Deep Learning and Traffic Classification: Lessons learned from a commercial-grade dataset with hundreds of encrypted and zero-day applications

Lixuan Yang, Alessandro Finamore, Feng Jun, Dario Rossi
Huawei Technologies, France

***Abstract*—The increasing success of Machine Learning (ML) and Deep Learning (DL) has recently re-sparked interest towards traffic classification. While classification of *known* traffic is a well investigated subject with supervised classification tools (such as ML and DL models) are known to provide satisfactory performance, detection of *unknown* (or zero-day) traffic is more challenging and typically handled by unsupervised techniques (such as clustering algorithms).**

In this paper, we share our experience on a commercial-grade DL traffic classification engine that is able to (i) identify known applications from encrypted traffic, as well as (ii) handle unknown zero-day applications. In particular, our contribution for (i) is to perform a thorough assessment of state of the art traffic classifiers in commercial-grade settings comprising *few thousands* of very fine grained application labels, as opposite to the *few tens* of classes generally targeted in academic evaluations. Additionally, we contribute to the problem of (ii) detection of zero-day applications by proposing a novel technique, tailored for DL models, that is significantly more accurate and light-weight than the state of the art.

Summarizing our main findings, we gather that (i) while ML and DL models are both equally able to provide satisfactory solution for classification of known traffic, however (ii) the non-linear feature extraction process of the DL backbone provides sizeable advantages for the detection of unknown classes.

***Index Terms*—Machine learning, Data mining and (big) data analysis, Network monitoring and measurements.**

I. Introduction

Classification of Internet traffic is a well investigated subject, whose research interest started in the early 2000s, to supplant light packet inspection (i.e., port-based) and deep packet inspection (i.e., payload based) technologies with statistical tools able to characterize broad traffic classes, and the specific applications within each class. Seminal works such as [1], ignited a *first wave* of classification approaches [2]–[7] essentially focused on extracting features for classifying a relatively small set of applications, relying on classic Machine Learning (ML) approaches based on careful –but human intensive– feature engineering processes. This first wave culminated with very simple yet effective techniques, referred to as "early traffic classification" [3], [4] that readily used time series information (e.g., the size and direction of the first few packets in a flow) to take classification decisions.

The tremendous successes of Convolutional Neural Networks (CNN) in the image recognition field [8] ignited a *second wave* of traffic classification approaches leveraging Deep Learning (DL) techniques [9]–[18]. DL is becoming particularly appealing in reason of domain-specific CNN hardware accelerators (known as "tensor processing units") that started appearing in the last few years, and make CNN a viable and appealing option for real-time traffic classification [19]. In reason of the tremendous push toward encryption in the Post-Snowden era, this second wave of research is particularly relevant since industrial players are now actively looking at deploying statistical classification approaches – that so far mostly remained an academic exercise, as recently pointed out in [20], which is due to a gap between the industrial interests and the attention of academic research.

To understand why this happens, we recall that, with reference to Fig.1, a classification engine needs to perform two functions: namely a application identification function $f(x)$ and a zero-day application detection function $g(x)$. Shortly, the goal of $\ell = f(x)$ is to determine from an input x an application label $\ell \in [1, K]$ among a set of K known applications. Supervised ML/DL techniques are well suited to learn the function $f(x)$, in a process called training. However, whereas commercial DPI tools are able to handle *hundreds to thousands* of application classes, statistical techniques developed in the academic world consider only a *few tens of classes* – which is significantly simpler than commercial needs.

Second, and most important, the application landscape keeps evolving, for which commercial engines need to be able to detect *detect zero-day applications*, which is the purpose of a function $g(x)$: its goal is to assess whether the supervised label $\ell = f(x)$ should be rejected, since the sample x likely belongs to a class that was never presented during training of $f(x)$, overriding the label to a $\ell = 0$ zero-day class. Clearly, DL and CNN are inherently limited, as any supervised ML classification techniques, for which an unsupervised technique such as clustering is generally used for this crucial task – which is recognized as a major blocking point to the deployment of statistical traffic classification [20].

In this work, we make two major contributions. Our first contribution is to share important insights concerning $f(X)$ gained from real deployments. Here, rather than proposing new techniques, our contribution is to contrast state of the art ML and DL techniques on exactly the same input (packets size time series for "early traffic" classification), by leveraging a commercial-grade dataset comprising tens of millions of flows, and thousands of application labels – significantly larger that what typical can be found in academic literature.

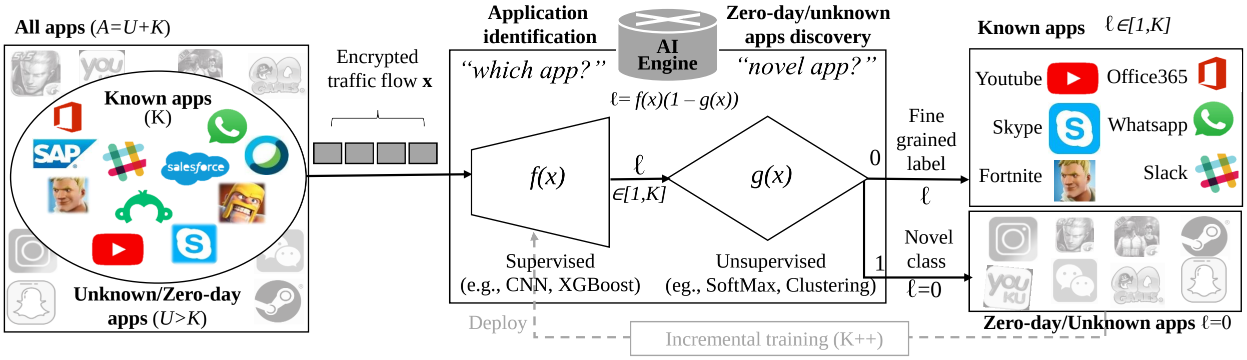


Fig. 1: Synoptic of known application identification and zero-day application discovery.

Second, and most importantly, we systematically analyze and thoroughly compare techniques for zero-day application detection $g(x)$. Our main contribution here is to introduce a novel technique that is taylored for DL models, does not require retraining or altering DL models and is very lightweight and effective. In a nutshell, our technique exploits gradient backpropagation to assess the amount of novelty in the feed-forward inference: the larger the gradient resulting from the first backpropagation step, the lesser the chance that the novel sample has been extensively seen during training, hence the larger the likelihood that the new sample comes from a zero-day application.

In this paper, we first cover the state of the art concerning known application identification $f(x)$ and unknown traffic detection $g(X)$ in Sec. II, and introduce our commercial grade dataset in Sec. III. We next detail the ML/DL techniques we use for known/unknown traffic identification in Sec. IV. We separately address the evaluation of known application identification in Sec. V and thoroughly evaluate detection of unknown application classes in Sec. VI. Finally, we summarize our findings and discuss their implications in Sec. VII.

II. Related work

A. Identification of known traffic classes $f(x)$

The first decade of 2000 witnesses the first wave of traffic classification, with methodologies aiming to identify fine-grained applications (e.g., YouTube, Skype, Whatsapp) or coarse-grained services (e.g., video streaming, video call, messaging). In this section we summarize the main lessons learned from this first phase, and focus our attention to the second wave where DL-based methods are introduced. We summarize the relevant literature for both periods into Tab. I

1) First ML wave: Numerous surveys cover the first-wave of traffic classification [7], [22]. In this era, "classic" ML techniques are studied, relying on engineered flow features (FF) as in [23], or packet payload (PP) as in [2], [5], and it culminates in the adoption of simple, yet effective, use of timeseries (TS) based on properties of the first packets of a flow such as packet size (S), direction (±), and seldom interarrival time (ΔT) [3], [4]. Lightweight TS approaches are particularly important since (i) they operate "early" at the beginning of a flow, as opposite to "post mortem" as for techniques based on FF that are computed after a flow ends, and (ii) they sustain line-rate operation with minimal additional computational complexity. Indeed, whereas payload-based techniques are inherently limited in the amount of memory they need to access even when processing is done on GPUs [24], early TS techniques [3], [4] have been amenable to line rate classification in excess of 5 Mfps [25] using only general purpose CPU.

2) Second DL wave: The second wave of research reconsidered all possible inputs exploited in the first wake — from PP [9], [11], [12], [18], to FF [10], [14], [15], to TS [13], [17], and hybrid FF+TS [14]— but has not been exempt from flaws and pitfalls experienced by the first wave methodologies.

For instance, we argue that owing to encryption, most work exploiting packet payload PP is essentially learning the content of the TLS Server Name Indication (SNI) extension [21], [26], i.e., binding a flow to the hostname advertised in clear in the SNI: ultimately, this means that 1d-CNN approaches[1] leaveraging PP are *a complex mean to do relatively trivial HTTPS protocol dissection and pattern matching.* In other words, while DL is an elegant and automated way to statistically learn SNI "dictionaries", the open question is whether CNN inference can be brought to an operational point with lower computational footprint than traditional techniques.

Similarly, some work present results biased by the inclusion of *port numbers* in their TS [13], [14]. For instance, authors in [13] show a stricking 93% for CNN operating on a single packet input: this is tied to the fact that in the academic dataset, HTTP, SSL and DNS (that are amenable to port 80, 443 and 53 respectively), account for over 80% flows, and suggests that DL architectures described in those works *ultimately exploit the very same port-based information that they are supposed to supplant.*

More generally, the second wave shares the same key weakness already noted in the first wave: the difficulty of

[1] Similarly happens for 2d-CNN, that additionally artificially construct "spatial" dependencies in the payload that are suitable for images of the physical world, but are less motivated for textual/binary protocols.

TABLE I: Related work $f(x)$: Supervised application identification

1st: ML	**Input**	**Architecture**	**Samples**	**Classes**	**Performance**	**Notes**
2004 [1]	2 FF (avg. D, avg S)	k-NN vs LDA	5.1M	4 (7)	95% (91%)	100% adding an extra feature related to ΔT
2005 [2]	PP	9 DPI heuristics	573M	10	99%	macro-classes and manual ground truth creation
2006 [3]	TS_5 (±S)	K-means	n.a.	10	≈90%	
2007 [4]	TS_3(±S, ΔT)	2d gaussian filter	30k	4	≈90%	multi-dimensional statistical fingerprint

2nd: DL	**Input**	**Architecture**	**Samples**	**Classes**	**Performance**	**Notes**
2015 [9]	PP (1000B)	1D CNN	300k	58	90% in top-25	Introduces DL to TC
2016 [10]	FF (40)	RF (non-DL)	131k	110	99%	Introduces APP classification
2017 [11]	PP (784B=28×28)	2D CNN	750k	20	99%	payload as image
2017 [12]	PP (784B)	1D CNN	750k	12	≈ 90%	payload as blob
2017 [13]	TS_{20} ($\pm S, \Delta T, p$)	LSTM + 2D-CNN	266k	15	96% (82% w/o p)	several models: (CNN+RNN-2a)
2017 [14]	TS_{10} ($\pm S, \Delta T, p$) + FF (28)	CNN	22k	5+(5 real)	99% (88%)	
2017 [15]	FF (22)	GAN (vs DT and RF)	682k	2	all 99%	SSH vs non-SSH
2018 [16]	PP, FF, TS	MLP, SAE LSTM, CNN	138k	49	80-86%	DL [9], [11]–[13], [18] vs RF [10]
2019 [17]	TS → 2D histo (1500^2)	LeNet-5 like	21k	10	99%	
2019 [21]	TS_6 ($\pm S$) + PP (256×6)	1D CNN + LSTM	unclear	80	95%	large scale dataset
2020 [18]	Payload	1D CNN and SAE	unclear	17	98%	dataset is public, but flows are not specified
This work	*TS*	*1D CNN*	*≈10M*	*200+635*	*91% top-200*	*private commercial-grade dataset*

***Input**: flow features (FF), packet payload (PP), flow duration (D); time series (TS) of packet size (S), direction (±), interarrival (ΔT) and ports (p).*
***Architecture**: Random Forest (RF); Multi-layer perceptron (MLP); Stacked Autoencoders (SAE), Convolutional neural networks (CNN); Long-short term memory (LSTM); Generative adversarial networks (GAN)*

cross comparing in a fair manner these different architectures. Indeed, as it clearly emerges from Tab. I, every work uses different datasets, with different sample size (from 20k to 750k samples), and with different target classes (from 2 to 100), achieving performance in excess of >99% (under specific conditions), making an apple-to-apple comparison difficult.

Fortunately, commendable work such as [16] started appearing, aiming to an independent evaluation of previously published work. The comparison carried out in [16] (of [10]–[13], and by mean of datasets different from the one used by original authors) reveals a different scenario from the one pictured by the original publications: (i), the expected performance, in practice, drops significantly below <90% for any architecture; (ii) there is no clear winner, although 1d-CNN have consistently better results among the candidate approaches; (iii) 1d-CNN has a limited gain over shallow Multi Layer Perceptron (MLP) over the same input (+6%) or Random Forest (RF) over FF input (+3%). We underline that such insights are possible only when broadening the evaluation scope beyond the typical race for 100% classification accuracy.

Despite its merits, [16] still partially falls into an apple-vs-orange comparison. For instance, the classic RF model inherited from [10] is based on engineered flow features (FF), whereas the CNN models are either based on packet payload (PP) [11], [12] or packet time series (TS) [13]. As such, is extremely difficult to attribute improvement to either the learning technique (i.e., ML vs DL) or the model input (e.g., FF vs TS). To counter this problem, sharing the same spirit of [16] *we perform an independent evaluation of two state of the art ML/DL techniques, applied to exactly the same input (TS), on a commercial grade dataset.* So doing, we reveal problems that only appear at scale, and that are often ignored in the academic community.

B. Detection of zero-day applications $g(x)$

Zero-day applications detection is a network-domain problem arising when either new applications appear or known application change their behavior. In this case, a zero-day enabled classifier is able to identify the "new" traffic as "unknown", avoid mislabeling it as one of the known classes. Zero-day detection is also known as *open-set recognition* in the knowledge discovery domain: in this work, we contribute a new open-set recognition method (Sec.IV-B) that is lightweight and accurate compared against representative techniques from the state of the art (Sec VI).

We summarize the relevant literature in Tab. II dividing it into three categories based on whether zero-day detection is performed on input x, output $\ell = f(x)$ or in the inner-stages of the model: note that techniques based on input/output data are decoupled from the design of $f(x)$, which is a desirable property as altering existing models would make deployment more difficult. Computational complexity is another important aspect: as $g(x)$ is meant to be applied on each input, hence cannot be significantly slower than $f(x)$.

1) Output: The most common approach for zero-day detection on output data is thresholding SoftMax outputs [29]. OpenMax [28] revises SoftMax activation vectors adding a special "synthetic" unknown class (by using weigthing induced by Weibull modeling). Alternative approaches include the use of Extreme Value Machine (EVM) [27], based of Extreme Value Theory (EVT), and, more recently, clustering on the CNN feature vectors with a PCA reduction of dimension [30].

All these approaches have the advantage of a limited complexity, and of not requiring modification of a pre-trained model: this makes them particularly relevant and worth considering for a direct performance comparison. Our proposed methods, based on evaluating gradients change via backpropagation, also fit in this class.

TABLE II: Related work $g(x)$: Zero-day application detection

	Year [ref]	**Technique**	**Applicability**	$f(x)$ **Modification**	**Complexity**
Output	2015 [27]	Extreme Value Machine (FV)	ML/DL	None	Medium (Weibull inference)
	2015 [28]	✓ OpenMax (AV)	ML/DL	None	Medium (Weibull inference)
	2017 [29]	✓ SoftMax	ML/DL	None	Low (threshold)
	2020 [30]	✓ Clustering (FV)	DL	None	High (clustering)
	This work	*Gradient backpropagation*	*DL*	*None*	*Low (backpropagation on last layer)*
Inner	2017 [31]	GAN-OpenMax	ML/DL	Yes	Medium(weibull inference)
	2018 [32]	Classifier K+1;	ML/DL	Yes	Low(threshold)
	2018 [33]–[36]	Clustering loss Functions	ML/DL	Yes	Low(threshold)
	2018 [37]	Confidence learning	ML/DL	Yes	Low(threshold)
	2019 [38]	CNN + AE	ML/DL	None	High(AE inference)
	2019 [39]	AE	ML/DL	None	High(AE inference)
	2020 [40]	Sigmoid activation	ML/DL	Yes	Medium(weibull inference)
Input	2007 [41]	Input clustering	ML	None	Not assessed
	2015 [42]	✓ Input clustering	ML	None	High (hundreds of clusters for small K)
	2017 [43]	Input modification wrt. Temperature scaling	DL	None	High (full backpropagation + inference)
	2018 [44]	Input modification wrt. Mahalanobis	DL	None	High (full backpropagation + inference)

Technique*: Feature Vector (FV); Activation Vector (AV); Autoencoders (AE), Convolutional neural networks (CNN); Generative adversarial networks (GAN)*
✓: Technique we compare against in this work

2) Inner: At their core, DL methods project input data into a *latent space* where is easier to separate data based on class labels. A set of work then proposes specific ways to alter this latent space to purposely simplify open-set recognition.

For instance, [39] uses AutoEncoders (AE) to transform input data, and apply clustering to the transformed input, while [38] uses latent representation along with OpenMax [28] activation vectors. Other works instead rely on Generative Adversarial Networks (GAN) to explore the latent space in order to generate "unknown classes" data to train a classifier for the class $\ell = 0$ i.e., a K+1 classifier. For instance, [31] generate unknown classes by mixing the latent representation of known classes, while [32] uses optimisation methods to create counterfactual samples that are close to training samples but do not belong to training data. All these methods require specific architectures (so they are hardly deployable) and extra training (so their computational complexity can be high).

Other work propose to alter activation [40] or loss functions [33]–[36]. In [40] authors replace the SoftMax activation with a sigmoid, and fit a Weibull distribution for each activation output to revise the output activation. Special clustering loss functions [33]–[36] can be used to further constraint points of the same class to be close to each other, so that unknown classes are expected to be projected into sparse region which is far from known classes. All these methods constrain to use special DL architectures and cannot be used on existing models; additionally such architectural modifications can alter the accuracy of the supervised classification task, for which we deem this category unapt for the zero-day detection task.

3) Input: A last class of work works directly in the input space. Such class includes both the seminal work for zero-day detection in the traffic classification context [41] as well as the current state of the art [42] where authors integrate into a classifier a zero-day detection module aiming to continuously update traffic classes knowledge. Shorlty, [41], [42] use K-means to cluster input data, and identify unknown applications by thresholding the distance of an input sample from the clusters' centroid. These approaches can be considered as the state of the art in the network domain, and are thus worth considering despite their complexity as a reference benchmark.

A key challenge when using input-clustering is the creation of clusters fitting well the different classes. To better control this, some works have recently proposed to apply transformations on the input so control models output by mean of Mahalanobis based score [44] or temperature scaled SoftMax score [43]. The drawback of those proposals is their additional computational cost beyond [41], [42], which makes these last approaches of little appeal from practical perspective.

III. Dataset

In this work we use a dataset collected from 4 Huawei's customer deployments in China. Each vantage point records per-flow logs where each entry relates to a flow 5-tuple (anonymized ipSrc, ipDst, portSrc, portDst, l4Proto) with aggregate metrics (bytes, pkts, avg rtt, etc.) and per-packet info (size, direction, inter packet gap of the first 100 packets). Each log entry is also annotated with application labels provided by a Huawei commercial-grade DPI engine. We use the per-packet statistics to create the TS and the provided application label to train out ML/DL models. Overall, the dataset[2] correspond to traffic activity across four weeks by tens of thousands network devices.

1) Collection environments: The dataset is collected via two Huawei's product lines that offer network monitoring and services solutions for the *Enterprise campus* and *Customer OLT/ONT* market segments in China. We underline that traffic encryption in China is not as pervasive as in the Western world yet. Moreover, as commonplace in the Enterprise market, branches employ HTTPS proxies, so that DPI can work unperturbed. This explains the availability of a very large

[2] The dataset does not provide any sensible information about end-users, and no raw pcap collections have been performed.

TABLE III: Commercial-grade dataset description

Scope[1]	Classes	%	Flows	%	Byte	%
	10	0.3%	3.4M	32.6%	5.6 TB	53.0%
	20	0.6%	4.6M	43.6%	7.2 TB	67.1%
$f(x)$	$K'=50$	1.5%	7.2M	68.3%	8.8 TB	82.7%
	100	3.1%	8.7M	82.9%	9.8 TB	91.9%
	$K''=200$	6.2%	9.9M	94.0%	10.3 TB	97.1%
$g(x)$	250	7.7%	10.2M	95.5%	10.4 TB	98.2%
	835	25.9%	10.4M	99.3%	10.6 TB	99.8%
noise	1,000	30.9%	10.5M	99.5%	10.6 TB	99.9%
	3,231	100%	10.5M	100%	10.6 TB	100%

[1] Denotes the portion of the dataset which is relevant for the supervised traffic identification $f(x)$ vs unsupervised zero-day detection $g(x)$.

TABLE IV: Breakdown of most popular (K) vs Zero-day TCP/UDP (U) applications.

Proto	Known	Zero-day	Tot	Perc	Openness
	apps K	apps U	$U+K$	$\frac{U}{U+K}$	$1-\sqrt{\frac{2K}{2K+U}}$
TCP	162	500	662	75.5%	37.3%
UDP	38	135	173	78.0%	40.0%
Total	200	635	835	76.0%	37.8%

number of label: the dataset comprises *3231 application labels that is* 30× *the largest number of classes considered in the literature* [10]. The dataset is a private Huawei asset and cannot unfortunately be shared[3] – as often remarked in the literature, the lack of common dataset is one of the major limit of this field (see discussion in Sec.VII).

2) Traffic imbalance: The dataset offers the rather typical class imbalance as illustrated in Tab.III. The typical number of classes in academic literature K=10–50, only convers a tiny portion of the applications catalog (0.3–1.5%), but capture a sizeable portion of the traffic flows (32.6–68.3%) and bytewise volume (53.0–82.7%). Yet, even when using $K' = 50$, about 1/3 of flows (1/5 of bytes) are not covered: a commercial product needs to target about $K'' = 200$ to cover roughly 95% of traffic (specifically, 94.0% of flows and 97.1% of bytes). In other words, the common scenarios studied in literature is faraway from business needs. Despite academic models work well (generally, in excess of 99%) for K=10–50 classes, it is not obvious to project results from literature on a given architecture when confronted to hundreds more classes – this is at the core of our investigation in Sec.V.

3) Dataset scope: As suggested by the column *Scope* in Tab.III, in the remainder we are going to focus on the top-200 applications when discussing known application classification $f(x)$. We then extend the scope to include all top-835 applications with at least 100 samples for zero-day application detection $g(x)$. To better highlight this, Tab.IV further details the split between known (K) and unknown (U) applications for the TCP and UDP protocols. The table additionally reports the percentage of unknown classes $U/(U+K)$ and the openness $1-\sqrt{2K/(2K+U)}$, which is a metric traditionally used to assess the difficulty of the open-set recognition task.

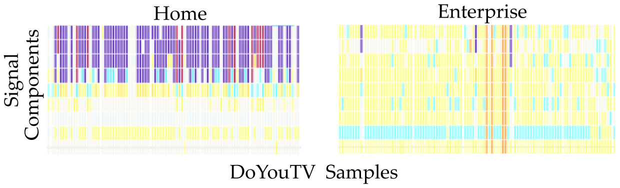


Fig. 2: Example of multi-modal behavior for the same application across two environments.

The top-200 applications (162 TCP and 38 UDP) falls is the scope of $f(x)$, covering 94% of the flow and 97% of the bytes. Beyond that point, increasing the number of classes only minimally affects coverage (using K=250 improves the dataset coverage by only 1%). We introduce DL and ML models for known application identification in Sec.IV-A, and evaluate them in Sec.V.

A second portion of 635 applications (500 TCP and 135 UDP) increases the overall coverage to 99.8% of the bytes. We remark that the number of flows per class is small to properly train and validate a supervised model $f(x)$: while the top-200 classes have on average 50,000 labeled flows, these additional 635 classes have only about 750 flows on average. Conversely, this set of applications is well suited to assess zero-day detection $g(x)$. We stress that this small fraction of roughly 5% (2%) of flows (bytes) represents the wide majority (76%) of the overall labels. We introduce the zero-day discovery techniques in Sec.IV-B, and evaluate them in Sec.VI.

Finally, a long tail of applications (75% of the classes, with about 13% of classes having only one sample) accounts for a tiny fraction of the flows and bytes (about 0.1%). We consider those applications as "noise", and we discard them from the analysis given their limited statistical and practical relevance.

4) Multi-modality induced by the environment: An interesting phenomenon we observe in our dataset relates to traffic "modes", i.e., an application's TS change due to the network environment. We showcase this for the popular application *DoYouTv* in Fig.2. The heatmaps depict the top-100 TS across all flows observed from a residential (left) and enterprise campus (right) vantage point. Each column represents one TS, with rows visually encoding the TS values (10 packet sizes) by mean of a color scale. Although not explicitly pointed out in previous literature, multi-modality is an intrinsic effect of access type, encapsulation, etc. as well as firewalls, NATs, etc., configurations which may alter time-series properties (packet sizes in this case). This phenomenon has practical relevance since implies that what learned from one network does not necessarily generalize to all networks, hence it needs to be

[3] We are currently investigating the possibility to release a highly anonymized (e.g., shuffled and normalized timeseries, etc.) and semantically deprived (e.g., no texual labels) dataset to the community. This is however beyond the scope of the authors decision, and due to lengthy processes involving legal aspects should be decoupled from the publication process.

dealt with at training time by purposely including samples from all vantage points, or by distributed learning [45].

IV. Methodology

A. Known application identification $f(x)$

In this section we introduce our models based on packets data TS. The rationale of our selection it twice fold. First, we seek to avoid providing yet-another-solution better than the previous literature just for the sake of "arm race". Second, we are interested in models lower-bound performance as to provide a conservative assessment and avoid to reveal sensitive product information. As we shall see, our choices are not limiting us from providing key observations.

1) Output: We focus on fine-grained traffic classification, with models targeting the identification of K classes. Recalling Tab.I, the DL classifiers proposed in the literature consider K<50 classes. Works considering a larger set of applications either reports the accuracy of the top classes (top-25 [9]) or the dataset used is practically limited to fewer classes (e.g., the top-15 classes represent over 99% of the traffic in [13]). Only a few works study K=50 classes [10], [16]. Conversely, Tab.III shows that we need to consider K=200 classes to cover 95% of flows (97% of bytes) in our dataset. We thus consider K'=50 and K''=200 so to both compare against academic state-of-the art [16], as well as cover business needs. We leave as future work an in-depth evaluation of how to identify all 3231 classes available in the dataset.

2) Input: Models input are TS of the first 10 UDP (100 TCP) packets size and direction, which values are $\pm S \in \mathbb{Z}$ rescaled into $[0,1]{\in}\mathbb{R}$ by normalizing the packet sizes over the maximum MTU. Generally speaking, this input is appealing given its ease to collect, and has been consistently found to yield excellent performance across both ML [3], [4] and DL [13], [14], [16].

Notice that this simple input could be complemented by further TS information related to packets interarrival, TCP header flags, TLS SNI strings, etc.. For instance, the first wave of literature found the inter packets time ΔT to be a valid input [3], [4]. Yet, this can be prone to errors, as ΔT can in practice represent the time between a packet sent and its response (the Round Trip Time – RTT), which correlates more with the distance between the endpoints rather than capturing the application behavior.

Overall, we opt for simple one-dimensional $\pm S$ timeseries as to focus on models lower bound performance (which is desirable from a scientific standpoint), and avoid using sensitive/privacy-related info such as TSL SNI strings (which is desirable from a business standpoint). Differently from previous literature [16], we use the same input for both ML and DL models to have an apple-to-apple comparison between modeling techniques.

3) DL model (1d-CNN): As representative DL model, we use a 1d-CNN given that (i) is an architecture well fit for timeseries input, (ii) it provides superior performance compared to 2d-CNN [11], [12], and (iii) its related literature is also independently evaluated [16]. We stress that we rule out recursive networks (and LSTM in particular) in reason of their additional computational complexity.

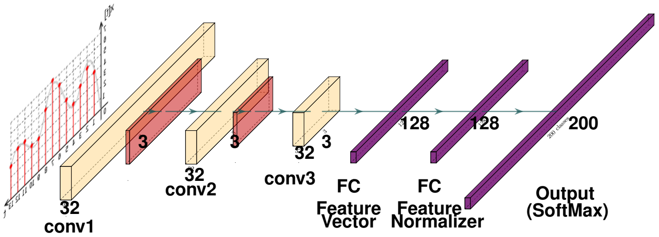


Fig. 3: Baseline CNN architecture for known application identification $f(x)$ (hyperparameters for TCP traffic portion).

To address the different classes, we use one model for TCP and one for UDP. As per Tab.IV, the top-200 classes include 162 TCP and 38 UDP applications. The architecture of both models follow the most common design choices and is sketched in Fig.3. Considering UDP (TCP) traffic, the input layer feeds a stack of convolutional filters having depth 3, and 16 (32) filters of 1×3 size, with ReLU activation and max-pooling layers, followed by one fully connected layer of size 64 (128) for a total depth of 4 layers before the final SoftMax classification of size 38 (162). While detailed hyperparameters selection differs across each work, this architecture is rather typical and is exploited (with minor differences) by [11]–[14], [17]. Notice that in the architecture, one fully connected layer is explicitly indicated as *Feature Vector*: as we shall detail in Sec.IV-B, this layer plays an important role for novelty discovery in related literature.

To ease comparison, we focus only on CNN model *space complexity* W_{DL} defined as the overall number of model parameters. We intentionally avoid *time complexity* measurements (e.g., training duration, or inference latency) as they directly relate to implementation details and GPU/TPU hardware acceleration [19]. This still allows a fair comparison against state of the art, since only space complexity is typically discussed in (or can be easily derived from) the literature.

4) ML model (XGBoost): As representative ML model, we use extreme gradient boosting (XGBoost), a tree-based ensemble technique widely acknowledged as a ML state of the art classifier in numerous applications domains [46], [47]. We define the model *space complexity* W_{ML} as total number of nodes in the trained ensemble. As a mean to control W_{ML}, we fix the number of trees T=100 and cap the individual tree depth $d \in \{2,4,7\}$.

5) Models comparison: Both ML and the DL models are trained using the same dataset folds. This allows a punctual comparison of classification accuracy and complexity, which we carry on in Sec.V. Clearly, W_{ML} and W_{DL} space complexity enable only a qualitative comparison since the individual operations in the ML domain (e.g., memory comparison and branching) and DL domain (e.g., tensor products) intrinsically differ. At the same time, a precise comparison of computational complexity would also be tied to specific implementation and system aspects [19], [25], that we consider to be out of the scope of this work.

B. Zero-day application detection $g(x)$

To identify zero-day applications, the $g(x)$ function needs to reject the classification label for those input samples x belonging to classes never exposed to $f(x)$ during training (see Fig.1). In other words, $g(x)$ acts as a binary classifier: when $g(x)$=1 the sample is considered novel and the classification result is rejected by overriding the final class result with $\ell = f(x)(1 - g(x)) = 0$. We use known $g(x)$ from the literature based on input data, inner/hidden layers telemetry, or the models output. Additionally, we contribute a novel technique which is (i) superior to alternatives methodologies and (ii) both lightweight and easily deployable.

1) Input based clustering (ML and DL): Clustering is a common technique to identify unknown classes using input data [41], [42]. Clusters are formed using training samples of the known applications. Comparing the distance between an input x and the clusters centroid with a predefined threshold allows to accept/reject the $f(x)$ label. Specifically, the reject criterion is:

$$g^{IN}(x) = \mathbb{1}\left(\min_{c\in[1,C]} d(x,c) > \epsilon_{IN}\right) \tag{1}$$

where $d(x, c)$ is a distance metric (euclidean, manhattan, etc.), and ϵ_{IN} is an arbitrary threshold. Intuitively, a small ϵ_{IN} increases the chance to reject a correctly classified sample of a known application (false negative), whereas setting a too large threshold leads to unknown application going undetected (false positive). The reference state of the art technique is [42]. It adopts a large number of clusters $C \gg K$ so to fit the input space, in which case it shows that a simple K-means suffices to coarsely assess whether an input sample is close to (far from) the clusters of a known application. Clearly, the number of clusters C tradeoffs computational complexity (as a point needs to be compared against C centroids) and accuracy (as a high number of clusters C provides a finer grain coverage of the input space).

2) Feature-extractor based clustering (DL only): When using a DL model, clustering can be applied also to *Feature Vectors* (FVs), i.e., the output generated by one of the hidden layers when processing an input sample (see Fig.3). Essentially, the training process of DL models performs non-linear transformations that improve its discriminate power. The output of the feature vectors layer is a *high-dimensional latent space* where the projected input samples are more easily separable with respect to their class. It is possible to induce clustering in the latent space by either using a clustering loss function [33]–[36], or by constraining the space (e.g., by normalizing the layer into a hyper-sphere). As previously, the C clusters are constructed at training time applying K-means to $FV(x)$. The rejection criterion is:

$$g^{FV}(x) = \mathbb{1}\Big(\min_{c\in[1,C]} d(FV(x), c) > \epsilon_{FV}\Big) \tag{2}$$

where, as previously, $d(\cdot)$ and ϵ_{FV} are a distance metric and an arbitrary threshold respectively. This technique is standard in the DL domain, and can be considered as a state of the art for DL-based clustering [30].

3) Output based rejection (ML and DL): Output-based techniques leverage additional information from the ML/DL model output, such as the SoftMax [29] or OpenMax [28] probabilities of each class. Denoting with $v(x)$ the activation vectors of an input sample x, and with $v_i(x)$ the i-th component of the vector, then the SoftMax value for class k given input x is:

$$P(y = k|x) = \frac{e^{v_k(x)}}{\sum_{i=1}^{K} e^{v_i(x)}} \tag{3}$$

Denoting further with $c = \arg\max_k P(y = k|x)$ the most likely class selected by the supervised model, then the SoftMax rejection criterion is:

$$g^{SM}(x) = \mathbb{1}\Big(P(y = c|x) < \epsilon_{SM}\Big) \tag{4}$$

that is, the SoftMax of the output class $c = f(x)$ is required to be larger than an arbitrary threshold ϵ_{SM}. This technique should be considered a naïve baseline, since it is well known that supervised models can be *overconfident*, i.e., the SoftMax value can be high despite the classification is wrong, which stems from the saturating nature of the activation function [48].

Part of the problem is rooted in the fact that the SoftMax function normalises only considering the space of known applications: to solve this issue, OpenMax [28] introduces an extra "unknown" label and re-normalizes the activation vector before computing distances. More in details, OpenMax introduces a weight vector ω_b that captures the distribution of the activation vectors $v(x)$ of the K known classes by fitting a Weibull distribution. The vector ω includes also a extra "synthetic" activation value, namely the *novel class* $\ell = 0$. A new activation vector $\hat{v}(x)$ is then derived as follows:

$$\hat{v}(x) = v(x) \circ w_b(x) \tag{5}$$

$$\hat{v}_0(x) = \sum_{k=1}^{K} v_k(x)(1 - w_b^k(x)) \tag{6}$$

The OpenMax probabilities are then derived with the renormalized activations vector, including for the novel class $c = 0$

$$P'(y = k|x) = \frac{e^{\hat{v}_k(x)}}{\sum_{i=0}^{K} e^{\hat{v}_i(x)}} \tag{7}$$

and denoting with $c' = \arg\max_k P'(y = k|x)$ the class with the largest OpenMax values, the rejection criterion then becomes:

$$g^{OM}(x) = \mathbb{1}\Big(c' = 0 \vee P'(y = c'|x) < \epsilon_{OM}\Big) \tag{8}$$

where the supervised classification is rejected either when the novel class $c = 0$ has the largest value, or when the renomalized OpenMax value for the most likely class $c \in [1, K]$ is smaller than a threshold ϵ_{OM}.

4) Gradient based rejection (DL only): Finally, we propose a novel method for zero-day application detection using *back-propagation gradients as a proxy for novelty*. The original idea is to pretend the inference as correct and perform a shadow training step: (i) evaluate the output label $c = f(x)$; (ii) treat the label c as the groundtruth and compute the magnitude of the first backpropagation step δ^{L-1}, but in a "shadow" mode, i.e., without actually altering the CNN model; (iii) use δ^{L-1} to assess if the input is of a known or unknown application. More precisely, given

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{9}$$

where $\nabla_a C$ is the partial derivative of the model cost function C with respect to activation (i.e., the magnitude of the update) and $\sigma'(z^L)$ is the activation vector $v(x)$. To limit computation complexity, we limit backpropagation to the last layer $W^{(}L-1)$, which contains most information concerning the classes.

$$\delta^{L-1} = (W^{(L-1)^T}\delta^L) \odot \sigma'(z^{L-1}) \tag{10}$$

where $W^{(L-1)^T}$ is the transpose of the weight matrix of $(L-1)$-th layer. Intuitively, the larger δ^{L-1}, the more likely the input relates to an unknown class. Based on this intuition, we conceive a simple family of gradient-based rejection criteria:

$$g_n^{GR}(x) = \mathbb{1}\left(\|\delta^{L-1}\|_n > \epsilon_{GR}\right) \tag{11}$$

where the norm $\|\cdot\|_n$ and ϵ_{GR} are free hyperparametes. In particular, for L1 (i.e., max gradient) and L2 norms (i.e., the square root of squared gradient sum) we have:

$$g_1^{GR}(x) = \mathbb{1}\left(\mathrm{max}_i\delta_i > \epsilon_{GR}\right) \tag{12}$$

$$g_2^{GR}(x) = \mathbb{1}\left(\sqrt{\sum_{i=1}^{K}\delta_i^2} > \epsilon_{GR}\right) \tag{13}$$

Clearly, backpropagation is an essential tool in CNN, and gradients have been used in many aspects of DL, from training (e.g., to speed up convergence [49] possibly in distributed settings [45]) to extracting side-channel information (e.g., to gather information about clients participating into a federated learning cohorte [50]). However, we are not aware of such use for zero-day applications detection in particular, nor for openset recognition in general.

V. Identification of Known Applications

Models in literature are trained to identify $K' \leq 50$ classes, and evaluated using $X \in \{10, 50\}$ classes. In this section, we extend this approach towards a more business-drive scenario with $K'' \leq 200$ classes and $X \leq 200$. To address their diversity, we train and evaluate TCP (162 applications) and UDP classifiers (38 applications) separately, but we also report their combined performance (top-200 applications).

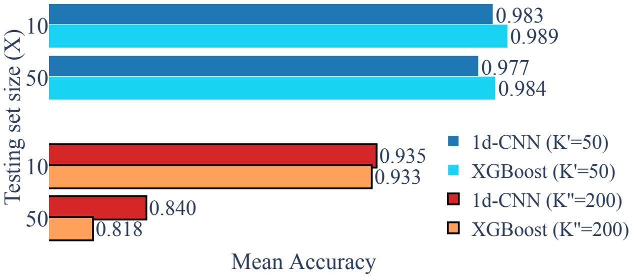


Fig. 4: *Class cardinality bias*: When training the same DL/ML model for a large number of classes $K'' \gg K'$ accuracy drops also for the top $X \leq K$ classes, which is unnoticed in academic literature.

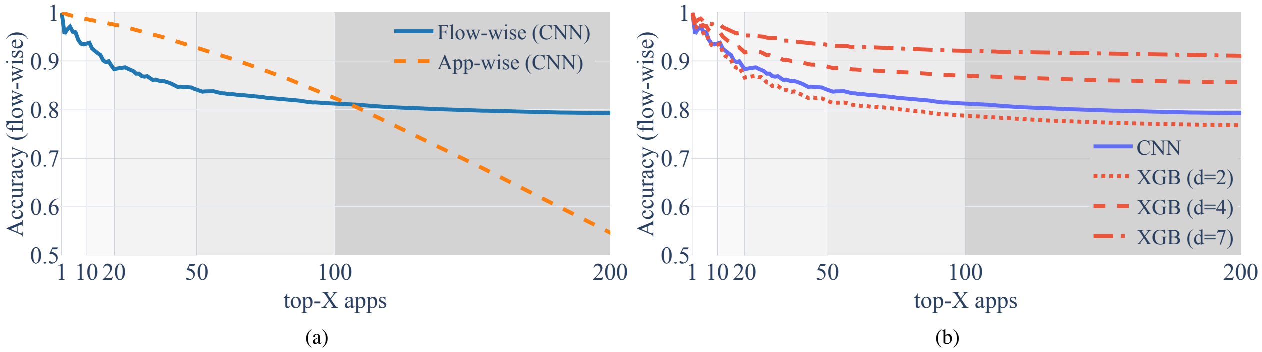
We start illustrating how analyzing only the top-K classes can bias the classification accuracy (Sec.V-A). Next, we dig into classification confounding factors and labels semantic (Sec.V-B). Finally, we discuss models space complexity (Sec.V-C).

A. High-level view

1) Class cardinality bias: Fig.4 shows the classification accuracy when evaluating $X \in \{10, 50\}$. For this analysis, we purposely trained the DL and ML models so that when $X = K' = 50$ they match flow-level accuracy reported in the literature for the top-$X = 50$ classes (≈99%). Results show that, when ML/DL models are trained to recognize at most $K' = 50$ classes, the classification accuracy of the top-10 and top-50 classes is stable. Conversely, when models are trained to recognize 4× more classes ($K'' = 200$), accuracy notably degrades for top-10 classes, and it further degrades for the top-50 classes. *In other words, focusing on a small number of classes makes the problem trivial to solve (even without DL).*

2) Mean accuracy bias: To further elaborate, Fig.5 shows the evolution of the mean accuracy for a model with $K'' = 200$ classes. We consider the top-X classes when ranked by their popularity (classes with larger number of flows first), and performance (best classified class first). The picture shows that, (i) despite classification accuracy worsen as X gets closer to K'' (dashed line), (ii) the overall number of correctly classified flows remains satisfactory in reason of the application popularity skew (solid line). Furthermore, this effect (iii) appears only for a number of classes larger than the one typically used in the academic literature. *Otherwise stated, focusing on the average accuracy of a limited number of classes $K' \ll K''$ hides phenomena typical in commercial scenarios.*

3) DL bias: We argue that the second wave of traffic classification published until this moment may have exaggerated the quality of DL methodologies, as already observed in computer science [51], and other fields [52]. We exemplify this in Fig.5 showing how the accuracy of an XGBoost model accuracy can be made (significantly) better than a CNN model by simply



(a)

(b)

Fig. 5: *(a) Mean accuracy bias.* Average per-flow (affected by class imbalance) or per-application accuracy (each application counted equally). Notice that the per-application accuracy overestimate performance in the small class regime $K < 100$, and vice-versa happens for $K > 100$. *(b) Model bias.* By tuning model hyperparameters, it is easy to obtain operational points that show superiority of one class of approaches (purposely biased to ML in this example).

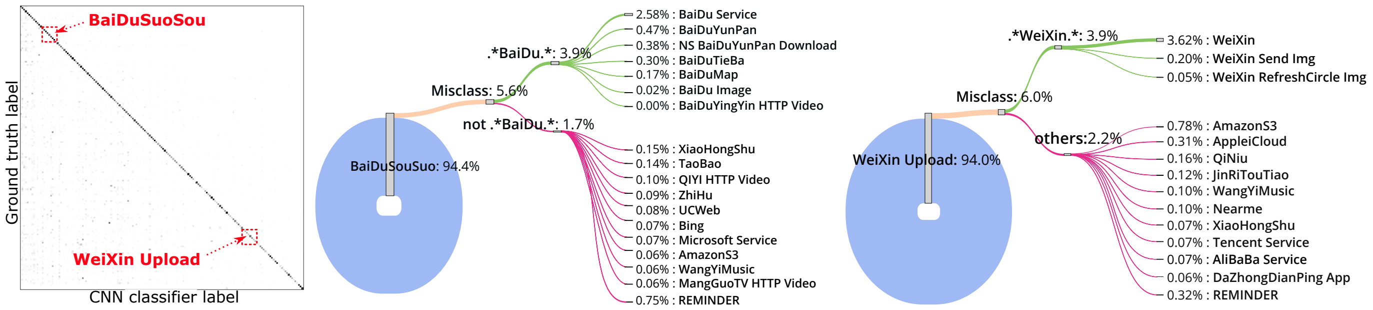


Fig. 6: Investigating classification performance for TCP traffic: (left) classification confusion matrix, with applications sorted by volume of bytes – performance are very good, but application with less traffic can suffer from lower accuracy; (center) BaiDuSouSuo classification breakdown – misclassification mostly relate to alternative labeling of traffic of the same application; (right) WeiXin Upload – a large portion of misclassifications are due to other behavior of the same application.

tuning the maximum tree depth in the example — *though we could easily have adopted the opposite viewpoint.*

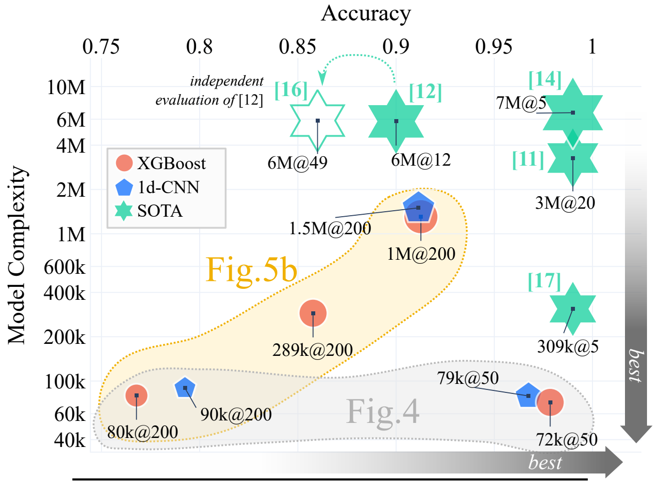
B. Deeper insights

We next dive into labels semantic to dissect misclassification causes. Again, we focus on the DL model with $K'' = 200$ classes. Fig.6 (left) shows the classification confusion matrix as a heatmap, with application labels sorted alphabetically (labels not shown): the sharp diagonal indicates that flows are classified with the expected label.

To further drill down, we pick a representative class for each side of the diagonal, *BaiDuSuoSou* (the BaiDu search service) and *WeiXin Upload* (the file upload service of the WeiXin messaging application), and we dissect their classification results by mean of a sankey diagram. For *BaiDuSuoSou*, Fig.6 (center) shows that 5.6% of flows are misclassified, but 3.9% are labeled as BaiDu-related services (e.g.,searches for image, maps, or social) and overall only 1.7% of flows are completely misclassified. The same considerations hold for *WeiXin Upload* where only 2.2% of the misclassification is imputed to non WeiXin related classes, and we find similar pattern among other popular applications (not shown). *In other words, a non negligible part of the misclassifications are for neighboring services of the same application provider.*

This phenomon is tied to the very fine-grained groundtruth labels on the one hand, and to the existence of different "modes" of the same application on the other hand. Intuitively, packet size at the input of the CNN reflect the application-level signaling at the beginning of a flow, which due to shared codebase and libraries, can be common across multiple type of flows of the same application provider. *This suggests that fine-grained applications identification is possible, with high accuracy, even for a larger sets of known application classes that those used in the literature.*

This holds particularly true for applications where sufficient samples are provided to the training process: notice indeed the degradation of flow-wise accuracy for less popular applications in the top-200, as early shown in Fig.5-(b). At the same time, as labels may be scarce for some applications, there is need for improving models accuracy in presence of classes imbalance beyond, e.g., the use of focal loss. This opens to experimentation of techniques such as few-shot learning [53], and ensemble-based methodologies [54] that we leave for future work and which may be of interest for the community.



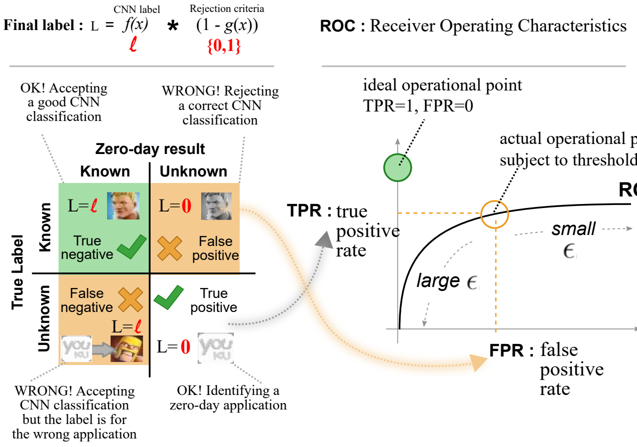
Method	$W[\times 1k]$	K	$W/K[\times 1k]$	Acc. [%]
[14]	6,640	5	1,328	99
[17]	309	5	61	99
[12]	5,800	12	483	90
[11]	3,270	20	163	99
[16]	5,800	49	119	86
1d-CNN	79	50	1.44	96
	90	200	0.450	79
	1,479	200	7.5	91
XGBoost	72	50	1.44	96
	80	200	0.4	76
	289	200	1.44	85
	1,307	200	5	91

Fig. 7: *Complexity*. Scatter plot of model space complexity W vs accuracy. Shapes are directly annotated ($W@K$) with the modelsize W and classes K, and the shape size is proportional to the weigths-per-class W/K value. The table details the scatter plot values.

C. Complexity

When assessing models accuracy is important not to underestimate the impact of their spatial complexity. As from Sec.IV-A, for DL models the space complexity is the number of model's parameters W_{DL}, while for ML models we use the number of nodes in the tree ensemble W_{ML}. We reiterate that space complexity does not directly translate into computational complexity (as this depends on the DL/ML architecture, the specific operations executed at inference time) nor energy expenditures (as this depends on the available hardware). Yet, it allows to abstract from specific implementation (i.e., software, hardware acceleration, system design choices) and enables a qualitative comparison between models. In particular, as models have disparate capabilities, we compare not only the absolute models size W, but especially the model size normalized over the number of output classes W/K.

The scatter plot in Fig.7 illustrates the accuracy vs complexity tradeoff for both our and literature models. We underline that the accuracy (complexity) comparison should be interpreted quantitatively (qualitatively). Results show models in the literature to be quite heavyweight, using up to $W/K = 6.6M/15 = 1.3$M and generally $W/K \gg 100$k weights-per-class, with the most parsimonious approach employing about 61k weights-per-class [17]. In contrast, the CNN (XGBoost) models we considered can achieve about 90% accuracy on the top-200 with just 7k (5k) weights-per-class, while still covering a 4× bigger number of classes.



Final label : L = f(x) * (1 - g(x))
CNN label
Rejection criteria
ℓ
{0,1}
ROC : Receiver Operating Characteristics
OK! Accepting a good CNN classification
WRONG! Rejecting a correct CNN classification
Zero-day result
Known
Unknown
True Label
Known
Unknown
L=ℓ
True negative
L=0
False positive
False negative
L=ℓ
True positive
L=0
WRONG! Accepting CNN classification but the label is for the wrong application
OK! Identifying a zero-day application
ideal operational point TPR=1, FPR=0
actual operational point subject to threshold ϵ
ROC
TPR : true positive rate
small ϵ
large ϵ
FPR : false positive rate

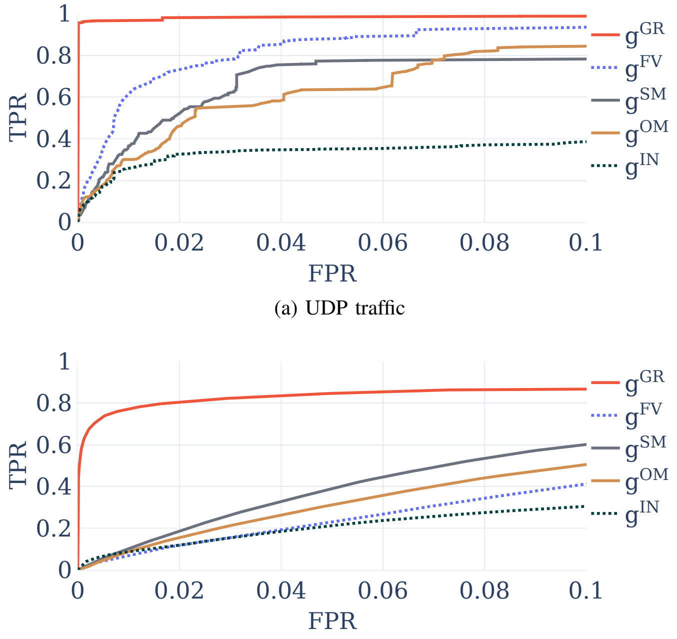
Fig. 8: *Quantifying zero-day application detection*: confusion matrix based on zero-day rejection (left); ROC curve for zero-day application detection (right).

Two other observations emerge from the picture. On the one hand, when the number of classes is large ($K{\approx}200$), it becomes necessary to increase models size to maintain accuracy performance (recall Fig.5), but CNN and XGBoost can be tuned to achieve similar performance. On the other hand, when the number of classes is small ($K{\approx}50$), it is unreasonable to use an humongous number of weights to discriminate them (see Fig.4), particularly since it is possible to design parsimonious models with just hundreds of weights-per-class achieving same-or-better performance (low right corner of the scatter plot). *As such, by neglecting model complexity, the risk is to propose solutions that are classic equivalent of shooting a mosquito with a cannon, which may hinder deployability.*

VI. Detection of Unknown Applications

In this section we evaluate zero-day applications detection. We built upon the CNN models evaluated in the previous section by applying the five zero-day application detection methods introduced in Sec.IV-B: input clustering, feature vectors clustering, SoftMax, Openmax, and our novel Gradient-based method.

Fig.8 sketches our evaluation methodology. Given a trained model, we perform the classification of both known and unknown classes, and construct a Receiver Operating Characteristic (ROC) curves from True Positive Rate (TPR)—how good is the model in correctly identifying the 635 zero-day applications—and False Positive Rate (FPR)—how often the model rejects a known top-200 application relabeling it as zero-day application. The ideal operational point is for (TPR=1, FPR=0), but in practice one needs to tradeoff the two metrics subject to the methods performance and tuning. In our scenario, setting a small ϵ allows to identify all unknown classes (high TPR) but some of the known classes can be



(a) UDP traffic

(b) TCP traffic

Fig. 9: *Zero-day applications detection*: ROC curves for UDP and TCP traffic.

wrongly identifies as zero-day (high FPR). As illustrated in Fig.8, the reverse is true when using a large ϵ. While the ROC curve is useful for comparing algorithms over a large set of parameter values, from a practical viewpoint is more useful to fix a maximum tolerable amount of FPR (i.e., correct $f(x)$ classification rejected due to wrong $g(x)$ choice). Since the majority of the traffic is represented by the top-200 applications, rejecting no more than 1% classifications is a desirable target: fixing FPR=1%, algorithms then can be compared according to the TPR amount of zero-day flow (or apps) that are correctly detected.

In the following, we first overview zero-day detection across the methods (Sec.VI-A), then we elaborate on the reasons behind their performance differences (Sec.VI-B), and we conclude evaluating their computational complexity (Sec.VI-C).

A. High-level view

Recalling Tab.IV, the TCP model can identify 162 known classes, and zero-day detection is tested against 162+500 known+unknown classes; the UDP model can identify 38 known classes, and zero-day detection is tested against 38+135 known+unknown classes. Notice that input and feature vectors clustering require a specific training step to apply K-mean: we use C=1000 clusters (which is 5× the number of known applications, to account for multi-modal signals) and gather the cluster centroids from the same training data used to train the supervised model $f(x)$. The ROC curves depicting zero-day detection capabilities for all methods are reported in Fig.9 (limited to FPR$<$10% which already exceeds reasonable operational point from an operational perspective).

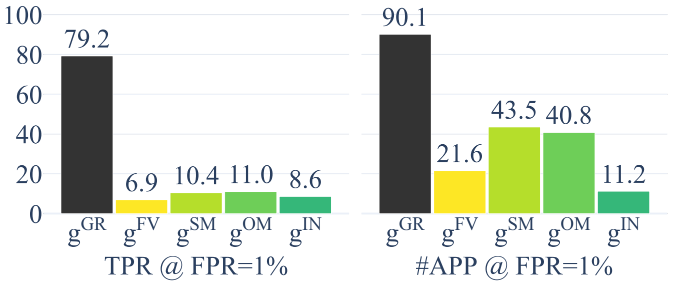


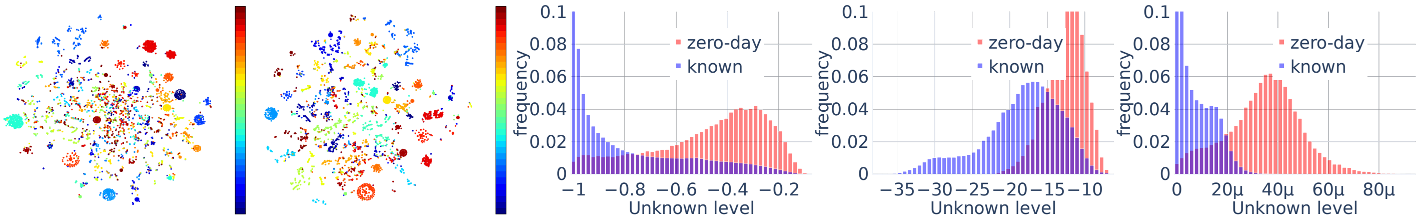
Fig. 10: *Zero-day applications detection*: Fraction TPR of correctly detected zero-day flows (left) and applications (right) when FPR=1% of correctly identified known applications are rejected. An application is considered to be detected when at least one flow for that application is correctly detected as zero-day.

It is immediate to gather that our proposed Gradient-based method (g^{GR}) outclasses all alternatives by a large margin: performance are almost perfect for UDP (TPR=96%, FPR=0.1%) and remain good in the more challenging TCP scenario (TPR=78%, FPR=1%) especially compared to alternative approaches.

While we expected OpenMax to be superior with respect to SoftMax [28], we find the opposite holds in our dataset: with the recommended hyper-parametrization [28] by setting the tail size to 20 when fitting Weibull distribution, SoftMax outperforms (or marginally differ from) OpenMax for TCP (or UDP). Clustering the input data (g^{IN}) is the worst option. The performance for clustering the feature vectors (g^{FV}) instead vary depending on the scenario: for UDP, it is the second best option; for TCP, it does not appear to be better than alternatives. For both scenarios we used M=1000 clusters: while increasing the number of clusters may provide benefits [42], however as we shall see the current setting already has a prohibitive computational cost, which make increasing the number of clusters hardly viable.

We confirm the results also when considering the top-200 applications by combining the TCP and UDP results, focusing on the target FPR=1% and assessing the fraction of correctly detected zero-day flow and applications in Fig.10 (where a zero-day application is considered detected if at least one of its flows is detected). At 1% FPR, it can be seen that while our gradient based approach correctly detects 79.2% of all zero-day flows (90.1% of the apps) the other approaches are only able to detect at most 11% zero-day flows (43.5% of apps).

Additionally, Fig.9 and Fig.10 also suggest that tolerating a higher FPR would only have a marginal improvement for the other approaches and that our Gradient-base method is less sensible to threshold tuning. In fact, while for all methods the TPR varies depending on the FPR, in Gradient-based ROC curves the TPR increases very quickly at small FPR, and grows slowly afterward: i.e., after the curve “knee”, it is almost insensitive to the configured threshold.



(a) Input space
Purity: 0.91
Silhouette: 0.71

(b) Feature Vector space
Purity: 0.97
Silhouette: 0.72

(c) SoftMax
Intersection: 0.39
Bhattacharyya: 0.36

(d) OpenMax
Intersection: 0.46
Bhattacharyya: 0.31

(e) Gradient
Intersection: 0.21
Bhattacharyya: 0.87

Fig. 11: Deeper insight on zero-day application detection: Qualitative visual differences of (a) Input-clustering and (b) FV-clustering via tSNE projection (UDP application only), and histograms of unknown level for (c) SoftMax, (d) OpenMax and (e) Gradient-based methods. Relevant cluster (purity, silhouette) and distribution distance metrics (Intersection, Bhattacharyya distance) are used to quantitatively support the visual comparison.

B. *Deeper insights*

Results discussed so far clearly show differences between the methodologies, but the underling reasons causing the performance gaps are not obvious: we therefore dig into the specific details of each method, starting by clustering methods. Fig.9 shows that input-clustering is the worse performing methodology. This is somehow expected, since the non linear transformations operated by the CNN feature extraction aim to empower better separation in the feature vectors space than the input space. We can however visualize this effect by mean of a bidimensional t-SNE [55]. To ease readability, Fig.11-(a,b) show the t-SNE only for the 38 UDP known applications, each associated to a different colors. Both plots show sharp clusters, but notice how the center of the input space (Fig.11-a) is more confused than the feature vectors one (Fig.11-b). We can further quantify this effect using *clusters purity* (the average fraction of dominant class in each cluster) and *clusters silhouette* (how similar an app is to its own cluster—-cohesion—compared to other clusters—separation). We find the FV space having the highest purity (0.97-vs-0.91), while both spaces have similar silhouette (0.71-vs-0.72).

To investigate the three remaining methods we measure the *unknown level* that quantifies predictions uncertainty: For SoftMax, based on (3) the unknown level is the negative of the maximum probability $UL^{SM} = -max_{k\in[1,K]}P(y = k|x)$. For OpenMax, based on (6) the unknown level is the *novel class* $\ell = 0$ value $UL^{OM} = \hat{v}_0(x)$. For our Gradient-based method, based on (9), the unknown level is the magnitude of the backpropagation update $UL^{GR} = \|\delta^L\|_n$. Visually, it is helpful to qualitatively examine the distribution of unknown level for known and unknown classes. To further precisely quantify and compare the overlap for the last three methods, it is convenient to use standard metrics for comparing distributions: in particular, we use the *Intersection index* $I = \sum \min(UL_{\text{known}}, UL_{zero-day})$ and the *Bhattacharyya distance* $D = -ln \sum \sqrt{(UL_{\text{known}} UL_{zero-day})}$

A desirable method should be able to separate known and zero-day applications unknown level distributions, which are shown in Fig.11-(c,d,e), or equivalently have low intersection I and high distance D, which are tabulated below the respective figures. Intuitively, known applications should provide small unknown level. For instance, for SoftMax the unknown level is capped at -1 corresponding to strong confidence in the model prediction. Conversely, when processing traffic of a zero-day application, the confidence is expected to reduce. The same is true for OpenMax and Gradient-based, although the unknown levels are unbounded. We can see that distributions overlaps for OpenMax, which has higher intersection I and lower Bhattacharyya distance D than the other methods, which explains the lower performance. This large confusion reflects the activation vector pattern might have a large diversity for the known classes, hence a simple mean activation vector (3) per class is insufficient. Furhermore, we observe that intersection I of gradient based method is about half of SoftMax, and that similarly distance D is over twice bigger, explaining the better performance earlier observed.

TABLE V: Zero-day application detection: $g(x)$ training and boostrap complexity

Method	g(x)	Bootstrap cost (per dataset)	Inference cost (per sample)
Gradient	$g^{GR}_{u,2}$	-	0.01 ms
	$g^{GR}_{u,1}$	-	0.01 ms
SoftMax	g^{SM}	-	0.01 ms
OpenMax	g^{OM}	11s	0.13 ms
FV-clustering	g^{FV}	24m 36s	4.58 ms
Input-clustering	g^{IN}	20m 4s	4.46 ms

C. *Complexity*

We conclude discussing both bootstrap and inference complexity of the novelty discovery methods, which are reported in Tab.V. Considering bootstrap, SoftMax and Gradient-based do not need any specific process, whereas OpenMax introduces a small cost to fit the Weibull distribution, while clustering-based methods are the heaviest due to clustering (K-means with C=1000 clusters in our setup). Notice that bootstrap cost are reported as a reference but, given the episodic and offline nature of the operation, they are less relevant from a practical perspective than the cost of the inference, as this latter needs to be performed for every flow.

It is thus more interesting to relatively compare the inference cost of all approaches: the trend remains similar, with OpenMax being $13\times$ slower than our method, and clustering-based being more than the two orders of magnitude slower than our method. Cluster-based approaches suffer from the computation of a large number (specifically C) of pair-wise distances (involving square and square root operations) in a fairly large space (specifically, the feature vector FV space of size 128). We point out that while distance computation can be reduced (eg., smaller FV space, int8 quantization, manhattan distance, etc.) however the performance of the clustering-based methods can be hardly expected to improve; conversely, increasing the number of clusters M might improve accuracy performance, but would nevertheless render the method even more complex and thus even less appealing.

The gradient based method therefore brings the best of both worlds, as it is not only significantly more accurate as early reported, but also as computationally complex as SoftMax: the reported precision in Tab.V do not even allow to appreciate inference cost differences, which are essentially a 200×200 Hadamard product in (9) and a matrix multiplication between a 128×200 and 200×1 matrices, followed by a 128×128 Hadamard product in (10). Additionally, while the complexity results are to be interpreted in a relative sense (i.e., across algorithm), we stress that the raw computational performance reported here comes from a non optimized implementation on tensorflow v1.9, is able to perform 100,000 $g(x)$ operations per-second – enough to sustain real system requirements [19].

VII. Summary and Discussion

In this paper, we tackle the issue of a commercial-grade traffic classification engine capable of (i) fine-grained application identification of several hundreds classes as well as of (ii) detecting zero-day applications that were not part of models knowledge base. We test the engine application identification capabilities on the top-200 applications (covering 95% of the flows and bytes) and the novelty discovery capability on top-835 applications (extending the flows and byte coverage to over 99%).

Summarizing our main findings, we gather that (i) ML and DL models are both equally able to provide satisfactory solution for classification of known traffic: in particular, the type of CNN architectures used in the literature are well suited, as their accuracy exceeds 90% for the top-200. At the same time, our results also point out that models complexity is commonly overlooked in literature: this yields to models unnecessarily complex for relatively simple tasks, and is endangering the practical relevance of the research.

More interestingly, we also gather that (ii) ML and DL differ in their ability to detect zero-day traffic, as the non-linear feature extraction process of the DL backbone yields to sizeable advantages over ML for this task. In particular, our main contribution is to provide a novel, simple yet surprisingly accurate technique for zero-day detection that exploits DL backpropagation computation.

While our work advance with respect to the state of the art, some open points are worth sharing and discussing with the research community

1) Open dataset: pooling efforts: In reason of our results, constructing an open corpus with rich *class diversity* and large *class cardinally* K should be a priority goal to allow for meaningful and fair cross-comparison of research proposals. While this is within reach for large industrial players, legal and business aspects prevent them to share openly their datasets – this is hardly a surprise, but unfortunately apply also to the datasets used in this paper.

While collecting large volumes of real labeled network data is a daunting effort for a *single academic* partner, pooling effort across *multiple research groups* N in a coordinated manner can be an effective strategy to achieve this goal – for instance, each partner can gather K/N classes, coordinating to keep a null/low class overlap between groups $\mathcal{K}_i \cap \mathcal{K}_j = \emptyset$. Also, different research groups are already doing active measurement collections for specific application types (video, games, etc.) and with a different goal than traffic classification (congestion control, QoE, etc.), so that the true burden lies in the coordination. Yet, this is commonplace in other communities (e.g., ImageNet has 15 million images labeled in 20k classes), and the traffic classification community should take inspiration from those efforts. This would allow to tackle more challenging classification problems, with a more significant application diversity, a set of fine-grained labels and a comprehensive coverage.

2) Deployability: the elephant in the room: Given that traffic classification is a mature research topic, focusing on raw classification performance of a supervised model, albeit of novel DL models, does not help making academic models stepping out academic venues, for which it would be imperative to tackle other pressing problems that impact models deployability in the real world [20].

Deployability issues also includes aspects related to *model training*, such as for instance *continuous* (to tackle drift of existing classes [56] or incrementally adding zero-day application [57]), or *distributed* (e.g., privacy respectful federated learning [45], [58]) or *parsimonious* learning (e.g., due to heavy tailed application popularity, samples of zero-day applications will be scarcer with respect to well established classes, so that few-shot learning [59] technique are necessary to add classes beyond the top-200, even with commercial grade datasets such as the one used in this paper).

Finally, deployability issues especially includes aspects related to *model inference*, such as *auditing/explainability* of classification decisions for the non experts (which has practical relevance since unlike decision trees, DL models have no direct explanation [60], [61]) as well as more precise assessment of *computational costs* (e.g., to ensure the model execution is within the CPU/energy budget [19]), an aspect that this paper briefly covers but does not fully elucidate.

References

[1] M. Roughan *et al.*, "Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification," in *ACM IMC*, 2004.

[2] A. W. Moore and K. Papagiannaki, "Toward the accurate identification of network applications," in *Proc. PAM*, 2005.

[3] L. Bernaille *et al.*, "Traffic classification on the fly," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 2, pp. 23–26, 2006.

[4] M. Crotti *et al.*, "Traffic classification through simple statistical fingerprinting," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 1, pp. 5–16, 2007.

[5] D. Bonfiglio *et al.*, "Revealing skype traffic: when randomness plays with you," in *Proc. ACM SIGCOMM*, 2007.

[6] H. Kim *et al.*, "Internet traffic classification demystified: myths, caveats, and the best practices," in *Proc. ACM CoNEXT*, 2008.

[7] T. T. Nguyen and G. J. Armitage, "A survey of techniques for internet traffic classification using machine learning." *IEEE Communications Surveys and Tutorials*, vol. 10, no. 1-4, pp. 56–76, 2008.

[8] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[9] Z. Wang, "The applications of deep learning on traffic identification," *BlackHat USA*, 2015.

[10] V. F. Taylor *et al.*, "Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic," in *Proc. IEEE EuroS&P*, 2016.

[11] W. Wang *et al.*, "Malware traffic classification using convolutional neural network for representation learning," in *Proc. IEEE ICOIN*, 2017.

[12] W. Wang *et al.*, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *Proc. IEEE ISI*, 2017.

[13] M. Lopez-Martin *et al.*, "Network traffic classifier with convolutional and recurrent neural networks for internet of things," *IEEE Access*, vol. 5, pp. 18 042–18 050, 2017.

[14] Z. Chen *et al.*, "Seq2img: A sequence-to-image based approach towards ip traffic classification using convolutional neural networks," in *Proc. IEEE BigData*, 2017, pp. 1271–1276.

[15] L. Vu *et al.*, "A deep learning based method for handling imbalanced problem in network traffic classification," in *ACM International Symposium on Information and Communication Technology*, 2017.

[16] G. Aceto *et al.*, "Mobile encrypted traffic classification using deep learning," in *Proc. IEEE TMA*, 2018.

[17] T. Shapira and Y. Shavitt, "Flowpic: Encrypted internet traffic classification is as easy as image recognition," in *IEEE INFOCOM Workshops*, 2019.

[18] M. Lotfollahi *et al.*, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Computing*, vol. 24, no. 3, 2020.

[19] M. Gallo *et al.*, "Real-time deep learning based traffic analytics," in *ACM SIGCOMM, Demo session*, Aug. 2020.

[20] F. Pacheco *et al.*, "Towards the deployment of machine learning solutions in network traffic classification: A systematic survey," *IEEE Communications Surveys and Tutorials*, pp. 1–1, 2018.

[21] S. Rezaei *et al.*, "Large-scale mobile app identification using deep learning," *CoRR*, vol. abs/1910.02350, 2019. [Online]. Available: http://arxiv.org/abs/1910.02350

[22] R. Boutaba *et al.*, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, p. 16, 2018.

[23] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *Proc. ACM SIGMETRICS*, 2005.

[24] G. Vasiliadis *et al.*, "Midea: a multi-parallel intrusion detection architecture," in *Proc. ACM CCS*, 2011.

[25] P. S. del Rio *et al.*, "Wire-speed statistical classification of network traffic on commodity hardware," in *Proc. ACM IMC*, 2012.

[26] D. Eastlake *et al.*, "Transport layer security (tls) extensions: Extension definitions," IETF RFC6066, Tech. Rep., 2011.

[27] E. M. Rudd *et al.*, "The extreme value machine," *CoRR*, vol. abs/1506.06112, 2015. [Online]. Available: http://arxiv.org/abs/1506.06112

[28] A. Bendale and T. E. Boult, "Towards open set deep networks," *CoRR*, vol. abs/1511.06233, 2015. [Online]. Available: http://arxiv.org/abs/1511.06233

[29] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks," *CoRR*, vol. abs/1610.02136, 2016.

[30] J. Zhang *et al.*, "Autonomous unknown-application filtering and labeling for dl-based traffic classifier update," in *INFOCOM 2020*, 2020.

[31] Z. Ge *et al.*, "Generative openmax for multi-class open set classification," *CoRR*, vol. abs/1707.07418, 2017. [Online]. Available: http://arxiv.org/abs/1707.07418

[32] L. Neal *et al.*, "Open set learning with counterfactual images," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[33] M. Hassen and P. K. Chan, "Learning a neural-network-based representation for open set recognition," *CoRR*, vol. abs/1802.04365, 2018.

[34] E. Aljalbout *et al.*, "Clustering with deep learning: Taxonomy and new methods," *CoRR*, vol. abs/1801.07648, 2018.

[35] K. G. Dizaji *et al.*, "Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization," *CoRR*, vol. abs/1704.06327, 2017.

[36] J. Yang *et al.*, "Joint unsupervised learning of deep representations and image clusters," *CoRR*, vol. abs/1604.03628, 2016.

[37] T. DeVries and G. W. Taylor, "Learning confidence for out-of-distribution detection in neural networks," 2018.

[38] R. Yoshihashi *et al.*, "Classification-reconstruction learning for open-set recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[39] S. Zhao *et al.*, "Towards unknown traffic identification via embeddings and deep autoencoders," in *2019 26th International Conference on Telecommunications (ICT)*, 2019, pp. 85–89.

[40] J. Jang and C. O. Kim, "One-vs-rest network-based deep probability model for open set recognition," 04 2020.

[41] J. Erman *et al.*, "Offline/realtime traffic classification using semi-supervised learning," *Performance Evaluation*, 2007.

[42] J. Zhang *et al.*, "Robust network traffic classification," *IEEE/ACM Transactions on Networking*, vol. 23, no. 04, pp. 1257–1270, jul 2015.

[43] S. Liang *et al.*, "Principled detection of out-of-distribution examples in neural networks," *CoRR*, vol. abs/1706.02690, 2017. [Online]. Available: http://arxiv.org/abs/1706.02690

[44] K. Lee *et al.*, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," 2018.

[45] C. B. Lixuan Yang and D. Rossi, "Heterogeneous data-aware federated learning," in *Proc. IJCAI Federated Learning workshop*, 2020.

[46] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[47] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proc. ACM KDD*, 2016.

[48] D. Ulmer and G. Cinà, "Know your limits: Uncertainty estimation with relu classifiers fails at reliable ood detection," 2021.

[49] D. Wen *et al.*, "An overview of data-importance aware radio resource management for edge machine learning," 2019.

[50] M. Nasr *et al.*, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," *IEEE Symposium on Security and Privacy*, 2019.

[51] Z. C. Lipton and J. Steinhardt, "Troubling trends in machine learning scholarship," *Communications of the ACM*, vol. 62, pp. 45–42, 2019.

[52] M. Hutson, "Eye-catching advances in some ai fields are not real," *Science Magazine*, May 2020.

[53] Y. Wang *et al.*, "Generalizing from a few examples: A survey on few-shot learning," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–34, 2020.

[54] G. Hinton *et al.*, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[55] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *JMLR*, vol. 9, pp. 2579–2605, 2008.

[56] V. Carela-Español *et al.*, "A streaming flow-based technique for traffic classification applied to 12+ 1 years of internet traffic," *Telecommunication Systems*, vol. 63, no. 2, pp. 191–204, 2016.

[57] S.-A. Rebuffi *et al.*, "icarl: Incremental classifier and representation learning," in *IEEE CVPR*, 2017.

[58] J. Konečný *et al.*, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[59] Y. Wang *et al.*, "Generalizing from a few examples: A survey on few-shot learning," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–34, 2020.

[60] C. Beliard *et al.*, "Opening the deep pandora box: Explainable traffic classification," in *Proc. IEEE INFOCOM, Demo session*, 2020.

[61] Z. Meng *et al.*, "Interpreting deep learning-based networking systems," in *Proc. ACM SIGCOMM*, 2020.