS should analyze network traffic data and respond very quickly to attacks. However, the speed in answering to these attacks, mainly depends on the pre-processing phase, including also packet capturing and decoding and on the time necessary to detect the attacks. In this paper, we are not interested in the speed of the pre-processing phase, also because there are many efficient solutions exploiting multi-core and distributed architectures [56], which can also be included in our IDS. However, the speed of detection of known attacks and the time necessary to detect new attacks, which can be very long for supervised IDS, is crucial for reducing the damages that may be caused.

In order to assess the detection time both for known attacks (test time) and for new types of attack (training time), in this subsection, we measured these times for different partitioning of the overall CICIDS dataset (2,301,825 of tuples), varying from 1% to 10% of its data instances.

Table 10 shows the average training time of the base DNN models and the training time for the different ensemble-based approaches when the percentage of labeled training data is 1%, 2%, 5% and 10%. It is important to note that the training time reported concerns only the construction of the combiner sub-model. We omit instead the time necessary to build the base models, since these could be trained incrementally on their respective data chunks as soon as they arrive. Moreover, the times shown in the table could be reduced substantially by using a parallelized computation scheme, using a multi-core or distributed architecture. In addition, these results should be interpreted with the understanding that the algorithms are not optimized as for the execution time, neither do they exploit the advantage of GPUs. The overhead of the *ensemble_max* is not reported because it is negligible, as it simply computes the maximum of the different base models, without any additional computation.

Considering a percentage of 1% (2%) of the dataset, which is sufficient to obtain a good degree of accuracy, the overhead due to using an ensemble of classifiers varies from about 12 (17) seconds for the *ensemble_stack* approach to about 16 (21) seconds for the *ensemble_feature* method.

As for the test time, once the classifiers are trained, it is quite fast and does not slow down the process of detecting the attacks. Indeed, this time, computed for 1000 tuples, is 0.155 s for the DNN base model and it varies slightly from 0.549 s for the *ensemble_stack* to a maximum of 0.589 s for the *ensemble_feature*. Obviously for the *ensemble_max*, the overhead in comparison with the base models is negligible.

Anyway, in spite of having a higher computational overhead, the big advantage of using an ensemble-based approach, as verified experimentally in Section 6.5, is that it can obtain a good level of accuracy by using a lower number of labeled tuples in comparison with base models. One could object that a possible solution would be to use a base model with a higher percentage of training set, instead using an ensemble-based approach with a lower number of examples, and obtaining the same detection time. However, in spite of having the same accuracy, collecting a higher number of labeled data can be costly and in some cases not applicable, because no sufficient labeled attacks may be available.

7. Conclusion, discussion and future work

7.1. Summary of the contribution

A framework for coping with the IDS task, based on a chunk-wise ensemble learning scheme, is proposed here. The framework first builds up a number of specialized base DNN classifiers, induced from disjoint segments of a data stream, then fuses the outputs of these classifiers by means of different combiner sub-nets. In more details, four alternative NN architectures are proposed as a combiner: a simple non-trainable architecture, named *ensemble_max*, that simply returns the class predicted with the highest probability and three trainable architectures, named *ensemble_stack*, *ensemble_feature* and *ensemble_moe* that automatically learn the combining function on the basis of the features and the characteristics of the data itself.

Notably, the proposal addresses several challenging issues that tend to affect real-life NIDS scenarios, which primarily include the scarcity and unbalancedness of training data and the local nature of the data chunks, in addition to the non-stationarity of attack behaviors. In particular, the base classifiers in our ensemble model are built by training a DNN architecture featuring both dropout layers and residual-like skip connections (providing ensemble-like capabilities internally to each base classifier and allowing for curbing the risks of overfitting and slow convergence), using a cost-sensitive loss function (allowing for paying adequate attention to the detection of attack instances, representing samples of the minority class).

The empirical results, obtained on two popular benchmark IDS datasets, have confirmed the better accuracy in terms of AUC and AUC-PR of our approach, compared to existing incremental and non-stationary learning solutions, both on the overall datasets and on most of the different subsets of attacks considered. In particular, among the alternative schemes for the combiner sub-net, the MOE-based one

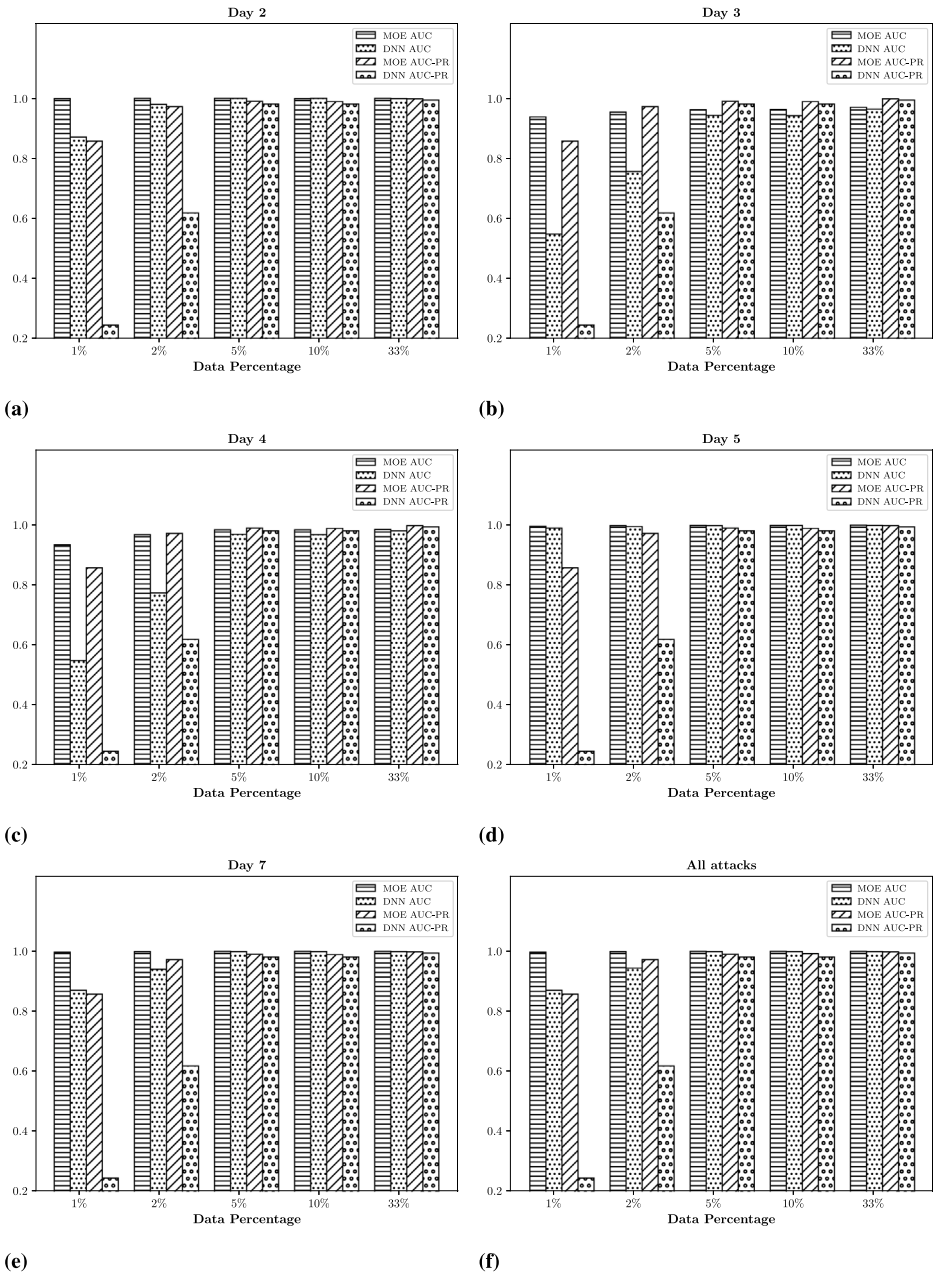


Fig. 5. Comparison of the MOE-based ensemble with the DNN using the AUC and AUC-PR for the ISCX dataset using 1%, 2%, 5%, 10% and 33% of the dataset.

(*ensemble_moe*) outperforms the other schemes. In more detail, MOE performs better than the others both for the ISCX dataset and for the CICIDS dataset, specifically on the entire dataset and for most of the single days, especially for the metric of AUC-PR. However, in a scenario with hard computation constraints (e.g., owing to very high streaming rates and very short chunking windows), the faster non-trainable ensemble *ensemble_max* combiner could be a valid alternative, in spite of accepting some degradation in the detection accuracy.

Despite our experimental analysis have provided some evidence for the ability of our approach to deal with changing and recurring behaviors, we did not evaluated it in an online IDS settings, also due to the lack of benchmark NIDS datasets with long enough temporal coverage. In order to further investigate on this respect, we conducted a few additional tests, simulating an incremental train-and-test usage of our approach over an artificial (three-week ever changing) stream of data extracted from dataset CICIDS. The results of these trials, discussed in brief in Appendix, confirmed the effectiveness of our approach over non stationary data, and its superiority to a purely incremental DNN classifier. A more extensive evaluation of our approach in online classification settings, possibly, with the help of long spanning benchmark NIDS datasets, is out of the scope of this work — to the best of our knowledge, such an evaluation of IDSs is lacking in the literature, likely due to the lack of benchmark datasets that both cover a long enough time range and exhibit different kinds of concept drifts.

Finally, an analysis of the sensitivity to the scarcity of training data establishes that the use of the ensemble, in comparison with the DNN architecture alone (trained as a single DNN classifier), permits a better level of accuracy to be obtained even when a small percentage of labeled data is used. In fact, also using 1% of the training data, the ensemble reaches a good performance, while the DNN base model needs a higher amount of data (from 10% and over) to reach an acceptable level of performance, especially in terms of AUC-PR, which is particularly relevant for unbalanced data.

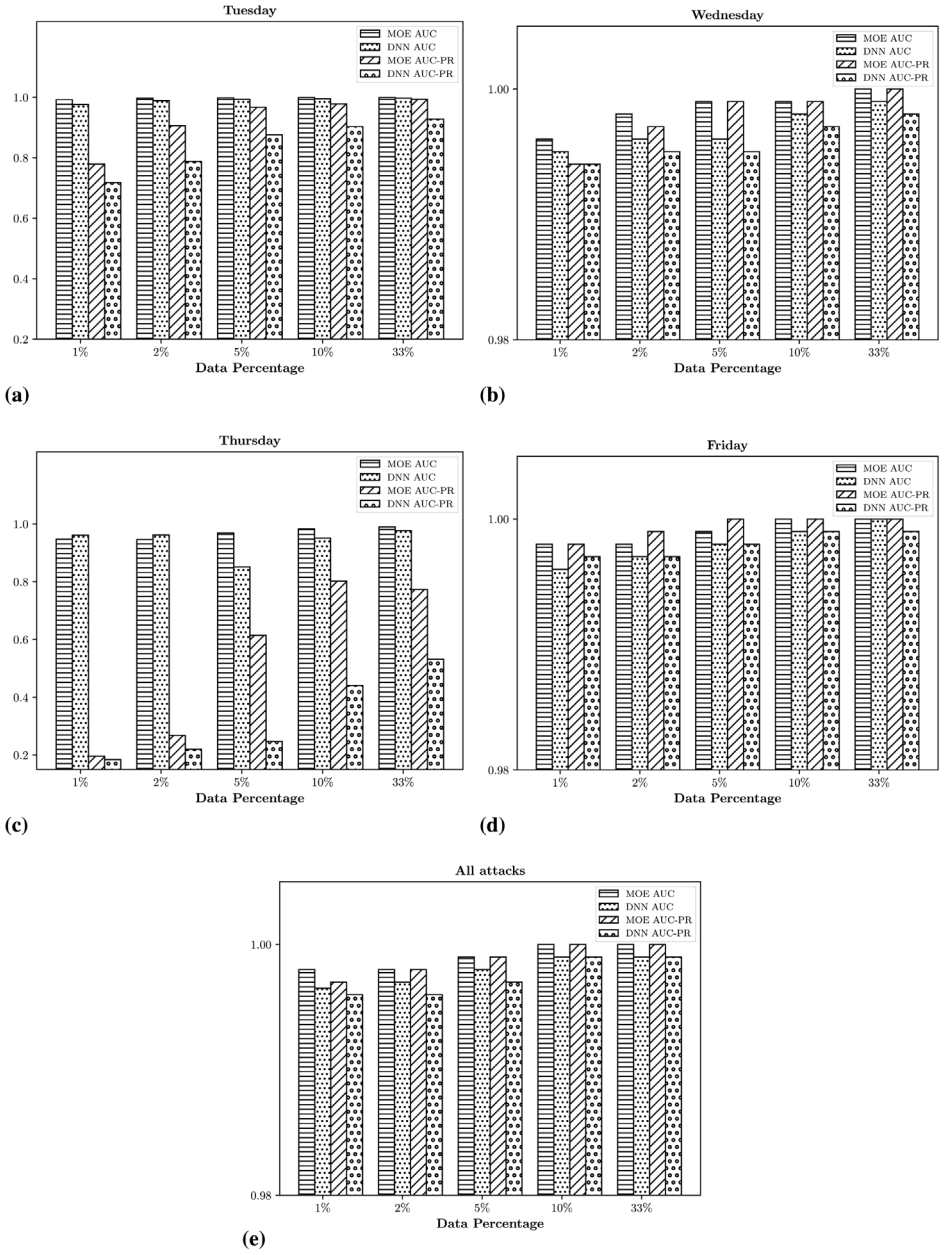


Fig. 6. Comparison of the MOE-based ensemble with the DNN using the AUC and AUC-PR for the CICIDS dataset using 1%, 2%, 5%, 10% and 33% of the dataset.

7.2. Novelty and significance of the contribution

To the best of our knowledge, the combination of deep learning and chunk-based learning techniques is an unexplored line of research in the current literature, despite the former kind of techniques having been proven very powerful in IDS settings, while chunk-wise ensemble learning is widely reckoned as a quite effective and efficient strategy for the analysis of non-stationary data [6,7], such as those that typically occur in IDS logs. As a matter of fact, the specific problem setting considered in this paper (featuring a mixture of challenging issues, concerning the use of non-stationary and small training samples, the presence unbalanced classes and especially the need for fusing specialized base classifiers) looks quite novel in itself with respect to the literature in the field.

The chunk-wise ensemble learning scheme and the peculiar DNN architectures adopted in our approach, make it quite different from all the solutions employing DNN classifiers or ensembles of such classifiers that have been proposed in the field of (N)IDSs (cf. Section 3). In particular, an important distinguishing trait of our proposal with respect to previous DNN ensembles, lies in the use of ad-hoc ensemble combination schemes (including the flexible context-aware *ensemble_moe*, *ensemble_feature*, *ensemble_stack*), in the place of usual weighted averaging/voting mechanisms and their drift-adaptive variants [6,7].

As also summarized in Tables 1 and 2, these additional capabilities of our approach allow it advance the state of the art in the field.

Besides allowing us to assess the feasibility and validity of our idea of hybridizing chunk-wise ensemble learning and deep learning methods, the experimental findings discussed in this paper made us confident in the fact that our work can act as a methodological basis for developing effective, versatile, robust and scalable enough intelligent systems for the analysis of streaming NIDSs logs.

Table A.11
Results obtained by our approach and the baseline in a simulated incremental train-and-test setting. Two scenarios have been considered, which correspond to using either two ($b = 2$) or eight ($b = 12$) chunks to initialize the intrusion detection models tested.

Model	b	F1	AUC	AUC-PR
ensemble_moe	2	0.807	0.966	0.801
b-DNN	2	0.657	0.939	0.803
ensemble_moe	12	0.855	0.996	0.820
b-DNN	12	0.660	0.938	0.793

7.3. Future work

In order to move further along the path of turning our DNN ensemble classification framework into a fully-fledged solution for continuously analyzing real NIDSs log streams, we plan to pursue several directions of research in the future.

Future investigation will concern, in particular, the integration and evaluation of methods for: (i) adaptively setting the data chunking scheme; (ii) detecting (with the help of change/drift detection algorithms), and handling actively, radical changes in global distribution of the log data; (iii) pruning/selecting the base classifiers in a way that the ensemble is made keep a diverse and representative enough collection of models. Solutions developed in the area of ensemble-based stream analysis (including advanced chunk-based approaches to non-stationary learning) [3] offer a solid basis for this research. However, these solutions need to be adapted to peculiarities of our classification setting, especially as concerns the specialized nature of the base classifiers (owing to the restricted coverage of the population of the attacks that is likely to be provided by the data chunks).

Moreover, we will investigate on extending our approach with semi-supervised learning or transfer learning mechanisms, in order to also possibly exploit the (typically vast amounts of) unlabeled data occurring in NIDSs' logs. However, such an extension should be conceived in a careful way, balancing the opportunity of obtaining useful knowledge from such data with the need for keeping the approach computationally cheap enough for an online stream analysis scenario.

We will also investigate defining and evaluating more powerful DNN architectures for the base models, such as sequence-oriented ones (including, e.g., RNN, CNN or Transformer -based sub-nets), while paying attention to the higher risks of overfitting and/or slow-convergence that may arise in such a case. By the way, the choice of using classifiers reasoning a flat representation of network connections (which is indeed quite common in the literature of NIDSs) stemmed from our desire to allow for a fair comparison of our approach with public implementations of state-of-the-art incremental ensemble learning algorithms.

Clearly, our classification-based approach is likely to miss "zero-day" attacks that diverge substantially from the discovered intrusion patterns. This limitation is shared with the very many other supervised/discriminative learning techniques that have been appearing in the field of IDSs, which are usually complemented with anomaly detection tools in practical application scenarios. In this respect, an interesting direction of research would consist in allowing our framework to also include one-class classifiers (possibly based on deep auto-encoder architectures) in the ensemble, while possibly re-designing the logics of the combiner sub-net. Notably, considering the ability of one-class learning methods to perform well in highly-unbalanced two-class classification settings, one could even explore the extreme scenario where the ensemble is made of one-class classifiers only.

CRediT authorship contribution statement

F. Folino: Conceptualization, Methodology, Writing - original draft, Writing - review & editing. **G. Folino:** Supervision, Conceptualization, Methodology, Writing - original draft, Writing - review & editing, Formal analysis. **M. Guarascio:** Conceptualization, Methodology, Data curation, Software, Writing - original draft, Writing - review & editing, Investigation. **F.S. Pisani:** Software, Methodology, Investigation. **L. Pontieri:** Supervision, Conceptualization, Methodology, Writing - original draft, Writing - review & editing, Formal analysis.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work has been partially supported by EU H2020-SU-ICT-03-2018, Grant agreement ID: 830929, within Project CyberSec4Europe and by POR Calabria FESR-FSE 2014-2020, within Project SPIDASEC.

Appendix. Tests on the incremental classification of a simulated data stream

In order to study the proposed framework in a "lifelong" intrusion detection mode, we carried out some additional tests over a simulated stream of network traffic data. In order to make the study as realistic as possible, we started from data instances really stored in the one-week dataset CICIDS, and simulated a scenario where these instances spread over the time forming a three-week data stream.

To this end, after discarding the portion of data gathered for Monday (containing only normal traffic data) to further stress the detection ability of our approach, we randomly split (the data related to) each remaining day into 9 equally-sized subsets, as shown in Fig. A.7.

Let us denote as D_j^i the jth subset sampled from day D_i, for $i = 1, \ldots, 4$ –such that D_1, D_2, D_3 and D_4 refers to the data of Tuesday, Wednesday, Thursday and Friday, respectively.

To simulate both an alternate occurrence of different attack bursts and medium-range recurring behaviors, we tested our approach over the following sequence of subsets: D_1^1, D_1^2, D_1^3, D_2^1, D_2^2, D_2^3, D_3^1, D_3^2, D_3^3, D_4^1, D_4^2, D_4^3 || D_1^4, D_1^5, D_1^6, D_2^4, D_2^5, D_2^6, D_3^4, D_3^5, D_3^6, D_4^4, D_4^5, D_4^6 || D_1^7, D_1^8, D_1^9, D_2^7, D_2^8, D_2^9, D_3^7, D_3^8, D_3^9, D_4^7, D_4^8, D_4^9 —the symbol || separates the data concerning the first, second, and third week of the simulated data stream, respectively.

Hereinafter, let us rename these data sets as $C_1, C_2, \ldots, C_{36}$, respectively, for the sake of readability only.

We simulated an incremental application of our approach over this data stream, using each data set C_i of it as a separate data chunk, while fixing the maximal size of the ensemble to $k = 12$ (i.e. the ensemble is allowed to use only the latest twelve models as shown), in order to put a (quite strict) limit to its memory capacity. In particular, we first initialized the ensemble model with the former $b \leq 12$ chunks $C_1, \ldots, C_8$ (so obtaining a "bootstrap" version of the ensemble, containing only b classifiers), and then used the remaining chunks to incrementally update and test the ensemble. A pictorial representation of the incremental application of our approach over the simulated data stream is shown in Fig. A.8.

More precisely, the ensemble model trained on $C_1, \ldots, C_8$ was tested on C_9, while the updated version of the ensemble obtained after processing C_q (trained incrementally over $C_{q-12+1}, \ldots, C_q$) was tested on C_{1+1}, for $q = b + 1, \ldots, 35$.

In order to have a term of comparison, we also tested a baseline method (denoted as *b-DNN*) training and testing incrementally a single DNN classifier with the same architecture as our base models. The DNN classifier induced this way, from the same data stream as for our approach, is meant to mimic the standard procedure used to adapt a neural-network classifier (like those used by most of the current DL-based approaches to IDS) to work with streaming data.

Table A.11 shows the average results obtained by both our approach and the baseline in two variants of the simulation scenario, which

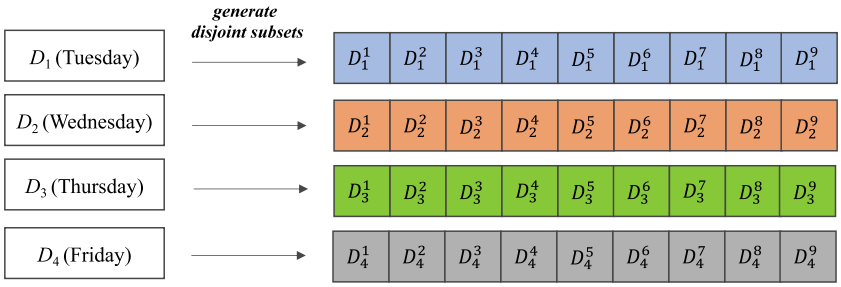


Fig. A.7. Split of days in 9 equally-sized subsets each.

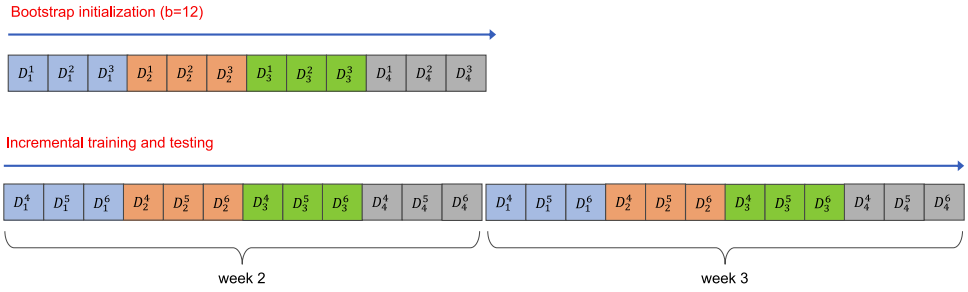


Fig. A.8. Simulation of an incremental application of our approach over the data stream extracted from dataset CICIDS.

differs for the number of chunks employed to initialize the intrusion detection model: $b = 2$ and $b = 12$.

By looking at the table, it is easy to see that the proposed ensemble-based approach outperforms the baseline (especially in terms of both F1 and AUC), in both cases. As expected, the performances of our approach neatly benefit from using more chunks ($b = 12$, rather than $b = 2$) to initialize the ensemble (so allowing it to acquire information on a wider population of attack patterns), while such an improvement is not observed for the purely incremental DNN classifier.

Notice that the worse achievements of our model in the case $b = 2$ strongly depend on the fact that, in the remaining part of the first week of simulation, it was called for detecting attack modalities that it never saw before. Anyway, the behavior of our approach looks pretty satisfactory, even in such a challenging test setting.[6]

In our opinion, these results offers some more evidence for the capability of our approach to deal with highly changing behaviors. A more extensive and deeper experimental evaluation of our approach in an online setting is left to future work.

References

[1] N. Shone, T.N. Ngoc, V.D. Phai, Q. Shi, A deep learning approach to network intrusion detection, IEEE Trans. Emerg. Top. Comput. Intell. 2 (1) (2018) 41–50.

[2] B. Dong, X. Wang, Comparison deep learning method to traditional methods using for network intrusion detection, in: 8th IEEE Intl. Conf. on Communication Software and Networks (ICCSN), 2016, pp. 581–585.

[3] B. Krawczyk, L.L. Minku, J. Gama, J. Stefanowski, M. Woźniak, Ensemble learning for data stream analysis: A survey, Inf. Fusion 37 (2017) 132–156.

[4] M. Woźniak, M. Graña, E. Corchado, A survey of multiple classifier systems as hybrid systems, Inf. Fusion 16 (2014) 3–17.

[5] G. Folino, P. Sabatino, Ensemble based collaborative and distributed intrusion detection systems: A survey, J. Netw. Comput. Appl. 66 (2016) 1–16.

[6] R. Elwell, R. Polikar, Incremental learning of concept drift in nonstationary environments, IEEE Trans. Neural Netw. 22 (10) (2011) 1517–1531.

[7] G. Ditzler, M. Roveri, C. Alippi, R. Polikar, Learning in nonstationary environments: A survey, IEEE Comput. Intell. Mag. 10 (4) (2015) 12–25.

[8] S. Masoudnia, R. Ebrahimpour, Mixture of experts: a literature survey, Artif. Intell. Rev. 42 (2) (2014) 275–293.

[9] F. Iglesias Vázquez, T. Zseby, Analysis of network traffic features for anomaly detection, Mach. Learn. 101 (1–3) (2014) 59–84.

[10] M.H. Bhuyan, D.K. Bhattacharyya, J.K. Kalita, Network anomaly detection: methods, systems and tools, IEEE Commun. Surv. Tutor. 16 (1) (2013) 303–336.

[11] G.I. Parisi, R. Kemker, J.L. Part, C. Kanan, S. Wermter, Continual lifelong learning with neural networks: A review, Neural Netw. 113 (2019) 54–71.

[12] A. Lazarevic, L. Ertöz, V. Kumar, A. Ozgur, J. Srivastava, A comparative study of anomaly detection schemes in network intrusion detection, in: Proceedings of SIAM Intl. Conf. on Data Mining (SDM), 2003, pp. 25–36.

[13] G. Maciá-Fernández, J. Camacho, R. Magán-Carrión, P. García-Teodoro, R. Therón, UGR'16: a new dataset for the evaluation of cyclostationarity-based network IDSs, Comput. Secur. 73 (2018) 411–424.

[14] G. Costa, M. Guarascio, G. Manco, R. Ortale, E. Ritacco, Rule learning with probabilistic smoothing, in: Proceedings of Intl. Conf. on Data Warehousing and Knowledge Discovery (DaWaK), 2009, pp. 428–440.

[15] A. Borji, Combining heterogeneous classifiers for network intrusion detection, in: Annual Asian Computing Science Conference, 2007, pp. 254–260.

[16] S.S. Sivatha Sindhu, S. Geetha, A. Kannan, Decision tree based light weight intrusion detection using a wrapper approach, Expert Syst. Appl. 39 (1) (2012) 129–141.

[17] G. Folino, F.S. Pisani, L. Pontieri, A GP-based ensemble classification framework for time-changing streams of intrusion detection data, Soft Comput. 24 (2020) 17541–17560.

[18] Y. Zhou, G. Cheng, S. Jiang, M. Dai, Building an efficient intrusion detection system based on feature selection and ensemble classifier, Comput. Netw. (2020) 107247.

[19] G. Folino, F.S. Pisani, P. Sabatino, A distributed intrusion detection framework based on evolved specialized ensembles of classifiers, in: Proceedings of Europ. Conf. on the Applications of Evolutionary Computation, 2016, pp. 315–331.

[20] S. Mahdavifar, A.A. Ghorbani, Application of deep learning to cybersecurity: A survey, Neurocomputing 347 (2019) 149–176.

[21] S. Ruder, An overview of gradient descent optimization algorithms, 2016, CoRR, abs/1609.04747.

[22] M.A. Salama, H.F. Eid, R.A. Ramadan, A. Darwish, A.E. Hassanien, Hybrid intelligent intrusion detection scheme, in: Soft Computing in Industrial Applications, Springer, 2011, pp. 293–303.

[23] S. Mohammadi, A. Namadchian, A new deep learning approach for anomaly base IDS using memetic classifier, Int. J. Comput. Commun. Control 12 (5) (2017) 677–688.

[6] It is easy to notice that the performances of our approach in Table A.11 are worse than those discussed in Sections 6.3 and 6.4. This comes from the fact that in the simulation scheme adopted here: (i) our ensemble is trained and tested incrementally, and (ii) the data chunks used here for training the models are far smaller (more than five times) than those used in Sections 6.3 and 6.4 .

[24] K. Alrawashdeh, C. Purdy, Toward an online anomaly intrusion detection system based on deep learning, in: Proceedings of 15th IEEE Intl. Conf. on machine learning and applications (ICMLA), 2016, pp. 195–200.
[25] S. Potluri, C. Diedrich, Accelerated deep neural networks for enhanced intrusion detection system, in: Proceedings of 21st Intl. Conf. on Emerging Technologies and Factory Automation (ETFA), 2016, pp. 1–8.
[26] F. Farahnakian, J. Heikkonen, A deep auto-encoder based approach for intrusion detection system, in: Proceedings of 20th Intl. Conf. on Advanced Communication Technology (ICACT), 2018, pp. 178–183.
[27] A. Javaid, Q. Niyaz, W. Sun, M. Alam, A deep learning approach for network intrusion detection system, in: Proceedings of 9th EAI Intl. Conf. on Bio-inspired Information and Communications Technologies (formerly BIONETICS), 2016, pp. 21–26.
[28] S.A. Ludwig, 2017 IEEE Symposium Series on Computational Intelligence (SSCI), 2017, pp. 1–7,
[29] X. Gao, C. Shan, C. Hu, Z. Niu, Z. Liu, An adaptive ensemble machine learning model for intrusion detection, IEEE Access 7 (2019) 82512–82521.
[30] Y. Zhong, W. Chen, Z. Wang, Y. Chen, K. Wang, Y. Li, X. Yin, X. Shi, J. Yang, K. Li, HELAD: A novel network anomaly detection model based on heterogeneous ensemble learning, Comput. Netw. 169 (2020) 107049.
[31] S. Gamage, J. Samarabandu, Deep learning methods in network intrusion detection: A survey and an objective comparison, J. Netw. Comput. Appl. 169 (2020) 102767.
[32] Y. Bengio, Learning deep architectures for AI, Mach. Learn. 2 (1) (2009) 1–127.
[33] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (1–3) (2006) 489–501.
[34] N. Shone, T.N. Ngoc, V.D. Phai, Q. Shi, A deep learning approach to network intrusion detection, IEEE Trans. Emerg. Top. Comput. Intell. 2 (1) (2018) 41–50.
[35] A. Aldweesh, A. Derhab, A.Z. Emam, Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues, Knowl.-Based Syst. 189 (2020) 105124.
[36] A. Besedin, P. Blanchart, M. Crucianu, M. Ferecatu, Deep online classification using pseudo-generative models, Comput. Vis. Image Underst. 201 (2020) 103048.
[37] H.M. Gomes, J.P. Barddal, F. Enembreck, A. Bifet, A survey on ensemble learning for data stream classification, ACM Comput. Surv. 50 (2) (2017) 1–36.
[38] G. Ditzler, G. Rosen, R. Polikar, Discounted expert weighting for concept drift, in: IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments (CIDUE), 2013, pp. 61–67.
[39] G. Ditzler, G. Rosen, R. Polikar, Domain adaptation bounds for multiple expert systems under concept drift, in: Proceedings of Intl. Joint Conf. on Neural Networks (IJCNN), 2014, pp. 595–601.
[40] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778.
[41] A. Veit, M.J. Wilber, S. Belongie, Residual networks behave like ensembles of relatively shallow networks, in: Advances in Neural Information Processing Systems, 2016, pp. 550–558.
[42] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R.R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors, 2012, arXiv preprint arXiv:1207.0580.
[43] M. Guarascio, G. Manco, E. Ritacco, Deep learning, Encycl. Bioinform. Comput. Biol. ABC Bioinform. 1–3 (2018) 634–647.
[44] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: Proceedings of the 32nd Intl. Conf. on Machine Learning (ICML), 2015, pp. 448–456.
[45] D.H. Wolpert, Stacked generalization, Neural Netw. 5 (2) (1992) 241–259.
[46] C. Kandaswamy, L. Silva, L. Alexandre, J. Santos, J. Sá, Improving deep neural network performance by reusing features trained with transductive transference, in: Proceedings of Intl. Conf. on Artificial Neural Networks (ICANN), 2014, pp. 265–272.
[47] M.V. Mahoney, P.K. Chan, An analysis of the 1999 DARPA/Lincoln laboratory evaluation data for network anomaly detection, in: International Workshop on Recent Advances in Intrusion Detection, 2003, pp. 220–237.
[48] J. McHugh, Testing intrusion detection systems: A critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln laboratory, ACM Trans. Inf. Syst. Secur. 3 (4) (2000).
[49] A. Shiravi, H. Shiravi, M. Tavallaee, A.A. Ghorbani, Toward developing a systematic approach to generate benchmark datasets for intrusion detection, Comput. Secur. 31 (3) (2012) 357–374.
[50] R. Panigrahi, S. Borah, A detailed analysis of CICIDS2017 dataset for designing intrusion detection systems, Int. J. Eng. Technol. 7 (2018) (2017) 479–482.
[51] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (Jan) (2006) 1–30.
[52] N.C. Oza, Online bagging and boosting, in: Proceedings of IEEE Intl. Conf. on Systems, Man and Cybernetics, Vol. 3, 2005, pp. 2340–2345.
[53] B. Wang, J. Pineau, Online bagging and boosting for imbalanced data streams, IEEE Trans. Knowl. Data Eng. 28 (12) (2016) 3353–3366.
[54] Y. Freund, Boosting a weak learning by majority, Inform. and Comput. 121 (2) (1995) 256–285.
[55] A. Bansal, S. Kaur, Extreme gradient boosting based tuning for classification in intrusion detection systems, in: Proceedings of Intl. Conf. on Advances in Computing and Data Sciences, 2018, pp. 372–380.
[56] H. Qadeer, A. Talat, K.N. Qureshi, F. Bashir, N.U. Islam, Towards an efficient intrusion detection system for high speed networks, in: Proceedings of 17th Intl Bhurban Conf. on Applied Sciences and Technology (IBCAST), 2020, pp. 428–433.