Nearest Cluster-based Intrusion Detection through Convolutional Neural Networks

Giuseppina Andresini[a,*], Annalisa Appice[a,b], Donato Malerba[a,b]

[a] *Department of Informatics, Università degli Studi di Bari Aldo Moro, via Orabona, 4 - 70125 Bari - Italy*
[b] *Consorzio Interuniversitario Nazionale per l'Informatica - CINI, Italy*

Abstract

[1] The recent boom in deep learning has revealed that the application of deep neural networks is a valuable way to address network intrusion detection problems. This paper presents a novel deep learning methodology that uses convolutional neural networks (CNNs) to equip a computer network with an effective means to analyse traffic on the network for signs of malicious activity. The basic idea is to represent network flows as 2D images and use this imagery representation of the flows to train a 2D CNN architecture. The novelty consists in deriving an imagery representation of the network flows through performing a combination of the nearest neighbour search and the clustering process. The advantage is that the proposed data mapping method allows us to build imagery data that express potential data patterns arising at neighbouring flows. The proposed methodology leads to better predictive accuracy when compared to competitive intrusion detection architectures on three benchmark datasets.

Keywords: intrusion detection, deep learning, convolutional neural network, clustering, nearest neighbour search

*Fully documented templates are available in the elsarticle package on CTAN.

*Corresponding author (Tel: +39 (0)805443262 Fax: +39(0)805443269)
Email addresses: giuseppina.andresini@uniba.it (Giuseppina Andresini), annalisa.appice@uniba.it (Annalisa Appice), donato.malerba@uniba.it (Donato Malerba)

[1] This version of the contribution has been accepted for publication, after peer review but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: https://doi.org/10.1016/j.knosys.2021.106798. Accepted Version is subject to the publisher's Accepted Manuscript terms of use https://www.elsevier.com/about/policies-and-standards/copyright

1. Introduction

The recent trend in cybersecurity research is recognising deep learning as a definitely relevant approach in computer network security [1, 2], since thousands of zero-day attacks (e.g. new variants of attack) occur due to the addition of various protocols, and most of them are small variants of previously known cyber-attacks [3]. This situation indicates that even advanced mechanisms, such as conventional machine learning systems, try to face the difficulty of detecting these small mutants of attack over time. However, the non-linear activation layers of deep neural networks may actually facilitate the discovery of effective patterns, which keep their effectiveness also under drifting conditions [4].

Following this research direction, various deep learning architectures (see [5] for a survey) have been recently investigated in the intrusion detection literature [6]. However, these methods neglect possible data patterns that may emerge, accounting for network flows within a neighbourhood. They mainly perform learning, accounting for the vector-type representation of the characteristics of single flows. On the other hand, our idea is that the knowledge hidden along the boundary of the neighbour flows, which belong to either the same class or the opposite class of a target flow, may be processed to learn a more accurate intrusion detection model.

So, the primary innovation of this study is the definition of a new deep learning pipeline, that couples the characteristics of a target network flow to the characteristics of the neighbour of the flow under consideration, which belongs to the same class, as well as the characteristics of the neighbour that belongs to the opposite class of the target flow. Enriching the representation of any target flow with neighbourhood information helps deep learning to capture the specific behaviour of the considered feature space inside the input region of the data, belonging to the same class as the imagery target sample as well as along the boundary of the input regions of data associated with the class of the target sample and the opposite class. This allows us to diminish the number of mis-

classifications to discriminate network intrusions from normal flows. Another innovation is that this joint information – the characteristics of the network flows coupled to the characteristics of the neighbour flows – is represented as multiple rows of image-like 2D pixel grids, instead of being concatenated into 1D vectors. By introducing this imagery representation of the network traffic data, we are able to properly process network flows as the input of a 2D Convolutional Neural Network (CNN)[7] architecture.

In particular, the 2D CNN architecture is trained to learn a classification model, in order to discriminate attacking network flows from normal ones, based on the image of the characteristics of the flow under consideration and the characteristics of its neighbours. We note that 2D CNNs are designed to work with grid-structured inputs, which have strong spatial dependencies in local regions of the grid [7]. To produce this spatial dependency condition, we adopt an imagery encoding process of the network traffic data, that is able to reinforce the distribution of similar characteristics (pixels) at close columns. In this way, we are able to take advantage of the ability of 2D CNNs to apply filters on spatially-neighbouring "pixels" of input images, in order to detect spatial patterns (e.g. edges, shading changes, shapes and objects) [8]. These spatial patterns contribute to extracting significant features from the input, thus gaining predictive accuracy.

Specifically, in this paper, we define a new multi-stage intrusion detection methodology that cascades:

- an autoencoder that is used to build a high-level, robust feature vector representation of network data flows;
- a combination of the clustering process and the nearest neighbour search, that is performed in order to represent the network flows as 2D grids of pixels, by accounting for a clustering model of the neighbouring information;
- a 2D CNN architecture that is trained on the 2D imagery representation

of the network flows, in order to build a classification model that discriminates the attacking flow behaviour from the normal one.

We note that autoencoders, nearest neighbour search, clustering and 2D CNNs have already been explored in the literature. However, to the best of our knowledge, the novelty of this study is the specific formulation adopted for these components (in particular, for deriving the imagery representation of the network flows), as well as the effectiveness of the combination of these components in a methodology that actually outperforms the intrusion detection accuracy of several state-of-the-art competitors on various benchmark data sets. In particular, this study contributes to proving that the formulated nearest cluster-based method is an effective means to delineate intrusion-informative patterns, emerging on flows within a neighbourhood. These patterns have a geometric shape, which allows us to build a 2D representation of network flows. So, we are able to train 2D CNNs that achieve classification accuracy gain by speeding-up both learning and predicting operations. In general, our methodology gains in accuracy compared to various intrusion detection models, comprising those using convolutions (but without neighbouring patterns), in order to yield the final classifications.

This paper is organised as follows. The related works are presented in Section 2. The formulated machine learning methodology is described in Section 3, while the implementation details are reported in Section 4. The findings in the evaluation of the proposed strategy are discussed in Section 5. Finally, Section 6 refocuses on the purpose of the research, draws conclusions and proposes future developments.

2. Related works

Although previous research on intrusion detection is mainly populated with conventional machine learning techniques [9, 10, 11], the recent advances made in deep learning have put the problem of intrusion detection at a more challenging level of study and relative computational solutions at an improved level of

performance [12, 13, 14, 15, 16]. Since in this paper we revamp 2D CNN architectures in combination with clustering and nearest-neighbour search, we focus the overview mainly on the cybersecurity literature that applies these methods for intrusion detection.

The CNNs define a type of robust, popular neural network designed to process input data stored in arrays [7]. These neural networks are commonly used for processing 2D arrays of images or audio spectrograms. They are also used frequently for three-dimensional (3D) arrays (videos and volumetric images). Their use in 1D arrays (signals) is less frequent, but is increasing in cybersecurity [17, 18, 17].

The success of 2D CNNs is largely attributed to the use of local filtering and pooling in the network architecture. These operations enable the network to capture deeply the spatial structure of image data. Although 2D CNNs are mainly used in computed vision, the authors of both [19] and [20] originally adopt 2D CNNs in intrusion detection and focus their research on how network flows can be mapped into 2D image arrays, expressing latent characteristics of input data within a 2D data representation. In particular, Li et al. [19] describe a quantization method to convert the value of each numeric feature into an 8-digit binary pixel. The input representation built with this method is finally processed as the input of two popular CNNs, that is, ResNet50 [21] and GoogLeNet [22]. Kim et al. [20] extend the method described in [19] by introducing an RGB-like encode of the data. This input representation is processed in combination with GoogLeNet Inception V3 [22]. The experiments described in [20] prove that their approach outperforms the seminal one in [19]. Millar et al. [23] introduce an image encoding approach, named Flow-Image, which is inspired by the natural language processing theory. They process network flow samples that represent sequences of ten packets. These samples are mapped into a 2D representation by encoding consecutive packets on consecutive rows.

All the above-mentioned studies with 2D CNN architectures are close to the research described here, as they all introduce possible 2D representations of network flows, in order to train 2D CNNs. However, to the best of our knowledge,

none of the existing state-of-the-art algorithms propose a 2D representation of the network flows, which encodes the neighbouring information in the imaging step. On the other hand, this is one of the innovative contributions of this study, which actually aids the proposed intrusion detection system to be more accurate than its competitors.

In addition, we note that, even if the recent research trend in intrusion detection ratifies the prevalence of the use of deep learning, commonly coupled with the supervised paradigm, there are also a few studies experimenting clustering, hence unsupervised learning, as a means to gain accuracy and/or make the learning process scalable. While staying under the umbrella of conventional machine learning, Salo et al. [24] perform clustering for detecting density regions of attacking or normal network flows, and train a conventional classifier from each cluster. A similar idea has been recently investigated in [25] to deal with the imbalanced condition, but in malware classification problems. Focusing on the scalability issue, Peng et al.[26] demonstrate that clustering performed with Mini Batch K-means can remain accurate, even scaling well with large amounts of network flow data.

On the other hand, Benaddi et al. [27] show that clustering can be combined with k-NN to reduce the amount of training data and speed up the nearest neighbour prediction process. This specific combination of clustering and the nearest-neighbour search is also developed in our study, since we perform the nearest-neighbour search on the cluster centroids of both the attacks and the normal samples, respectively. However, the authors of [27] use clustering to reduce the amount of training data and the nearest-neighbour search as the lazy learner of the intrusion detection pattern. On the contrary, we consider the nearest-neighbour search in combination with clustering to augment the training data representation and derive a 2D arrangement of the network flows (jointly with their neighbours). In this way, the intrusion detection pattern can be, in our proposal, deeply learned with a 2D CNN architecture, that is, trained on the neighbourhood-enriched 2D training set.

Finally, there are a few recent studies which cascade clustering and classi-

Table 1: Notation

Symbol	Description
$\mathbf{X}$	original feature space
M	size of $\mathbf{X}$
$\mathbf{X}'$	encoder-level feature space
m	size of $\mathbf{X}'$ with $m \leq M$
Y	target variable with domain $\{attack, normal\}$
$\mathcal{T}$	training collection of network flows
$\mathbf{e}$	network flow $\mathbf{e} = (\mathbf{x}', y)$ with $\mathbf{x}' \in \mathbf{X}'$ and $y \in \mathbf{Y}$
$\mathbf{c}^+$	the nearest cluster centre with $y = normal$
$\mathbf{c}^-$	the nearest cluster centre with $y = attack$
$\mathbf{E}$	2D image associated with e
$\mathbf{x}'$	feature vector $x' \in \mathbf{X}'$
$\mathbf{x}'^+$	feature vector $x' \in \mathbf{X}'$ associated with e^+
$\mathbf{x}'^-$	feature vector $x' \in \mathbf{X}'$ associated with e^-
k	number of cluster centres

fication also in the deep learning scenario. Kenaza et al. [28] perform spectral clustering and consider the distances of a training sample from the cluster centres, to produce the input space of a supervised deep neural network. Yang et al. [29] apply a fuzzy clustering stage to reduce the amount of the training data and the possible imbalance of the samples. Moreover, similarly to [24], they use each cluster to train its own sub-deep belief network classifier. The predictions of all sub-DBNs are aggregated on the basis of fuzzy membership weights. These are calculated for each test sample, according to the nearest neighbour criterion, in the cluster used to train each sub-DBN.

Similarly to the above-mentioned studies, we also adopt clustering to speed up the computation. However, we pursue this speeding-up with respect to the imaging stage, while the related works listed above mainly use clustering to accelerate the deep learning stage, by reducing the volume of data processed to train the networks. We also perform experiments proving that the efficiency in our methodology is gained by preserving the accuracy of the final CNNs trained with the produced images.

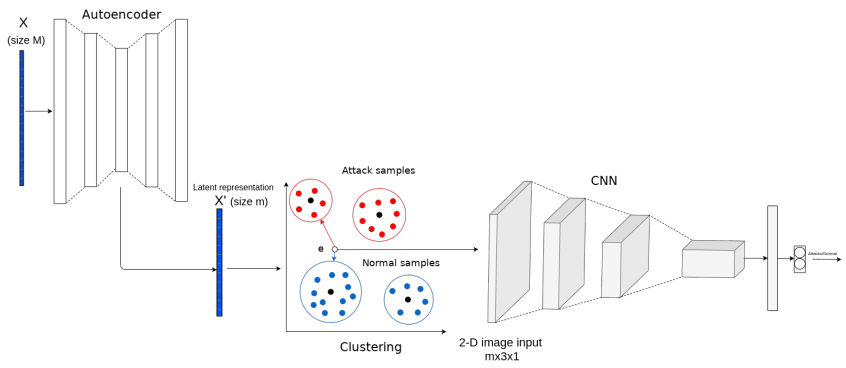
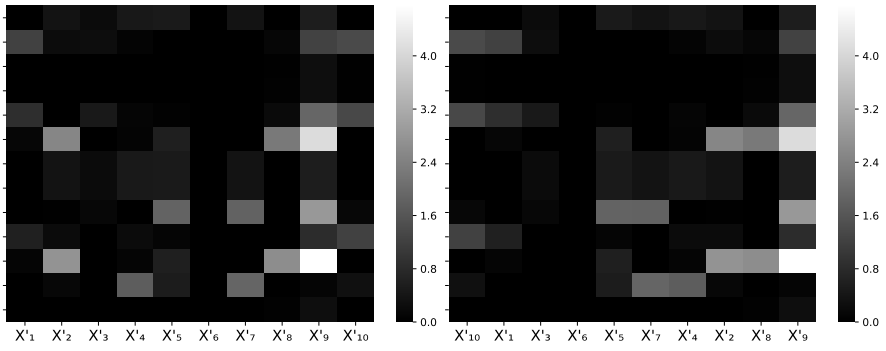


Figure 1: The CLAIRE methodology. It takes as input the training samples spanned on the original feature vector space – $\mathbf{X}$ – with size M, in order to train an autoencoder (left), whose encoder layer is used to extract a compressed, latent, sophisticated feature representation – $\mathbf{X}'$ – of the input, with size m with $m \leq M$ (bottom-center). The encoder-level feature vector representation is mapped into 2D images through nearest neighbouring and clustering. For each training sample ($\mathbf{e}$) the 2D image is built using the nearest normal cluster (in blue) and the nearest attacking cluster (in red). Finally, 2D images are used as input to train a CNN architecture (bottom-right).

3. The intrusion detection methodology of CLAIRE

In this Section we describe the intrusion detection methodology, named CLAIRE (CLuster-based neArest neighbour-based IntRusion dEtection through convolutional neural networks), formulated in this study. The methodology is illustrated in Figure 1, while the adopted notation is introduced in Table 1.

First we train an autoencoder [30] to build a high-level, robust feature representation of the characteristics of the input network flow data. The autoencoder consists of an encoder function $\mathbf{x}' = f(\mathbf{x})$ – mapping the input $\mathbf{x}$ to a hidden code $\mathbf{x}'$ – and a decoder which produces the reconstructed input $\hat{\mathbf{x}} = g(\mathbf{x}')$. We set the output layer of the encoder – $\mathbf{X}'$ – with a lower dimensionality than the input layer $\mathbf{X}$, so that the feature vector $f(\mathbf{x})$ is regarded as a compressed representation of the input $\mathbf{x}$. We consider the compressed representation $\mathbf{X}'$ (in place of $\mathbf{X}$) in the subsequent stages of the methodology.



(a) $\mathbf{X}'$ (b) sorted $\mathbf{X}'$

Figure 2: A portion of the heat map of CICIDS2017Train. In Figure 2a the columns are associated with the features of the output encoder layer $\mathbf{X}'$ of the autoencoder architecture. In the heat map shown in Figure 2a, the columns of Figure 2a have been permuted according to the distance information.

Second we permute the columns of $\mathbf{X}'$ by minimising the distance between the consecutive features of $\mathbf{X}'$. The permutation algorithm is described in Appendix A. The feature permutation step introduces an inter-column continuity in the 1D feature vector by reinforcing the distribution of similar features at close columns. This is highlighted in Figure 2. In particular, Figure 2b shows that the proposed distance-based permutation of the columns, built at the output encoder layer (shown in Figure 2a), contributes to better delineating the data continuity phenomenon across close columns.

Third we map $\mathbf{X}'$ to the 2D input of a CNN. This is done through a combination of the clustering step and the nearest neighbouring search. The clustering step is performed with the K-Means algorithm [31], run on the input normal data and the input attacking data separately. For each clustering execution, samples labelled with the same class are processed as they are spanned on $\mathbf{X}'$, by resorting to the Mini Batch K-means [32] algorithm.[2] In particular, for a

[2]This clustering algorithm is selected according to the analysis performed by [26], who proved that clustering of network flows performed with Mini Batch K-means can remain

given training sample $\mathbf{e} = (\mathbf{x}', y)$, with $\mathbf{x}' \in \mathbf{X}'$ and $y \in \{normal, attack\}$, we determine both the nearest normal cluster centre neighbour – $\mathbf{c}^+$ – and the nearest attacking cluster centre neighbour – $\mathbf{c}^-$ – of this target sample in the input space. The neighbourhood relation is evaluated by computing the Euclidean distance on $\mathbf{X}'$. We account for this neighbourhood information and transform the 1D representation of $\mathbf{e}$ into a 2D representation $(\begin{bmatrix} \mathbf{x}' \\ \mathbf{x}'^{+} \\ \mathbf{x}'^{-} \end{bmatrix}, y)$, where $\mathbf{x}'$, $\mathbf{x}'^{+}$ and $\mathbf{x}'^{-}$ are feature vectors spanned on $\mathbf{X}'$ of $\mathbf{e}$, $\mathbf{c}^+$ and $\mathbf{c}^-$, while y is the class associated with $\mathbf{e}$ in Y. Therefore, by computing this data transformation method, we are able to associate every target sample $\mathbf{e}$ with a $3 \times m \times 1$ image $\mathbf{E}$ spanned on three rows (one row for each sample under consideration) and m columns (one column for each encoder feature of $\mathbf{X}'$), respectively.

Figure 3a shows the 2D array associated with a normal network, while Figure 3b shows the greyscale image associated with this 2D array. We note that, thanks to the permutation step, Figure 3b highlights the phenomenon of continuity on the values of certain features at neighbour samples (e.g. the intra-column values of features X'_2, X'_3, X'_4 and X'_9). On the other hand, there are also columns, which draw attention to a change (discontinuity) that occurs when moving from normal behaviour to attacking behaviour. In Figure 3b, this change range is high on feature X'_1, that assumes similar values (0.46 and 0.48) on both the normal target sample e and its normal nearest neighbour c^+, while it assumes a significantly different value on the attacking nearest neighbour c^- (0.14). The knowledge hidden in the inter-row distribution of some columns is expected to aid the ability of 2D CNN to discriminate between classes.

Finally, we train the 2D CNN architecture, in order to process the 2D data that have been produced as a representation of the input network flows. This neural network is designed according to the theory on CNNs [7] (see Appendix B), in order to discriminate attacks from normal flows.

---

accurate, even scaling well with large amounts of data.

	X'_1	X'_2	X'_3	X'_4	X'_5	X'_6	X'_7	X'_8	X'_9	X'_{10}
$\mathbf{x}'$	0.46	0	0.014	0	0.57	1.91	2.13	0.36	0	0.08
$\mathbf{x}'^{+}$	0.48	0	0.07	0	0.53	1.93	2.14	0.34	0	0.12
$\mathbf{x}'^{-}$	0.14	0	0	0	0.47	1.80	1.35	0	0	0.23

(a) 2D array

(b) 2D image

Figure 3: 2D image representation constructed, with $m = 10$, for a normal network flow sample $\mathbf{e}$, collected in dataset CICIDS2017. The space of the centres of the clusters detected by reducing both the normal input data and the attacking input data are searched to identify neighbours $\mathbf{e}^+$ and $\mathbf{e}^-$. The 2D array of the selected network flow is reported in Figure 3a, while the greyscale image associated with this 2D array is shown in Figure 3b.

In conclusion, we consider the time complexity of the learning stage of the proposed methodology. The time cost of the autoencoder layers is $\mathbf{O}(\sum_{l=1}^{d_A} n_{l-1} n_l)$ [33], where d_A is the number of layers in the autoencoder, l is the index of a layer and n_l is the number of nodes in layer l. The time cost of performing clustering with Mini Batch K-means is $\mathbf{O}(N + btk)$ [34], where N is the number of input network flows, b is the given mini batch size, t is the number of iterations and k is the number of clusters. The time cost of searching the nearest neighbour cluster centroids of the network data flows is $\mathbf{O}(Nk)$. So, the cost of mapping the 1D feature vector representation of the encoded input data into 2D arrays is $\mathbf{O}(N + btk + Nk)$. Finally, the time cost of the convolu-

tion layers[3] is $\mathbf{O}\left(\sum_{l=1}^{d_C} n_{l-1} s_l^2 n_l m_l^2\right)$ [35], where d_C is the number of convolution layers in the 2D CNN, l is the index of a convolution layer, n_l is the number of filters at layer l, s_l^2 is the size of the 2D filters at layer l and m_l^2 is the size of the output feature map at layer l.

3.1. Some inherent remarks

Some inherent remarks concern the motivation behind the decision to use an autoencoder, clustering and an image representation of the network traffic in the proposed method.

The autoencoder introduces the compressed representation $\mathbf{X}'$ of the characteristics of the network flows, that is considered instead of $\mathbf{X}$ in the subsequent stages of the methodology. It allows us to ease the computation of the classification model without affecting its accuracy. The additional advantage is that learning the autoencoder with a non-linear activation function and multiple layers allows us to capture non-linear patterns in network flow data, by building a more robust encoding of their latent data distributions.

The clustering step is introduced since the cost of the nearest neighbouring search is linear in the number of candidate neighbours explored. The clustering step allows us to reduce the neighbour candidate volume. In addition, the cluster centres learn a generalisation of the training samples. So, processing the cluster centres (instead of the original samples) should limit the possible phenomenon of overfitting that may occur during the nearest neighbour search.[4]

[3] As shown in [35], the time cost of fully connected layers and pooling layers can be ignored in the cost analysis. These layers often take 5-10% of the computational time. Therefore, as in [35], we may only consider the trade-off among the convolutional layers. Regarding the pooling operations, the authors of [35] also show that the addition of pooling layers does not change the complexity of subsequent convolution layers in a 2D CNN architecture. So, based upon these considerations, we consider the presence of pooling operations which are irrelevant for the complexity of the subsequent convolution layers.

[4] Based upon these considerations, we expect that clustering speeds up the nearest neighbour search and aids the accuracy of a classification model, learned by accounting for nearest

Final remarks concern the actual advantages of inputting the described 2D representation of the network flow samples to the CNN architecture. One advantage of a 2D CNN architecture is capturing possible spatial contiguity in 2D data [36]. The produced image representation of the input data allows us to depict potential data patterns arising at neighbour samples. In particular, the neighbourhood information of any target sample highlights the specific behaviour of the considered feature space. This space is located inside the input region of the data, belonging to the same class as the imagery target sample, as well as along the boundary of the input regions of data associated with the target sample class and the opposite class.

4. Implementation details

CLAIRE has been implemented in Python 2.7. The source code is available online.[5] The deep neural network architectures are developed in Keras 2.3[6] – a high-level neural network API with TensorFlow[7] as the back-end.

For each dataset we conduct an automatic hyper-parameter optimization, using the tree-structured Parzen estimator algorithm, as implemented in the Hyperopt library [37]. This hyper-parameter optimization is done by using 20% of the entire training as a validation set, according to the Pareto Principle [38]. In particular, we randomly select the validation set with the stratified sampling procedure [39]. So, in CLAIRE, we automatically choose the configuration of the parameters that achieves the best validation loss. The values of the hyper-parameters, automatically explored with the tree-structured Parzen estimator, are reported in Table 2. The effectiveness of the configuration of the hyper-parameters, that is automatically selected by exploring twenty trials generated on the range of values reported in Table 2, is evaluated in Section 5.4.

neighbour information. The experimental study described in Section 5.3 provides the empirical evidence of the soundness of these considerations.

[5] `https://github.com/gsndr/CLAIRE`

[6] `https://keras.io/`

[7] `https://www.tensorflow.org/`

Table 2: Hyper-parameter search space for both the autoencoders and the 2D CNNs.

	autoencoders	classifier
batch size	$\{2^5, 2^6, 2^7, 2^8, 2^9\}$	$\{2^5, 2^6, 2^7, 2^8, 2^9\}$
learning rate	[0.0001, 0.01]	[0.0001, 0.01]
dropout	[0,1]	[0,1]

Table 3: Structure and configuration of the Autoencoder. FC denotes a fully-connected layer. The mean squared error (*mse*) is used as the loss function. The rectified linear unit (*ReLu*) is selected as the activation function for each hidden layer. The *Linear* activation function is used in the last layer.

Layer	Type	#Neurons	Activation function
1	FC	80	*ReLu*
2	FC	30	*ReLu*
3	FC	10	*ReLu*
4	Dropout	10	
5	FC	30	*ReLu*
6	FC	80	*Linear*

The autoencoder architecture comprises five fully-connected (FC) layers and one dropout layer, in order to prevent overfitting. The details of the configuration of this architecture are reported in Table 3. The 2D CNN architecture consists of three convolutional layers, two dropout layers and three fully-connected (FC) layers (see Figure 4). The network takes a training set of 2D samples as input and predicts a Bernoulli probability. The details of the configuration of the CNN architecture are reported in Table 4. Both the architectures select the rectified linear unit (*ReLu*) [40] as the activation function for each layer. This decision is motivated by the recent literature [16], which has proved that the best results are commonly achieved by using this activation function.

We note that the 2D CNN architecture described does not comprise pooling operations. This decision follows the conclusions drawn in [41], which have highlighted that including explicit pooling operations does not always improve the performance of CNNs. We empirically investigate the effectiveness of these conclusions in the intrusion detection scenario addressed in this study. In particular,

Table 4: Structure and configuration of the 2D CNN. The binary-cross entropy is used as the loss function. The rectified linear unit ($ReLu$) is selected as the activation function for each layer.

Layer	Type	#Neurons	#Filters	filter size	stride	Activation function
1	Conv2D		32	2×2	1	$ReLu$
2	Dropout					
3	Conv2D		64	2×2	1	$ReLu$
4	Dropout					
5	Conv2D		128	1×2	1	$ReLu$
6	Flatten					
7	FC	256				$ReLu$
8	FC	1024				$ReLu$
9	FC	2				$Softmax$

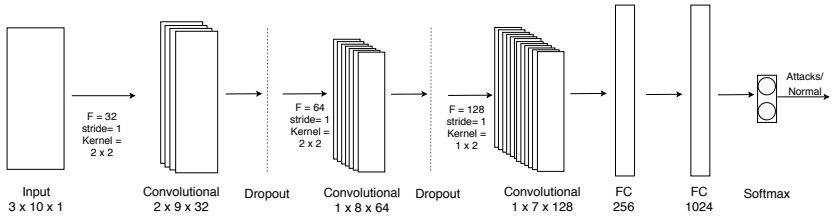


Figure 4: 2D CNN architecture of CLAIRE.

the experimental study described in Section 5.3 shows that the performance of CLAIRE does not actually benefit from the addition of pooling layers.

The networks are trained with mini-batches by back-propagation, and the gradient-based optimisation is performed using the Adam update rule [42]. The weights are initialized following the Xavier scheme. Furthermore, a maximum number of epochs equal to 150 has been set, retaining the best models and using an early stopping approach that achieves the lowest loss on the validation set (the same set used for the hyper-parameter optimization).

The Mini-Batch K-means[8] algorithm is implemented in the Scikit-learn library.[9] This implementation is run with the default parameter configuration (of mini-batch size and number of iterations), except for the number of clusters – k. The sensitivity of the intrusion detection methodology to the set-up of k is investigated in Section 5.3.

5. Empirical evaluation

We consider three benchmark datasets (see Section 5.1), in order to evaluate the effectiveness of the intrusion detection methodology implemented by CLAIRE. Each dataset includes both a labelled training set – processed to learn the intrusion detection model – and a testing set – considered to evaluate the intrusion detection ability of the trained model. In particular, the performance of CLAIRE is measured in terms of accuracy and efficiency (see Section 5.2).

The presentation of the results is organised as follows. First we evaluate the sensitivity of the performance of CLAIRE to the size of the clustering step (i.e. to the number of clusters) performed during the nearest neighbour search of the imaging stage, as well as to the use of pooling operations in the 2D CNN architecture. Then we study the effectiveness of the hyper-parameter optimization that CLAIRE performs during the training of its deep neural networks (see Section 5.4). Subsequently, we discuss the accuracy results that are reported in the recent intrusion detection literature and which are achieved by processing the datasets also considered in our study (see Section 5.5). Finally, we perform an ablation study (see Section 5.6), where we investigate how both the cluster-based nearest neighbour information and the 2D convolutions can jointly contribute to gain accuracy in the intrusion detection model learned by CLAIRE. This analysis also verifies the robustness of this contribution to the imbalance condition.

[8] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html

[9] https://scikit-learn.org/stable/

Table 5: Dataset description. For each dataset we collect: the number of attributes, the total number of network flow samples collected in the dataset, the number of normal network flows (and their percentage of the total size), as well as the number of attacking flows (and their percentage of the total size).

		Dataset		
		KDDCUP99	**UNSW-NB15**	**CICIDS2017**
Attributes	Total	**42**	**43**	**79**
	Binary	6	2	18
	Categorical	3	3	-
	Numerical	32	37	60
	Class	1	1	1
Training set	Total	**494021**	**82332**	**100000**
	Normal flows	97278 (19.7%)	37000 (44.9%)	80000 (80%)
	Attacking flows	396743 (80.3%)	45332 (55.1%)	20000 (20%)
Testing set	Total	**311029**	**175341**	**900000**
	Normal flows	60593 (19.5%)	56000 (31.9%)	720000 (80%)
	Attacking flows	250436 (80.5%)	119341 (68.1%)	180000 (20%)

5.1. Dataset description and experimental methodology

We consider three benchmark intrusion detection datasets, that is, KDDCUP99,[10] UNSW-NB15[11] and CICIDS2017.[12] KDDCUP99 is a benchmark dataset that is commonly used for the evaluation of intrusion detection systems also in recent studies [43, 44, 45]. In this study, we consider 10%KDDCUP99Train for the learning stage, while we use the entire testing set, denoted as KDDCUP99Test, for the evaluation stage.[13] This experimental scenario, with both 10%KDDCUP99Train and KDDCUP99Test, is commonly used in

[10] `http://kdd.ics.uci.edu//databases//kddcup99//kddcup99.html`

[11] `https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/`

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

[13] 10%KDDCUP99Train and KDDCUP99Test are populated with the data stored in kddcup.data_10_percent.gz and corrected.gz at `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`

the literature (e.g. [14, 46, 47]). UNSW-NB15 has recently been used in the evaluation of various intrusion detection methodologies [20, 48, 29, 49]. Finally, CICIDS2017 is commonly used in the evaluation of anomaly detection methodologies with the training performed on the first day [50, 51]. However, a few recent studies consider these data also in the evaluation of classification methodologies, as we do in this paper [20, 52, 46, 53]. In our experimental study we consider the training and testing sets of CICIDS2017, built according to the strategy described in [20]. Specifically, we build one training set with 100K samples and one testing set with 900K samples. Both training and testing samples are randomly selected from the entire 5-day log. For the creation of both the training and testing set, we used stratified random sampling. A summary of the characteristics of the datasets considered in this experimental investigation is presented in Table 5.

5.2. Evaluation metrics

The overall accuracy performance of the proposed methodology is measured by analysing the F1-score of the intrusion detection models learned. This is the harmonic mean of Precision and Recall, where Precision measures the ability of an intrusion detection system to identify only the attacks, while Recall can be thought of as the system's ability to find all the attacks. The higher the F1-score, the better the balance between precision and recall achieved by the algorithm. On the contrary, the F1-score is not so high when one measure is improved at the expense of the other. In addition, we consider Accuracy (that is measured in the evaluation of various competitors). This is the ratio of flows correctly labelled on all flows tested. The mathematical formulation of these accuracy metrics is reported in Table 6.

The efficiency performance is evaluated with the computation time spent training the intrusion detection model and the average time spent processing every testing sample. They are collected on a Linux machine with an Intel(R) Core(TM) i7-9700F CPU @ 3.00GHz and 32GB RAM. All the experiments have been executed on a single GeForce RTX 2080. The training TIME is measured

Table 6: Evaluation metrics: Accuracy, Precision, Recall and F1-score. These metrics are computed by accounting for the number of true positive—TP (number of attacks correctly detected), the number of true negative—TN (number of normal samples correctly detected), the number of false positive—FP (number of normal samples incorrectly detected as attacks) and the number of false negative—FN (number of normal samples incorrectly detected as attacks).

Metric	Mathematical formulation
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
F1-score	$2 \cdot \frac{P \cdot R}{P+R}$

Table 7: Set-up of k considered to run CLAIRE. For each dataset k ranges between $k = 500, 1000, 5000, 10000, 15000, \ldots, maxK$, where $maxK$ is selected as the minimum between the number of normal flows and the number of attacks in the training set.

dataset	$maxK$	k
KDDCUP99	95000	$500, 1000, 5000, 10000, 15000, \ldots, 90000, 95000$
UNSW-NB15	37000	$500, 1000, 5000, 10000, 15000, \ldots, 10000, 15000$
CICIDS2017	15000	$500, 1000, 5000, 10000, 15000$

in minutes, while the testing TIME is measured in milliseconds.

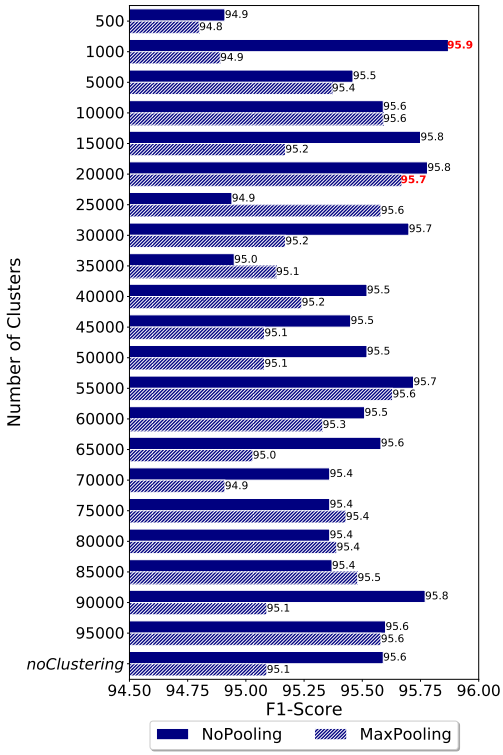
5.3. Sensitivity analysis

This sensitivity analysis is performed, in order to: (1) quantify the gain in both the accuracy and efficiency, due to the introduction of a clustering step during the nearest neighbour search, (2) analyze the need for pooling operations in the adopted 2D CNN architecture and (3) verify the dependence between the sensitivity of the intrusion detection performance and the size of the clustering step (i.e. the number of clusters k), in order to verify the existence of a default configuration that can be considered in all the datasets.

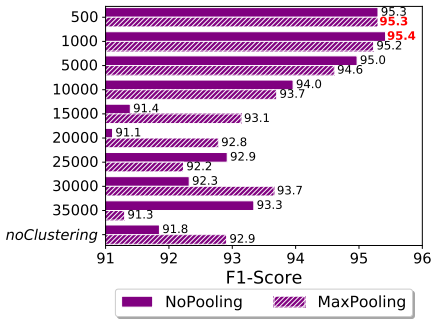
To this aim, we explore the performance of CLAIRE along the number of clusters k when the classification model is trained with two 2D CNN architectures. These architectures are denoted: (1) NoPooling, that is implemented in CLAIRE (see details in Section 4) without the pooling layers and (2) MaxPooling, that is implemented with the addition of the pooling layers. In the experiments performed, we use Max Pooling as a pooling operation. This decision is motivated by the considerations reported in [54], which assess that Max Pooling commonly preserves the most informative features, by offering better translation invariance than Average Pooling. To define the architecture MaxPooling, we modify the NoPooling architecture shown in Figure 4, by introducing two Max Pooling layers before the dropout layers. This is inspired by *LeNet-5* [36], that alternates convolutional layers and pooling layers. The Max Pooling layers are set with a stride equal to 1 and a 2×2 sliding window.

Table 7 describes the parameter set-up of the number of clusters k used in the experiments performed with KDDCUP99, UNSW-NB15 and CICIDS2017. As a baseline of this investigation, we also consider the configuration of CLAIRE with no clustering step performed during the imaging stage. In this baseline, denoted as noClustering, the nearest neighbour search is done by considering all training samples as candidate neighbours. For the analysis of the accuracy performance, we report the F1-score (see Figure 5), while for the analysis of the efficiency, we report the computation TIME spent completing the training phase (see figure 6), as well as the computation TIME spent completing the testing phase (see Figure 7).

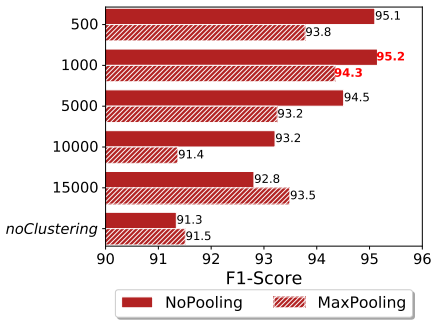
We start analysing the accuracy performance of both MaxPooling and NoPooling by varying k. The results of F1-score, reported in Figure 5, provide the empirical evidence that the clustering step does not cause any decrease in the accuracy performance; quite the opposite, it allows us to achieve a gain in the accuracy of the intrusion detection models learned in all the datasets. Although the gain in F1-score is negligible with KDDCUP99 (from 95.6 to 95.9 with the best clustering configuration of NoPooling and from 95.1 to 95.7 with the best clustering configuration of MaxPooling), it is high with both UNSW-NB15 (from



(a) KDDCUP99 - Testing F1-score



(b) UNSW-NB15 - Testing F1-score



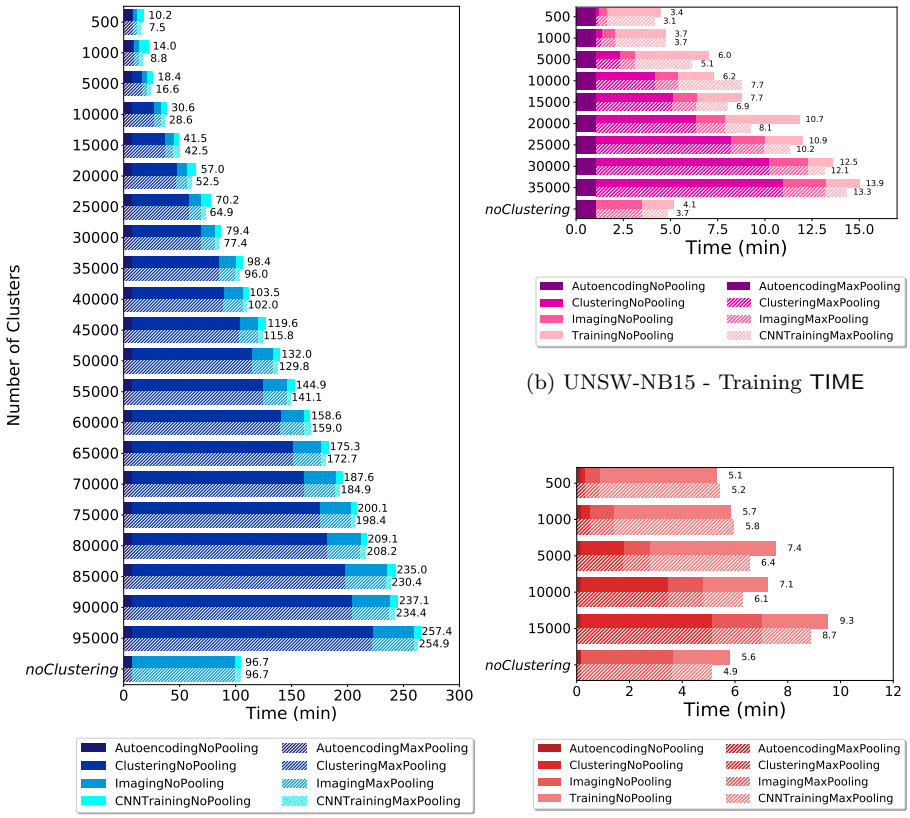
(c) CICIDS2017 - Testing F1-score

Figure 5: Sensitivity analysis: F1-score (axis X) computed on the testing samples, predicted by using the intrusion detection models learned with both the version of CLAIRE with pooling (MaxPooling) and the default version without pooling (NoPooling). The results are collected on the datasets KDDCUP99 (Figure 5a), UNSW-NB15 (Figure 5b) and CICIDS2017 (Figure 5c), by varying the number of clusters in the clustering step (axis Y). For every dataset the highest F-Score is in red for the configurations of CLAIRE with both MaxPooling and NoPooling.

91.8 to 95.4 with the best clustering configuration of NoPooling and from 92.9 to 95.3 with the best clustering configuration of MaxPooling) and CICID2017 (from 91.3 to 95.2 with the best clustering configuration of NoPooling and from 91.5 to 94.3 with the best clustering configuration of MaxPooling). This result confirms our hypothesis that CLAIRE actually benefits from the clustering step. Processing the cluster centres during the nearest neighbour search allows us to limit the occurrence of the phenomenon of overfitting in the nearest-neighbour search, which may compromise the accuracy of the classification model finally learned with this information.

The conclusions drawn above, on the effectiveness in terms of accuracy of the clustering step, are independent of the presence of the pooling operation. Additional considerations can be formulated by focusing attention on the sensitivity of the F1-score along the pooling operation. The higher accuracy achieved with NoPooling is always close to the higher accuracy achieved with MaxPooling (95.9 with NoPooling vs 95.7 with MaxPooling in KDDCup99, 95.4 with NoPooling vs 95.3 with MaxPooling in UNSW-NB15 and 95.2 with NoPooling vs 94.3 with MaxPooling in CICIDS2017). On the other hand, the higher accuracy with NoPooling can be observed at $k = 1000$ in all the datasets. These results show that no significant improvement can be achieved with pooling operations by agreeing upon the same default configuration of CLAIRE-NoPooling for all the datasets. This default configuration is that comprising the clustering step with $k = 1000$ coupled with the 2D CNN architecture NoPooling.

We complete this sensitivity study by evaluating the considered configurations along the efficiency performance. Figure 6 reports the training TIME spent computing the autoencoder (Autoencoding), completing the clustering stage (Clustering), performing the nearest-neighbour search for the 2D representation of the data (Imaging) and training the 2D CNN architecture. The results show that the differences between NoPooling and MaxPooling are negligible (less than one minute for completing the training phase) in all the datasets, independently of the size of the clustering step. On the other hand, the clustering step always speeds up the nearest-neighbour search, that is performed



(a) KDDCUP99 - Training TIME

(b) UNSW-NB15 - Training TIME

(c) CICIDS2017 - Training TIME

Figure 6: Sensitivity analysis: training TIME (axis X) spent in minutes completing the training process with both the version of CLAIRE with pooling (MaxPooling) and the default version without pooling (NoPooling). The results are collected by varying the number of clusters in the clustering step (axis Y) on the datasets KDDCUP99 (Figure 6a), UNSW-NB15 (Figure 6b) and CICIDS2017 (Figure 6c).

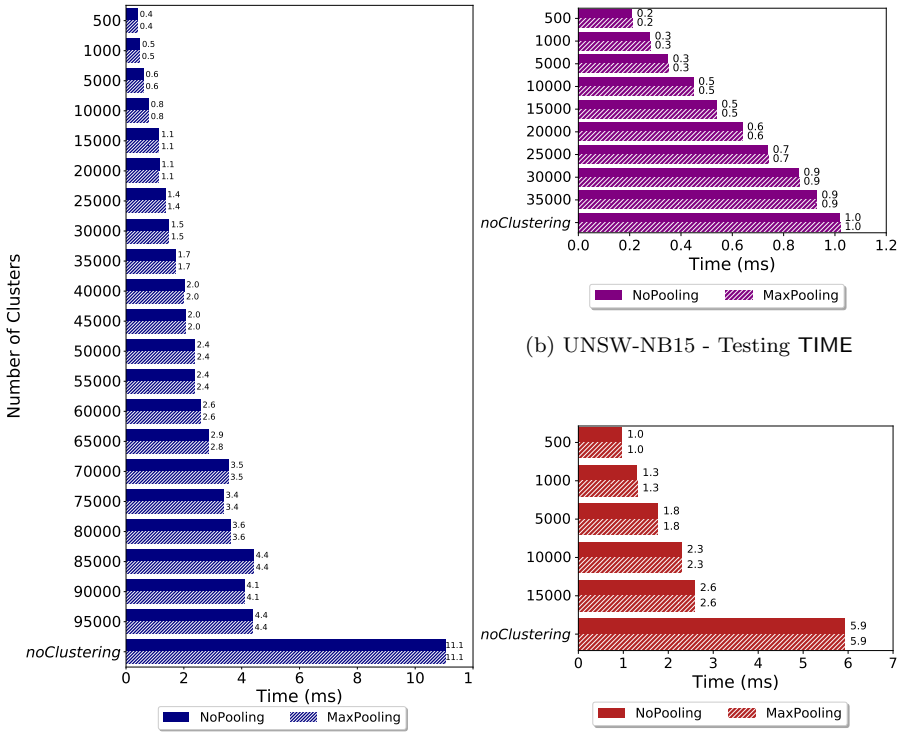
to construct the 2D representation of the training samples. In particular, the imaging stage of the configuration noClustering, which constructs the 2D data without the clustering step, is always more time-consuming than all the configurations where the clustering step is activated.

In any case, the completion of the clustering step has a time cost during the training. As regards the cost of clustering, we note that the longer time spent clustering the training set is fully counterbalanced by the shorter time spent imaging the training samples, when CLAIRE is run in the default configuration (i.e. clustering with $k = 1000$ and NoPooling) that we have identified with the F1-score analysis.

Final conclusions can be drawn from the analysis of the testing TIME. In particular, the results reported in Figure 7 highlight that the clustering step always speeds up the detection phase, i.e. the time spent, on average, using the learned intrusion detection model, to process new network traffic and classify each flow as a normal flow or an attacking flow. In general, the lower the number of clusters k, the quicker the testing phase, with the clustering stage activated.

5.4. Deep neural network hyper-parameter optimization analysis

In this Section we explore how CLAIRE is actually able to identify, during the training stage, the optimal deep neural network hyper-parameter configuration that performs well on the unseen testing sets. We recall that the hyper-parameter optimisation is done automatically for both the autoencoder and the CNN architectures (as reported in Table 2). We check the effectiveness of the hyper-parameters estimated for autoencoders by analysing the mean square error—MSE—measured between the original data and the same data encoded and decoded through the autoencoder architectures. We check the effectiveness of the hyper-parameters estimated for CNNs by analysing the accuracy—F1-score—of the classifications yielded with the CNN architectures. For each dataset we analyse these metrics computed on both the validation set (processed during the training stage to perform the hyper-parameter optimisation) and the unseen testing set.



(a) KDDCUP99 - Testing TIME

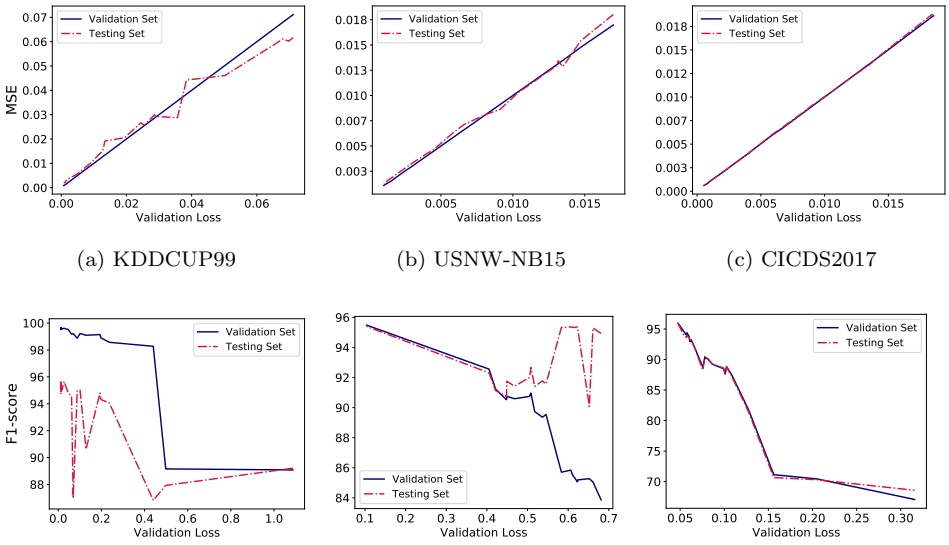
(b) UNSW-NB15 - Testing TIME

(c) CICIDS2017 - Testing TIME

Figure 7: Sensitivity analysis: average testing TIME (axis X) spent in seconds predicting each testing sample, by using the intrusion detection model learned with both the version of CLAIRE with pooling (MaxPooling) and the default version without pooling (NoPooling). The results are collected by varying the number of clusters in the clustering step (axis Y) on the datasets KDDCUP99 (Figure 7a), UNSW-NB15 (Figure 7b) and CICIDS2017 (Figure 7c).

Figures 8a-8c and Figures 8d-8f show how the MSE and F1-scores vary, respectively, concerning the architecture validation loss (mean square error in autoencoders and binary-cross entropy in CNNs). Plotted data are determined for CLAIRE in the default configuration (with $k = 1000$ and NoPooling) and with the tree-structured Parzen hyper-parameter estimator run on the range of values described in Table 2. These results confirm that the automatically selected hyper-parameter configuration (i.e. the one measuring the lowest architecture loss on the validation set) is optimal on both the validation set (where it is selected) and the unseen testing set for both the autoencoders and the CNNs. In fact, the architectures achieving the lowest validation loss exhibit the lowest MSE (in autoencoders) and the highest F1-score (in CNNs), respectively, in both the validation and the testing set. As expected, the MSE curves monotonically increase with the autoencoder loss. On the other hand, the F1-score curves approximate a monotonically decreasing trend with the CNN loss, except for the UNSW-NB15 testing set. In any case, also in UNSW-NB15, the hyper-parameter configuration, selected with the lowest validation loss, achieves the highest F1-score in both the validation and testing set.

Upon the completion of this analysis of the viability of the hyper-parameter optimization, we perform the Friedman-Nemenyi statistical test [55]. This is a non-parametric test commonly used to compare multiple algorithms over multiple data. It ranks the algorithms for each data set separately, so the best performing algorithm is given rank of 1, the second best rank 2 and so on [55]. We perform the test on KDDCUP99, UNSW-NB15 and CICIDS2017 to compare the twenty configurations of CLAIRE, which are evaluated using the tree-structured Parzen hyper-parameter estimator. We sort these configurations in the ascending order by the validation loss, so that CLAIRE1 is the best configuration achieving the lowest validation loss (i.e. the configuration automatically selected as the final intrusion detection model), while CLAIRE20 is the worst configuration achieving the highest validation loss. Figures 9a and 9b rank the configurations according to the result of the Friedman-Nemenyi statistical test done on the F1-score, measured on the validation set and the testing set, re-



(a) KDDCUP99 (b) USNW-NB15 (c) CICDS2017

(d) KDDCUP99 (e) USNW-NB15 (f) CICDS2017

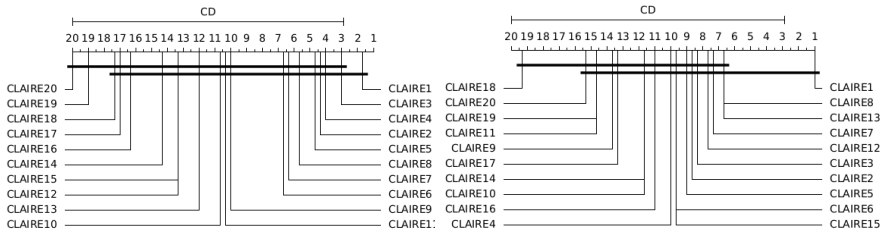
Figure 8: The MSE of the autoencoder data reconstruction (Figures 8a-8c) and the F1-score of the CNN classifications (Figures 8d-8f) measured on both the validation set and the testing set (axis Y), with respect to the architecture validation loss (axis X). The results are collected for KDDCUP99, UNSW-NB15 and CICIDS2017 datasets on the range of hyper-parameter values reported in Table 2, and explored according to the hyper-parameter optimization done during the training stage of CLAIRE (with $k = 1000$ and NoPooling).

spectively. The results of the test confirm that the configuration of CLAIRE that is automatically selected with the tree-structured Parzen hyper-parameter estimator is ranked the highest both on the validation set and on the testing set.

5.5. Competitor analysis

We compare the accuracy performance achieved by CLAIRE to that of several competitors, selected from the recent state-of-the-art literature. In particular, we consider the following competitors:

- 1D Convolutional Neural Network-based competitors (1D CNN): CNN-1D [49],CNN4 [16];
- 2D Convolutional Neural Network-based competitors (2D CNN): Grey-scale



(a) F1-score (Validation Set) (b) F1-score (Testing Set)

Figure 9: The Friedman-Nemenyi test calculated on the F1-score measured on both the validation set (Figure 9a) and the testing set (Figure 9b) of KDDCUP99, UNSW-NB15 and CICIDS2017. The F1-score is measured on the configurations of CLAIRE, which are trained on the range of hyper-parameter values reported in Table 2 and explored according to the hyper-parameter optimization done during the training stage of CLAIRE (with $k = 1000$ and NoPooling). The compared configurations of CLAIRE are sorted in ascending order by the validation loss.

[19], [20] and RGB [20];

- Long Short-Term Memory Neural Network-based competitor (LSTM): BLSTM [56];
- Recurrent Neural Network-based competitor (RNN): BRNN [56];
- Deep Neural Network-based competitors (DNN): DNN 4 Layers [14], DNN-3 [4], DNN4 [16], DBN [47] , A+DBN [47], MLP [49], WnD [15], MLP [15], SAE [15], DAE [57], AIDA [12] and RBM [15];
- Clustering+Deep Neural Network-based competitor (Clustering+DNN): MDPCA-DBN [29];
- GAN Network-based competitors (1D CNN): AnoGAN [58] [59] and ALAD [59].

We note that both the 2D CNN-based competitors and the Clustering+DNN-based competitor are the most related to CLAIRE. In fact, the 2D CNN-based competitors, similarly to CLAIRE, experiment various 2D encoding techniques,

in order to transform network flows into imagery data and train a 2D CNN architecture. On the other hand, the Clustering+DNN-based competitor combines clustering with deep learning. However, none of these competitors use neighbouring information (in combination with cluster centres) within the 2D data transformation, which is the main novelty of this study.

Table 8: Competitor analysis: The accuracy metrics of the competitors have been collected from the reference papers. "-" denotes that no value is reported in the reference paper.

dataset	category	description	Accuracy	F1-score
KDDCUP99	CLAIRE	2D CNN	**93.58**	**95.90**
	CNN4 [16]	1D CNN	92.47	-
	DNN4Layers [14]	DNN + Text-based encoding	93.00	95.50
	DNN-3 [4]	DNN	93.00	95.50
	DNN4 [16]	DNN	92.88	-
	DBN [47]	DNN	91.40	-
	A+DBN [47]	Autoencoder + DBN	92.10	-
	AIDA [12]	Autoencoder + MLP	92.36	95.04
	BLSTM [56]	LSTM	-	93.27
	BRNN [56]	RNN	-	91.82
	AnoGAN [58] [59]	GAN	-	88.65
	ALAD [59]	GAN	-	95.01
UNSW-NB15	CLAIRE	2D-CNN	**93.52**	**95.40**
	CNN-1D [49]	1D CNN	89.80	91.30
	Grey-scale[19][20]	2D CNN	80.00	84.00
	RGB [20]	2D CNN	83.00	86.50
	DNN4Layers [14]	DNN + Text-based encoding	76.50	90.10
	MLP [49]	DNN	86.60	88.90
	WnD [15]	DNN + Embedding	91.20	-
	MLP [15]	MLP	86.70	-
	SAE [15]	Autoencoder	88.20	-
	DAE [57]	Autoencoder + DNN	92.40	-
	AIDA [12]	Autoencoder + MLP	90.54	92.71
	MDPCA-DBN [29]	Clustering + DNN	90.18	91.49
	RBM [15]	RBM	87.10	-
CICIDS2017	CLAIRE	2D-CNN	**98.01**	**95.20**
	Grey-scale [19][20]	2D-CNN	-	82.00
	RGB [20]	2D-CNN	-	89.00
	AIDA [12]	Autoencoder + MLP	94.50	85.80

For all the methods in this comparative study, we collect the Accuracy and

F1-score, as these metrics are commonly provided in the reference studies. The results collected are reported in Table 8 for all the datasets. The results of CLAIRE are collected in the default configuration ($k = 1000$ and NoPooling). These results show that CLAIRE outperforms its competitors, comprising both the 2D CNN-based competitors (CNN4 evaluated on KDDCUP99, Grey-scale and RGB evaluated on UNSW-NB15 and CICIDS2017 in the reference studies) and the Clustering+DNN-based competitor (MDPCA-DBN evaluated on UNSW-NB15 in the reference study).

This empirical result contributes to assessing the significance and novelty of CLAIRE with respect to the key ideas (definition of a specific 2D representation of the network flows, use of a 2D CNN architecture and consideration of clustering information) which it puts forward. Although these ideas have in some way been experimented in the literature, they are developed in this study under a new algorithmic umbrella that has proved to effectively outperform various competitors including those recently developed with the same background.

5.6. Ablation study

We complete this experimental study by performing an ablation study. This study aims at investigating:

- how the proposed method can take advantage of coupling the information, synthesised through the cluster-based nearest neighbour search, to the architecture with 2D convolutions;
- how the proposed method (and also its baselines) is sensitive to the size of the imbalance condition.

To investigate the result of both the cluster-based nearest neighbour search and the 2D convolutions, we consider four configuration architectures as baselines. These are in turn defined by removing the (cluster-based) nearest neighbour information and the convolutions from the whole architecture of CLAIRE. In this way, we define four baseline architectures that are run using the same configuration (e.g. activation function and loss function) adopted to run CLAIRE

(see the description in Section 4). In all the baselines, convolution layers Conv2D are run with NoPooling and clustering is run with $k = 1000$ (default configuration). In particular, the baseline architectures are defined as follows:

- NN: $\mathbf{X}' \rightarrow \mathsf{FC}(256) \rightarrow \mathsf{FC}(1024) \rightarrow \mathsf{FC}(2)$. This architecture, that consists of the last 3 fully-connected layers of the CLAIRE architecture, takes as input samples $\mathbf{x}' \in \mathbf{X}'$, which are the training samples spanned on the feature space $\mathbf{X}'$.

- N+NN: $\mathbf{X}' \oplus \mathbf{X}' \oplus \mathbf{X}' \rightarrow \mathsf{FC}(256) \rightarrow \mathsf{FC}(1024) \rightarrow \mathsf{FC}(2)$. This architecture takes as input samples $\mathbf{x}' \oplus \mathbf{x}'^{+} \oplus \mathbf{x}'^{-} \in \mathbf{X}' \oplus \mathbf{X}' \oplus \mathbf{X}'$, where $\mathbf{x}'$, $\mathbf{x}'^{+}$ and $\mathbf{x}'^{-}$ are spanned on $\mathbf{X}'$ and row-concatenated in a 1D vector. $\mathbf{x}'^{+}$ and $\mathbf{x}'^{-}$ are the nearest neighbour (normal and attacking) samples of $\mathbf{x}'$, searched in the original training set (without clustering).

- C+N+NN: the architecture is like that of N+NN, but it performs the clustering step for the nearest neighbour search. It takes as input $\mathbf{x}' \oplus \mathbf{x}'^{+} \oplus \mathbf{x}'^{-}$ row-concatenated, where $\mathbf{x}'^{+}$ and $\mathbf{x}'^{-}$ are selected as the nearest neighbour (normal and attacking) cluster centres of $\mathbf{x}'$.

- N+CNN: $\begin{bmatrix} \mathbf{X}' \\ \mathbf{X}'^{+} \\ \mathbf{X}'^{-} \end{bmatrix} \rightarrow \mathsf{Conv2D}(32) \rightarrow \mathsf{Conv2D}(64) \rightarrow \mathsf{Conv2D}(128) \rightarrow \mathsf{FC}(256) \rightarrow \mathsf{FC}(1024) \rightarrow \mathsf{FC}(2)$. This architecture takes as input the training samples in the 2D representation, where $\mathbf{x}'$, $\mathbf{x}'^{+}$ and $\mathbf{x}'^{-}$ are feature vectors spanned on $\mathbf{X}'$ and assigned to separate rows of a 2D grid. $\mathbf{x}'^{+}$ and $\mathbf{x}'^{-}$ are searched without the clustering step activated.

We evaluate the performance of CLAIRE, NN, N+NN, C+N+NN and N+CNN on CICIDS2017, where the data have been collected in an imbalance scenario (the one to be expected in many real world networks), consisting of 80% normal flows and 20% attacks. The performance of the compared algorithms is measured in terms of Accuracy and F1-score. The overall results, reported in Table 9, show that CLAIRE is more accurate than all its baselines. This confirms the

Table 9: Ablation study: Accuracy and F1-score measured on CICIDS2017Test for CLAIRE, NN, N+NN, C+N+NN and N+CNN. The best results are in bold.

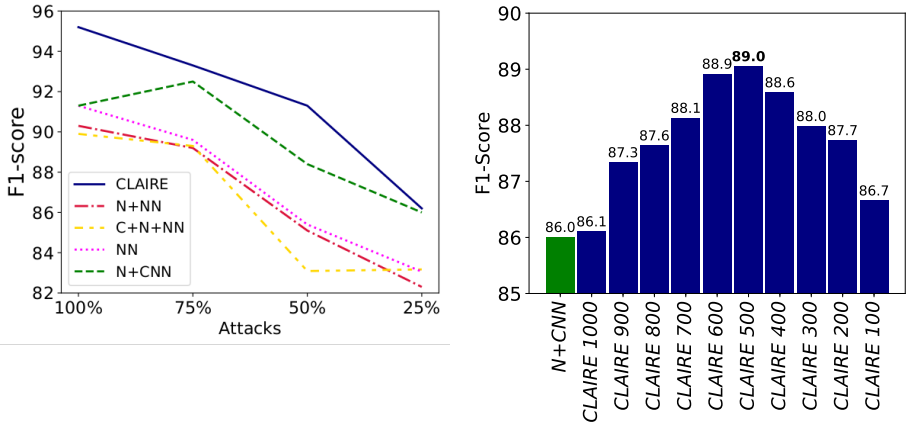
	Architecture				
	CLAIRE	NN	N+NN	C+N+NN	N+CNN
Accuracy	**98.01**	96.50	96.10	95.99	96.43
F1-score	**95.20**	91.34	90.29	89.98	91.34

effectiveness of combining cluster-based nearest-neighbour information and 2D convolutions, in order to gain accuracy in an intrusion detection task. Specific considerations can be made from the detailed analysis of the results.

In particular, we note that the use of the additional information, obtained using the nearest-neighbour search decoupled from the 2D convolutions, cannot guarantee an improvement in the accuracy. In fact, both N-NN and C+N+NN, which process the nearest neighbour information without the 2D convolutions, perform worse than NN, that ignores the nearest neighbour information. On the other hand, putting clustering aside, 2D convolutions improve the intrusion detection accuracy (N+CNN outperforms NN). This highlights that the accuracy gained by processing the nearest neighbour information is due to the ability to compute 2D convolutions on the 2D representation that we have introduced to handle the nearest data (instead of computing traditional fully-connected layers on vector data, built by concatenation). In any case, our analysis confirms, once again, that the nearest neighbour search, decoupled from the clustering step, may suffer from overfitting, with a decrease in the accuracy performance. In fact, the superiority of CLAIRE, due to the 2D convolutions on the nearest neighbour data, also depends on its ability to perform the nearest neighbour search which uses cluster centres as a generalisation of normal and attacking network flows (CLAIRE outperforms N+CNN).

To explore the sensitivity of the overall accuracy of the compared architectures to the size of the imbalance condition, we consider four trials with 100% (baseline), 75%, 50% and 25% attacks, respectively.

The F1-score of CLAIRE, NN, N+NN, C+N+NN and N+CNN, collected by



(a) F1-score by varying the number of attacks (b) F1-score on CICIDS2017Attacks25%

Figure 10: Imbalance condition analysis: F1-score of CLAIRE NN, N+NN, C+N+NN and N+CNN by varying the number of attacks in CICIDS2017 (Figure 10a). F1-score of N+CNN and CLAIRE on CICIDS2017Attacks25% with CLAIRE run by setting the number of clusters on normal samples (k_{normal}) equal to 1000 (default value) and by varying the number of clusters on attacks (k_{attack}) from 1000 to 100 (Figure 10b)

.

diminishing the number of attacks, is plotted in Figure 10a. Diminishing the number of attacks (and consequently stressing the imbalance condition) leads to a decrease in the F1-score of all the compared algorithm. We note that CLAIRE continues outperforming its baselines independently of the balance degree in the training set. The difference between CLAIRE and N+CNN becomes negligible when only 25% of the attacks (i.e. 5000 training attacks) are considered (although, also in this configuration, CLAIRE slightly outperforms N+CNN). We recall that the clustering step has been performed in the default configuration (with $k = 1000$). Clustering training 5000 attacks in 1000 clusters diminishes the generalization power of clustering, by reducing the ability to limit overfitting when performing the nearest neighbour search. This justifies the close performance of CLAIRE and N+CNN on CICID2017Attacks25%.

We empirically verify the validity of our interpretation of the performance of CLAIRE and N+CNN on CICIDS2017Attacks25%, by studying the sensi-

tivity of the F1-score of CLAIRE to the number of attacking clusters in CICIDS2017Attacks25%. To this aim, we perform the clustering step by maintaining the number of clusters of normal samples (majority class) set equal to 1000 (as in the default configuration), and by varying the number of clusters of attacks (minority class) between 1000 and 100. This allows us to check how the clustering step can gain in generalisation power on the attacking samples, independently of the imbalance condition. The results, reported in Figure 10b, confirm that CLAIRE gains in F1-score when we increase the generalization power of the clustering operation performed on the attacks (i.e. by diminishing k_{attack}). In fact, all the clustering configurations with $k_{attack} < 1000$ (and $k_{normal} = 1000$) augment the accuracy gap between CLAIRE and its variant without clustering — N+CNN — in the trial with CICIDS2017Attacks25%. The best improvement occurs with $k_{attack} < 500$. This result paves the way for future investigations concerning the size of clustering in the presence of the imbalance condition.

6. Conclusions

Conventional machine learning methods, proposed in the intrusion detection literature, are often not effective at detecting unforeseen network attacks. In this study we address the network intrusion detection problem by proposing a multi-stage methodology, denoted as CLAIRE, that combines nearest neighbour search, clustering and convolutional neural networks. In particular, we propose a novel encoding method, that accounts for the nearest neighbouring knowledge, to map the one-dimensional feature vector representation of the network flows into a two-dimensional imaging representation of these data. 2D data are used to train a CNN-based intrusion detection model.

We evaluate the effectiveness of the proposed methodology using three benchmark datasets (KDDCUP99, USW-NB15 and CICIDS2017), which contain network flows collected in different years. The experimental analysis proves the viability of the multiple stages of the proposed methodology. In addition, it

proves that CLAIRE gains accuracy compared to several, recently defined, state-of-the-art competitors using deep learning and/or clustering.

As future work, we plan to explore the effectiveness of the described encoding method, by carrying out experiments using common CNN architectures like those based on ResNet, Inception or LeNet. Moreover, we plan to explore how the RGB mapping, that was already explored in [20], can be used in combination with the neighbouring search as a base for an enhanced imaging representation of the network traffic. We also intend to examine in depth the criteria used to customise the parameter set-up of the clustering step in the presence of imbalance conditions. Finally, we plan to extend this investigation to the classification of the intrusion families.

Acknowledgments

We acknowledge the support of the MIUR-Ministero dell'Istruzione dell'Università e della Ricerca through the project "TALIsMan -Tecnologie di Assistenza personALizzata per il Miglioramento della quAlità della vitA" (Grant ID: ARS01_01116), PON RI 2014-2020 funding scheme, as well as the project "Modelli e tecniche di data science per la analisi di dati strutturati", funded by the University of Bari "Aldo Moro". The authors wish to thank Lynn Rudd for her help in reading the manuscript and Michele Gargaro for his support in the implementation of clustering step of the software.

Appendix A. Encoder feature permutation

Let $\mathbf{X}'$ be the output encoder layer of the autoencoder architecture. The permutation of the $\mathbf{X}'$ columns is completed through a process that consists of an initialization step and an iterative step.

In the initialisation step, we determine columns X'_i, X'_j, X'_h, X'_k of $\mathbf{X}'$, such

that:

$$X'_i, X'_j = \underset{X'_i \in \mathbf{X}', X'_j \in \mathbf{X}', X'_i \neq X'_j}{\arg\min} distance(X'_i, X'_j), \quad (A.1)$$

$$X'_h = \underset{X'_h \in \mathbf{X}', X'_h \neq X'_i, X'_h \neq X'_j}{\arg\min} distance(X'_i, X'_h), \quad (A.2)$$

$$X'_k = \underset{X'_k \in \mathbf{X}', X'_k \neq X'_i, X'_k \neq X'_j}{\arg\min} distance(X'_j, X'_k), \quad (A.3)$$

where $distance()$ is computed with the Euclidean distance. If $distance(X'_i, X'_h) \leq distance(X'_j, X'_k)$, we swap X'_1 with X'_j, X'_2 with X'_i and X'_3 with X'_h, otherwise we swap X'_1 with X'_i, X'_2 with X'_j and X'_3 with X'_k.

In the iterative step, we start with $i = 3$, then we determine column X'_j, such that:

$$X'_j = \underset{X'_j \in \mathbf{X}', j > i}{\arg\min} distance(X'_i, X'_j). \quad (A.4)$$

We swap X'_{i+1} with X'_j in $\mathbf{X}'$. We increment i by one and iterate this step until $i = \sharp\mathbf{X}' - 1$, where $\sharp\mathbf{X}'$ is the number of columns of $\mathbf{X}'$.

Appendix B. Convolution Neural Network

A Convolutional Neural Network (CNN) is a special kind of fully-connected feed-forward neural network, introducing three extra concepts: local filters (convolution), pooling and weight sharing [7]. This deep learning architecture is commonly used in the literature for processing grid-like topology data, such as the imagery data – two dimensional arrays of pixels – that have been produced here as a representation of the network flows. A convolutional layer convolves a set of filters that are replicated along the whole input to process small local parts of the input [30]. A pooling layer generates a lower resolution version of the convolutional layer output. The pooling operation is typically the average or the maximum. The use of pooling may add shift-invariance and tolerance to minor differences in the positions of the patterns in the input [60]. Higher layers use broader filters, that work on lower resolution inputs to process more complex parts of the input. Top fully-connected layers finally combine inputs from all positions to classify the overall inputs.

Specifically, given a two-dimensional input image $\mathbf{E}$, the k-th feature map at location (i, j), in a given convolutional l-th layer, is determined by the weight matrix $\mathbf{W}_k^l$ and the bias vector $\mathbf{b}_k^l$ of the k-th filter on the l-th layer, with a non-linear activation function $\sigma()$, such that:

$$h_{i,j,k}^l = \sigma(\mathbf{W}_k^l * \mathbf{E}_{i,j}^l + \mathbf{b}_k^l), \tag{B.1}$$

where $\mathbf{E}_{i,j}^l$ is the input patch centered at location (i, j) of the l-th layer and $*$ represents the convolution function. The kernel $\mathbf{W}_k^l$ is shared for each possible location (i, j), thus reducing the model complexity and making the network easier to train. A pooling layer, placed after a convolutional layer, aims to achieve *shift-invariance*, by reducing the resolution of the feature map $\mathbf{h}_{:,:,k}^l$ to:

$$y_{i,j,k}^l = pool(h_{m,n,k}^l), \forall (m, n) \in \mathcal{R}_{ij}, \tag{B.2}$$

where $\mathcal{R}_{ij}$ is a local neighbourhood around location (i, j). There are two common types of Pooling described in the literature: Max Pooling and Average Pooling. Max Pooling returns the maximum value from the portion of the image covered by the kernel, while Average Pooling calculates the average value for each path on the feature map. Max Pooling has recently been preferred to Average Pooling, as it allows the architecture to preserve the most informative features by offering better translation invariance [54].

The final set of layers in CNNs is fully-connected. In a fully-connected layer every neuron is connected to each hidden state in the previous layer, like a traditional feed-forward neural network. In particular, the output layer of a CNN is designed in an application-specific way, using an activation function such as logistic, softmax or linear activation [7]. In a classification task, like the one addressed in this study, softmax is usually chosen as the activation function.

References

[1] B. Dong, X. Wang, Comparison deep learning method to traditional methods using for network intrusion detection, in: 2016 8th IEEE International

Conference on Communication Software and Networks (ICCSN), IEEE, 2016, pp. 581–585.

[2] S. Naseer, Y. Saleem, S. Khalid, M. K. Bashir, J. Han, M. M. Iqbal, K. Han, Enhanced network anomaly detection based on deep neural networks, IEEE Access 6 (2018) 48231–48246.

[3] A. A. Diro, N. Chilamkurti, Distributed attack detection scheme using deep learning approach for internet of things, Future Generation Computer Systems 82 (2018) 761 – 768.

[4] R. K. Vigneswaran, R. Vinayakumar, K. P. Soman, P. Poornachandran, Evaluating shallow and deep neural networks for network intrusion detection systems in cyber security, in: 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), 2018, pp. 1–6.

[5] M. P. A. Drewek-Ossowicka, J. Rumiński, A survey of neural networks usage for intrusion detection systems, Journal of Ambient Intelligence and Humanized Computing (2020) 1–18.

[6] D. S. Berman, A. L. Buczak, J. S. Chavis, C. L. Corbett, A survey of deep learning methods for cyber security, Information) 10 (4) (2019) 1–35.

[7] C. C. Aggarwal, Neural Networks and Deep Learning - A Textbook, Springer, 2018.

[8] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (2015) 436–444.

[9] E. de la Hoz Correa, E. de la Hoz Franco, A. Ortiz, J. Ortega, B. Prieto, PCA filtering and probabilistic SOM for network intrusion detection, Neurocomputing 164 (2015) 71–81.

[10] R. R. Reddy, Y. Ramadevi, K. V. N. Sunitha, Effective discriminant function for intrusion detection using SVM, in: ICACCI, IEEE, 2016, pp. 1148–1153.

[11] R. A. R. Ashfaq, X. Wang, J. Z. Huang, H. Abbas, Y. He, Fuzziness based semi-supervised learning approach for intrusion detection system, Inf. Sci. 378 (2017) 484–497.

[12] G. Andresini, A. Appice, N. Di Mauro, C. Loglisci, D. Malerba, Exploiting the auto-encoder residual error for intrusion detection, in: 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS PW), IEEE, 2019, pp. 281–290.

[13] S. A. Althubiti, E. M. Jones, K. Roy, Lstm for anomaly-based network intrusion detection, in: 2018 28th International Telecommunication Networks and Applications Conference (ITNAC), IEEE Computer Society, 2018, pp. 1–3.

[14] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, S. Venkatraman, Deep learning approach for intelligent intrusion detection system, IEEE Access 7 (2019) 41525–41550.

[15] J. Yan, D. Jin, C. W. Lee, P. Liu, A comparative study of off-line deep learning based network intrusion detection, in: 10th International Conference on Ubiquitous and Future Networks, 2018, pp. 299–304.

[16] Y. He, Identification and processing of network abnormal events based on network intrusion detection algorithm, I. J. Network Security 21 (2019) 153–159.

[17] D. Kwon, K. Natarajan, S. C. Suh, H. Kim, J. Kim, An empirical study on network anomaly detection using convolutional neural networks, in: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), IEEE, 2018, pp. 1595–1598.

[18] A. Sharma, P. Malacaria, M. Khouzani, Malware detection using 1-dimensional convolutional neural networks, in: European Symposium on Security and Privacy Workshops, 2019, pp. 247–256.

[19] Z. Li, Z. Qin, K. Huang, X. Yang, S. Ye, Intrusion detection using convolutional neural networks for representation learning, in: ICONIP, Springer International Publishing, 2017, pp. 858–866.

[20] T. Kim, S. C. Suh, H. Kim, J. Kim, J. Kim, An encoding technique for cnn-based network anomaly detection, in: 2018 IEEE International Conference on Big Data (Big Data), IEEE, 2018, pp. 2960–2965.

[21] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, 2016, pp. 770–778.

[22] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2015, pp. 1–9.

[23] K. Millar, A. Cheng, H. G. Chew, C.-C. Lim, Using convolutional neural networks for classifying malicious network traffic, Deep Learning Applications for Cyber Security (2019) 103–126.

[24] F. Salo, M. Injadat, A. Moubayed, A. B. Nassif, A. Essex, Clustering enabled classification using ensemble feature selection for intrusion detection, in: 2019 International Conference on Computing, Networking and Communications (ICNC), IEEE, 2019, pp. 276–281.

[25] G. Andresini, A. Appice, D. Malerba, Dealing with class imbalance in android malware detection by cascading clustering and classification, in: Complex Pattern Mining - New Challenges, Methods and Applications, Vol. 880 of Studies in Computational Intelligence, Springer, 2020, pp. 173–187.

[26] K. Peng, V. C. M. Leung, Q. Huang, Clustering approach based on mini batch k-means for intrusion detection system over big data, IEEE Access 6 (2018) 11897–11906.

[27] H. Benaddi, K. Ibrahimi, A. Benslimane, Improving the intrusion detection system for NSL-KDD dataset based on pca-fuzzy clustering-knn, in: 6th International Conference on Wireless Networks and Mobile Communications, WINCOM 2018, Marrakesh, Morocco, October 16-19, 2018, IEEE, 2018, pp. 1–6.

[28] T. Kenaza, K. Bennaceur, A. Labed, An efficient hybrid svdd/clustering approach for anomaly-based intrusion detection, in: Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC '18, ACM, New York, NY, USA, 2018, pp. 435–443.

[29] Y. Yang, K. Zheng, C. Wu, X. Niu, Y. Yang, Building an effective intrusion detection system using the modified density peak clustering algorithm and deep belief networks, Applied Sciences 9 (2) (2019) 238.

[30] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.

[31] A. K. Jain, M. N. Murty, P. J. Flynn, Data clustering: A review, ACM Comput. Surv. 31 (3) (1999) 264–323. doi:10.1145/331499.331504.

[32] D. Sculley, Web-scale k-means clustering, in: Proceedings of the 19th International Conference on World Wide Web, WWW '10, ACM, New York, NY, USA, 2010, pp. 1177–1178.

[33] O. İrsoy, E. Alpaydın, Unsupervised feature extraction with autoencoder trees, Neurocomputing 258 (2017) 63–73. doi:10.1016/j.neucom.2017.02.075.

[34] K. Peng, V. C. M. Leung, Q. Huang, Clustering approach based on mini batch kmeans for intrusion detection system over big data, IEEE Access 6 (2018) 11897–11906.

[35] K. He, J. Sun, Convolutional neural networks at constrained time cost, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 5353–5360.

[36] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, in: Proceedings of the IEEE, 1998, pp. 2278–2324.

[37] J. Bergstra, D. Yamins, D. D. Cox, Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, in: ICML, 2013, pp. 115–123.

[38] K. Macek, Pareto principle in datamining: an above-average fencing algorithm, in: Acta polytechnica, Vol. 48(6), 2008, pp. 55–59.

[39] V. L. Parsons, Stratified sampling, in: Wiley StatsRef: Statistics Reference Online, American Cancer Society, 2017, pp. 1–11.

[40] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks., in: AISTATS, JMLR.org, 2011, pp. 315–323.

[41] J. T. Springenberg, A. Dosovitskiy, T. Brox, M. A. Riedmiller, Striving for simplicity: The all convolutional net, in: Y. Bengio, Y. LeCun (Eds.), 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings, 2015.

[42] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: ICLR, 2014.

[43] M. Labonne, A. Olivereau, B. Polve, D. Zeghlache, A cascade-structured meta-specialists approach for neural network-based intrusion detection, 16th Annual Consumer Communications & Networking Conference (2019) 1–6.

[44] L. Dan, C. Dacheng, J. Baihong, S. Lei, G. Jonathan, N. See-Kiong, Madgan: Multivariate anomaly detection for time series data with generative adversarial networks, in: Artificial Neural Networks and Machine Learning, 2019, pp. 703–716.

[45] H. Zenati, C. S. Foo, B. Lecouat, G. Manek, V. R. Chandrasekhar, Efficient gan-based anomaly detection, ArXiv abs/1802.06222.

[46] X. Qu, L. Yang, K. Guo, L. Ma, T. Feng, S. Ren, M. Sun, Statistics-enhanced direct batch growth self-organizing mapping for efficient dos attack detection, IEEE Access 7 (2019) 78434–78441.

[47] Y. Li, R. Ma, R. Jiao, A hybrid malicious code detection method based on deep learning, in: International journal of security and its applications, Vol. 9, 2015, pp. 205–216.

[48] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, S. Venkatraman, Deep learning approach for intelligent intrusion detection system, IEEE Access 7 (2019) 41525–41550.

[49] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, J. Lloret, Shallow neural network with kernel approximation for prediction problems in highly demanding data networks, Expert Systems with Applications 124 (2019) 196 – 208.

[50] Y. Zhang, X. Chen, L. Jin, X. Wang, D. Guo, Network intrusion detection: Based on deep hierarchical network and original flow data, IEEE Access 7 (2019) 37004–37016.

[51] P. Angelo, A. Costa Drummond, Adaptive anomaly-based intrusion detection system using genetic algorithm and profiling, Security and Privacy 1 (4) (2018) 1–13.

[52] A. Binbusayyis, T. Vaiyapuri, Identifying and benchmarking key features for cyber intrusion detection: An ensemble approach, IEEE Access 7 (2019) 106495–106513.

[53] A. Ahmim, L. Maglaras, M. A. Ferrag, M. Derdour, H. Janicke, A novel hierarchical intrusion detection system based on decision tree and rules-based models, in: 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS), IEEE, 2019, pp. 228–233.

[54] A. Géron, Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems, O'Reilly Media, 2017.

[55] J. Demšar, Statistical comparisons of classifiers over multiple data sets, JMLR.org 7 (2006) 1–30.

[56] A. Elsherif, Automatic intrusion detection system using deep recurrent neural network paradigm, in: J. Inf. Secur. Cybercrimes Res. (JISCR), 2018, pp. 28–41.

[57] M. AL-Hawawreh, N. Moustafa, E. Sitnikova, Identification of malicious activities in industrial internet of things based on deep learning models, Journal of Information Security and Applications 41 (2018) 1 – 11.

[58] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, G. Langs, Unsupervised anomaly detection with generative adversarial networks to guide marker discovery, in: Information Processing in Medical Imaging, Springer International Publishing, Cham, 2017, pp. 146–157.

[59] H. Zenati, M. Romain, C. S. Foo, B. Lecouat, V. R. Chandrasekhar, Adversarially learned anomaly detection, 2018 IEEE International Conference on Data Mining (ICDM) (2018) 727–736.

[60] D. Scherer, A. Müller, S. Behnke, Evaluation of pooling operations in convolutional architectures for object recognition, in: K. Diamantaras, W. Duch, L. S. Iliadis (Eds.), Artificial Neural Networks – ICANN 2010, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 92–101.