

# Many or Few Samples?

## Comparing Transfer, Contrastive and Meta-Learning in Encrypted Traffic Classification

Idio Guarino<sup>†</sup>, Chao Wang, Alessandro Finamore, Antonio Pescapè<sup>†</sup>, Dario Rossi

<sup>†</sup> Università degli Studi di Napoli Federico II, Huawei Technology France

**Abstract**—The popularity of Deep Learning (DL), coupled with network traffic visibility reduction due to the increased adoption of HTTPS, QUIC, and DNS-SEC, re-ignited interest towards Traffic Classification (TC). However, to tame the dependency from task-specific large labeled datasets we need to find better ways to learn representations that are valid across tasks. In this work we investigate this problem comparing *transfer learning*, *meta-learning* and *contrastive learning* against reference Machine Learning (ML) tree-based and monolithic DL models (16 methods total). Using two publicly available datasets, namely MIRAGE19 (40 classes) and AppClassNet (500 classes), we show that (i) by using DL methods on large datasets we can obtain more general representations with (ii) contrastive learning methods yielding the best performance and (iii) meta-learning the worst one. While (iv) tree-based models can be impractical for large tasks but fit well small tasks, (v) DL methods that reuse better learned representations are closing their performance gap against trees also for small tasks.

### I. INTRODUCTION

Network monitoring is paramount for networks operation, with *Traffic Classification* (TC) being a strategic activity to empower better decision making. Started more than a decade ago, the transition towards completely encrypted network traffic is today reinvigorated by the growing adoption of QUIC [1], DNSSEC [2], or initiatives like Apple iCloud Private Relay [3]—encryption is here to stay for end-users benefit at the cost of reduced traffic visibility for network operators.

For recovering from such reduction and pursuing better network management and automation, both academia and industry started the quest for smarter TC solutions. A first wave of Machine Learning (ML)-based solutions was already introduced twenty years ago, while more recent proposals focus on Deep Learning (DL) [4]—nowadays, TC is the king of supervised modeling tasks in network traffic analysis.

Training and managing ML/DL models for networks faces the “infinite loop” of collecting new data and re-train models to keep them up to date. The key to break this cycle resides in (i) training more generalized models and (ii) adapting them to new scenarios by means of little-to-no extra data. This calls for exploring *transfer learning* and *Few-Shot Learning* (FSL), two DL techniques designed to foster better representation learning. These techniques allow to reuse what learned from a source task  $\mathcal{T}_{source}$  to address a different task  $\mathcal{T}_{target}$ . Within the same scope, *contrastive learning* and *self-supervision* come with the promise of better knowledge extraction thanks to smarter use of data augmentation.

Considering TC literature, we find several studies investigating FSL in the context of intrusion detection [5, 6, 7, 8, 9, 10, 11, 12]. Yet, (i) most of them do not follow the conventional

meta-learning training protocol, hence (arguably) biasing their takeaways (see Sec. II, Sec. III); only [13, 14] investigated contrastive learning; no previous literature relies on state-of-the-art transfer learning techniques [15, 16]. Moreover, (ii) little attention has been spent on investigating ML tree-based approaches. Considering datasets, (iii) we find most studies using only up to 20 classes, i.e., possibly not enough variety to understand the training methodologies pros and cons. Last, (iv) most methods rely on payload bytes, reshaped as large 2d matrix, which can be costly to track for monitoring systems.

Motivated by the previous considerations, in this work we study transfer learning, meta-learning, and contrastive learning (a total of 16 variants) by means of two public datasets, namely MIRAGE19 (40 classes) and AppClassNet (500 classes). Beside benchmarking the methods, we aim at *understanding up to which extent large TC datasets can be used for learning better representation via DL*. For completeness, we study also ML tree-based models and traditional CNN-based models.

Our results show that TC literature might have overstated meta-learning methods benefits which are the worst performing in our assessment (at least  $-14/18\%$  from the best alternative). Conversely, and aligned with recent research in computer vision, contrastive learning (especially in a supervised setting) is quite effective, yet suffering from computational costs. Tree-based models are still superior to all methods but, while they can grow too deep—a 500 classes model on AppClassNet can grow up to a depth of 117 corresponding to 416GB file size—they are still the most practical solution when modeling classes with  $\leq 1,000$  samples/class. Still, the best DL alternatives are closing the performance gap against tree-based models.

In the following, we start by introducing the principles behind the DL techniques (Sec. II) and reviewing related computer vision and TC literature (Sec. III). We continue stating our research goals (Sec. IV) used when designing our experiments (Sec. V). We conclude by discussing our results (Sec. VI) and outlining future research avenues (Sec. VII).

### II. BACKGROUND

In this section, we review DL models training principles for monolithic, transfer, meta- and contrastive learning. We focus on a supervised task  $\mathcal{T}$  with a dataset  $\mathcal{D} = \{(x_1, y_1) \cdots (x_m, y_m)\}$ , where each input sample  $x \in \mathcal{X}$  maps to a class label  $y \in \mathcal{C} = \{1 \dots N\}$ .  $\mathcal{D}$  is partitioned into  $\mathcal{D}_{train}$ ,  $\mathcal{D}_{val}$  and  $\mathcal{D}_{test}$  and each  $(x_i, y_i)$  belongs to one partition only.

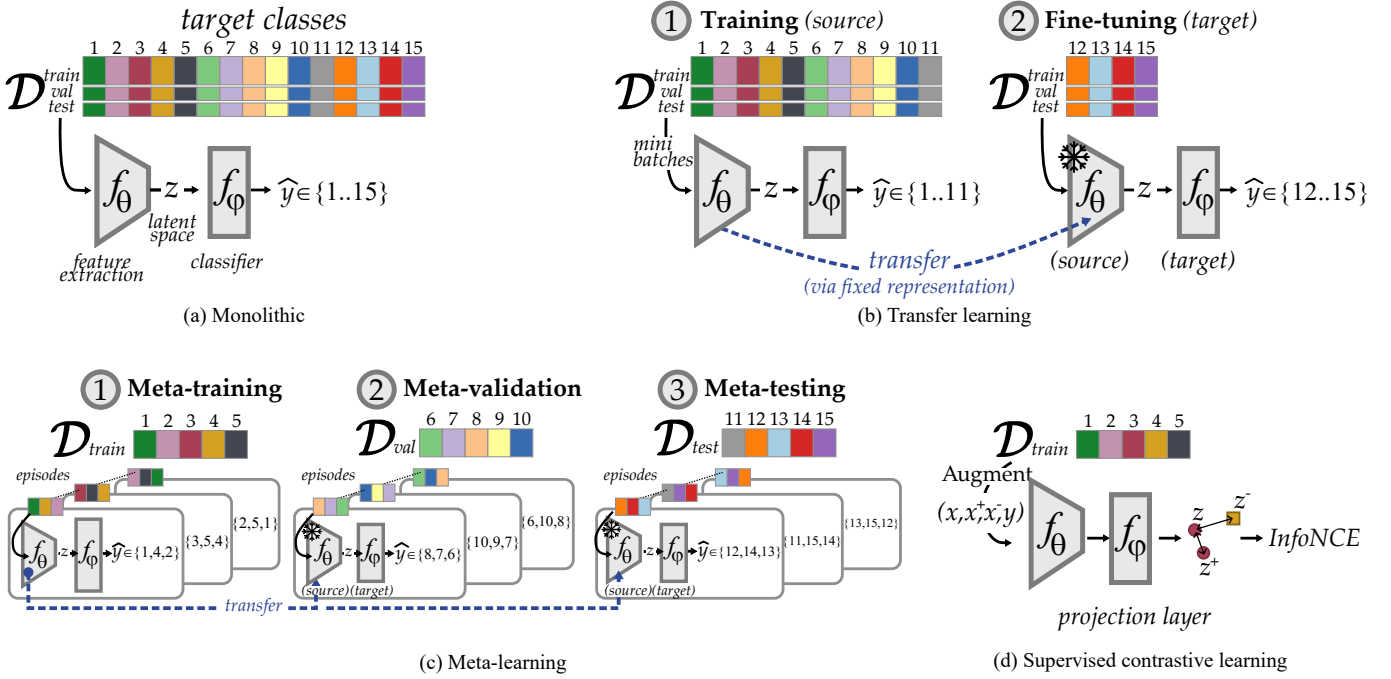


Fig. 1: DL training strategies comparison.

### A. Monolithic training

A DL model embodies a function  $f_{\theta, \varphi} : \mathcal{X} \rightarrow \mathcal{C}$  mapping input samples to labels. As sketched in Figure 1(a), the model decouples feature extraction (*a.k.a.* encoding, embedding)  $f_{\theta} : \mathcal{X} \rightarrow \mathcal{Z}$  from classification  $f_{\varphi} : \mathcal{Z} \rightarrow \mathcal{C}$  by means of an intermediate latent space  $\mathcal{Z}$  where each class projected samples should be easily separable. Training such model requires tuning the function parameters  $(\theta, \varphi)$  to minimize the difference (*a.k.a.* the classification loss) between true labels  $y$  and predicted ones  $\hat{y}$ . To do so, during training  $\mathcal{D}_{train}$  is re-played multiple times (*a.k.a.* epochs) through the model by means of mini-batches (each being a collection of inputs sampled from  $\mathcal{D}_{train}$ ) to progressively adjust the parameters. Each training epoch is usually validated by assessing classification performance on  $\mathcal{D}_{val}$ , with model parameters temporarily frozen for quantifying over-fitting (i.e., verifying that the model can classify data not belonging to the training set) and searching optimal hyper-parameters. Lastly, testing on  $\mathcal{D}_{test}$  yields the final classification performance. As sketched in Fig. 1(a), training, validation, and testing (*i*) share the same task which targets all available classes—the model is *monolithic*—and (*ii*) the dataset partitions are scanned at least once.<sup>1</sup> As we shall see, while transfer learning respects both principles, meta-learning violates them both.

### B. Transfer learning

Transfer learning [17] enables to address a target task  $\mathcal{T}_{target}$  via a model  $M_{source}$  trained for a different task

<sup>1</sup>In each training epoch, the dataset is shuffled and then sequential micro-batches are formed—this is semantically a scan of the dataset. The process is the same in validation and testing, although shuffling is unnecessary.

$\mathcal{T}_{source}$ . Intuitively, the stronger the relationship between  $\mathcal{T}_{source}$  and  $\mathcal{T}_{target}$ , the higher the chance the knowledge in  $M_{source}$  can be used (hence transferred) to  $\mathcal{T}_{target}$ . However, exactly quantifying task affinity is challenging [18], but training on large datasets is a “cheap” way to bypass the problem [19]. As sketched in Fig. 1(b),  $M_{target}$  is composed of the already trained feature extractor  $f_{\theta}^{(source)}$  (*a.k.a.*, trunk, backbone, encoder) and a new classifier  $f_{\varphi}^{(target)}$  to be trained from scratch and possibly having a different number of classes and/or architecture with respect to  $\mathcal{T}_{source}$ . Given the mixture of trained and untrained parameters, training of  $M_{target}$  is referred to as *fine-tuning* and can be performed in a few ways. Fig. 1(b) shows *fixed representation* scenarios where the transferred trunk is *frozen*  $\star$  when training the new classifier, i.e., only the new classifier parameters are optimized. Fixed representation is commonly used to compare different representation learning methods, but alternatively, one can fine-tune all parameters at once, or use a hybrid policy unfreezing trunk parameters after a certain number of epochs. No matter the selected option, transfer learning still adheres to monolithic training—dataset partitions are entirely scanned and  $\mathcal{T}_{target}$  enlists all available classes, yet those are disjoint with respect to  $\mathcal{T}_{source}$ .

### C. Meta-learning and episodic training

While transfer learning relies on *implicit* tasks affinity, meta-learning is designed to *explicitly* push cross-tasks representation extraction. To do so, (*i*)  $\mathcal{D}_{train}$ ,  $\mathcal{D}_{val}$  and  $\mathcal{D}_{test}$  contain disjoint set of classes and (*ii*) training is based on the generation of synthetic tasks  $\mathcal{T}_i$  called *episodes*. During an epoch, rather than completing a scan of  $\mathcal{D}_{train}$  by means

of random mini-batches, synthetic tasks  $\mathcal{T}_i$  are created by randomly sampling a subset of training classes  $\mathcal{C}_i^{(train)}$ . Then, for each class two disjoint sets of samples are randomly formed to drive the learning, namely *support*  $\mathcal{S}_i$  and *query*  $\mathcal{Q}_i$ . Episodic training is synonym of Few-Shot Learning (FSL) [20] and is also known as (*N-way, S-shot*) training: an episode contains  $N$  classes (ways) each having two sets of samples, namely  $S \stackrel{\text{def}}{=} |\mathcal{S}_i|$  shots and  $Q \stackrel{\text{def}}{=} |\mathcal{Q}_i|$  query samples acting as a “micro-training” and “micro-validation” set. We highlight that sometimes FSL is broadly referred to training (or fine-tuning) with small datasets but without episodic training (e.g., [13]). However, in this paper FSL is synonym for meta-learning.

Notice that while episodes are small batches of samples, by design they differ from monolithic training mini-batches as they guarantee class balance. In monolithic training, mini-batches are formed by randomly selecting samples across the dataset hence (i) they reflect the dataset class imbalance (if any) and (ii) even for already balanced dataset, there is no guarantee that all target classes to be present in a mini-batch. Conversely, in meta-learning, an episode has  $(S + Q)$  samples for each of the (randomly selected)  $N$  target classes (ways).

Overall, this process is known as meta-training and is complemented by the remaining two phases, meta-validation and meta-testing which still rely on fine-tuning the target models, but recall that  $\mathcal{D}_{train}$ ,  $\mathcal{D}_{val}$  and  $\mathcal{D}_{test}$  are disjoint. More in details, the target task  $M_{target}$  is composed by transferring the meta-learned source model trunk  $f_{\theta}^{(source)}$  and fine-tune a classifier specific for an episode. In other words, as pictured in Fig. 1(c), the evaluation (being validation or testing) creates multiple models (one per episode) so overall performance is an aggregation of per-episode models performance. As in monolithic training, meta-validation facilitates both hyper-parameters tuning and over-fitting assessment, while meta-testing yields the final performance. Although not strictly required, it is also common to maintain the same *N-ways* for meta-train, meta-validation and meta-test.

As for transfer learning, it is beneficial to use a  $\mathcal{D}_{train}$  offering many classes (e.g.,  $\geq 100$ ) but episodes are commonly small, i.e.,  $N=\{5, 10\}$  classes each having  $S=\{1, 5\}$  support samples and  $Q=\{10, 50\}$  query samples—meta-training is designed to push representation learning through tasks variety.

#### D. Contrastive learning

From a geometrical standpoint, a well trained supervised classifier should project input samples in the latent space so to pull together samples from the same class while distancing them from other classes samples. Recall that the common model classifier  $f_{\varphi}(\cdot)$  is a simple linear layer, hence a traditional classification loss (e.g., cross-entropy loss) *implicitly* trains the feature extractor to achieve such geometrical property. Contrastive learning [21] is a technique to *explicitly* regularize latent space geometry by means of data augmentation and a “contrastive game”: an input sample  $x \in \mathcal{X}$  (a.k.a. anchor) is transformed (via rotation, cropping, jittering, blurring, etc.) into a *positive sample*  $x^+ = \text{Augment}(x)$  while the remaining available samples in  $\mathcal{X}$  can be used as *negative*

TABLE I: Computer vision literature summary.

	Approach	KnowDistil?	Classifier	Dist-based?
Transf. Learn.	[15] Baseline	○	Linear	○
	[15] Baseline++	○	Cosine Sim.	●
	[16] RFS-simple(LR)	○	Logistic Reg.	○
	[16] RFS-distill(LR)	●	Logistic Reg.	○
	[16] RFS-simple(NN)	○	Nearest Neighbor	●
	[16] RFS-distill(NN)	●	Nearest Neighbor	●
Meta. Learn.	[27] ProtoNet	○	Euclidean distance	●
	[28] RelationNet	○	MSE	○
	[29] MAML	○	Linear	○
Contr. Learn.	[21] SimCLR	○	none	●
	[26] SupCon	○	none	●

*samples*; then the training loss is formed to penalize cases where  $(f_{\theta}(x), f_{\theta}(x^+))$  are far apart and  $(f_{\theta}(x), f_{\theta}(x^-))$  are too close. Multiple losses can drive the learning process with *InfoNCE* [22], which maximizes the mutual information between anchor and positive samples, being the most popular choice. Overall, contrastive learning differs from SMOTE [23], ADASYN [24] and other augmentations (e.g. random gaussian jittering) which “blindly” add variety to  $\mathcal{D}_{train}$  but no latent space regularization is explicitly introduced.

Contrastive learning is at the core of Self-Supervised Learning (SSL) and is defined for unsupervised tasks—with no extra information, a sample can only be *self-similar*, i.e., a positive sample is the transformation of the sample itself—with a wide design space related to the selection of transformations, positives and negatives samples [25]. As from Fig. 1(d), notice the presence of  $f_{\varphi}(\cdot)$  acting as a projection layer, not as a classifier. Even supervised variants of contrastive learning (e.g., [26]) are still effectively unsupervised as labels guide the positive samples selection, not a classification loss.

### III. RELATED WORK

In this section, we complement the DL training principles introduced so far with a review of computer vision and traffic classification literature as summarized in Table I and Table II.

#### A. Transfer learning

**CV literature.** Transfer learning is the most adopted methodology across literature. In particular, while early meta-learning literature proves it is possible to learn even from single samples [27], more recent literature [15, 16] suggests transferring from models created on large dataset (with many samples and many classes) yields better performance, especially when combined with fine-tuning with episodic training for the final target task, i.e., they mix transfer learning with meta-learning.

A prominent example is Baseline [15] which trains a  $M_{source}$  model using a large dataset (many classes and many samples) and then fine-tune a target task (from a disjoint set of classes) via episodic training. Baseline++ [15] is a variant of the same approach where the parameters of the classifier are treated as *class embedding*. More in details, both methods rely on a fully connected layer  $\mathbf{W} \in \mathbb{R}^{d \times c}$  where  $d$  and  $c$  represent the dimension of the latent space vectors and the number of target classes respectively. Unlike Baseline that simply

TABLE II: Traffic classification literature summary.

	Reference	Approach	Data	Classes		Shots	Inp.Type	Inters?
			Year	All	Target			
Transf. Learn.	[31] <i>Razaei20</i>	Multi-task learn.	18	5	5	n.a.	3 PS	●
	[32] <i>Sun18</i>	TrAdaBoost	05	12	12	n.a.	- FF	●
Meta Learn.	[5] FS-IDS	ProtoNet	14	8	3	1:10	26 FF	●
	[6] UMVD-FSL	ProtoNet	17,20	≈76	5/20	1/5	784 B	○
	[7] <i>Yu20</i>	ProtoNet	09,15	≈8	2/5	50	44 FF	●
	[8] OICS-VFSL	ProtoNet	09,15	≈12	2/4/7	1:50	-FF	●
	[9] RBRN	RelationNet	12,16	≈15	—	—	-B	●
	[10] FCNet	RelationNet	12,17	≈10	2	5/10	200 B	●
	[11] Festic	RelationNet	18,19	≈25	14	5:15	256 B	●
	[12] FCAD	MAML	17	43	13	5:20	33FF+8PS	○
Contr. Learn.	[13] <i>Horowicz22</i>	SimCLR	18	5	5	—	FlowPic	●

InpType: FF = flow features; B = payload bytes; PS = univariate packet time series; — = unspecified

Inters? target and base classes are ○disjoint or either ●completely or ●partially overlapped.

uses a standard cross-entropy classification loss, **Baseline++** relies on a cosine distance-based loss between  $\mathbf{W}=[\mathbf{w}_1 \cdots \mathbf{w}_c]$  columns and latent space projections  $z$  of input samples—each column  $\mathbf{w}_i$  “embeds” a class. Similar mechanisms power also RFS-simple(LR) and RFS-simple(NN) [16] differing from the previous only by the classifier  $f_\varphi(\cdot)$  being a logistic regression—a linear layer with sigmoid activations rather than softmax—and a 1-nearest neighbour approach respectively—the model does not have a classifier layer but rather classifies based on proximity to pre-defined labeled projected samples. RFS authors also introduced two other variants, namely RFS-distill(LR) and RFS-distill(NN), based on *self-distillation*, a special form of knowledge distillation [30]. In a nutshell, to reinforce models knowledge, the trained model is further fine-tuned by replaying  $\mathcal{D}_{train}$  and pairing the classification loss with an additional term comparing a smoother version (via temperature scaling) of logits with respect to the fine-tuned logits—in practice, the task becomes intentionally more complex to further push the learning. It follows that RFS-distill(LR) and RFS-distill(NN) use a 3-steps training: training the source model, apply distillation and finally fine-tune the target tasks.

**TC literature.** Only [31, 32] used transfer learning but none of the methods mentioned above. Namely, [31] pre-trained an unsupervised multi-task model targeting flows duration and bandwidth which was then transferred to a 5 classes task. Results show that this transfer was under-performing with respect to training directly a single-task model. Instead, [32] used TrAdaBoost [33], an ensemble ML method using reversed boosting, considering a very old dataset.

### B. Meta-learning approaches

**CV literature.** Computer vision literature is ripe with meta-learning and FSL methods [20]. In this study we focus on a small selection of methods based on their extreme popularity. ProtoNet [27] is the most well known metric-based meta-learning approach. ProtoNet learns class prototype which geometrically corresponds to the mean centroid of a class in the latent space. Query samples are then classified based on their euclidean distance with respect to class prototypes. The idea of class prototypes inspired different meta-learning meth-

ods, including Baseline(ClassEmb) (i.e., class embedding are semantically equivalent to class prototypes).

RelationNet [28] is another popular metric-based meta-learning method. While ProtoNet uses a closed form distance metric (i.e., euclidean distance), RelationNet introduces the idea of “meta-learning” such distance. Specifically, the classifier  $f_\varphi(\cdot)$  embodies a “relation” module trained to provide a similarity score between support and query samples. Curiously, the classification loss is based on Mean Squared Error (MSE) rather than softmax.

MAML [29] is the most popular optimization-based meta-learning approach. Differently from ProtoNet and RelationNet which optimize model parameters considering episodes in isolation, MAML uses a two-nested loops process: the inner loop fine-tunes based on each individual episode; the outer loop “re-weights” inner loop contributions across episodes via a second order gradient of the classification loss.

**TC literature.** As from Table II, several studies successfully applied these methods on network traffic, mostly targeting normal-vs-attack classification tasks in intrusion detection scenarios. While we consider those classifiers as specific forms of TC, we find *most of these studies violate the meta-learning principle of dis-joining train/val/test partitions*. Only [6, 12] followed the expected protocol, while in all the other studies the partitions overlapped either perfectly (e.g., meta-train a binary classifier normal-vs-attack and meta-test on the same classes) or partially (e.g., normal and the same attack classes traffic belongs to both meta-train and meta-test).

### C. Contrastive learning

**CV literature.** Similarly to FSL, also contrastive learning is a very active research area. Across variants, SimCLR [21] is the most popular method. SupCon [26] extends SimCLR for a “supervised” setting by simply considering as positive samples also all other augmented version of samples belonging to the same class of the selected anchor—it moves from self-similarity to class-similarity.

**TC literature.** We find only two previous work using contrastive learning for TC. In [13], authors applied contrastive learning in a “few-shot learning settings” without episodic training. More specifically, authors trained an unsupervised model using SimCLR and with a flowpic input representation—packet time-series are transformed into images representing the evolution of traffic over a time window; hence transformation is possible by either manipulating the time series (e.g., time shift) or the related image (color jittering, occlusion, etc.)—and transferred it to the supervised classification task using just a few labeled samples. In [14] instead, authors apply BYOL [34]—another popular contrastive learning methods that does not rely on negative samples—to a similar problem setting as in [13].

Worth also of mention is [35] where authors used augmentations similar to the one used in [13], yet no contrastive learning was applied.



## IV. RESEARCH QUESTIONS

TC literature favors meta-learning with respect to transfer learning (Table II). Yet, computer vision literature suggests to pay attention to the latter. Moreover, all previous TC literature on meta-learning provides positive results, but most of these studies violate the meta-learning principle of dis-joining the classes in  $\mathcal{D}_{train}$ ,  $\mathcal{D}_{val}$  and  $\mathcal{D}_{test}$ . Thus, we claim the need to re-assess these methodologies under different settings.

Q1: *Is transfer learning yielding better performance compared to meta-learning? Do the benefits, as observed in computer vision, of transferring from source models trained on many samples and many classes apply to TC use-cases as well?*

FSL targets scenarios suffering from limited amount of labeled samples. For instance, in TC this happens when new applications are introduced. However, differently from computer vision, we can also rely on traditional ML models to address these small tasks. Conversely previous TC literature focused mostly on DL methods adopting image-like input representation, while we argue that time series input is a more natural choice for network monitoring.

Q2: *How meta-learning performance compares with common ML tree-based models when considering as input time series of properties of the first  $P$  packets of a flow? Do more shots improve performance?*

Next to transfer learning and meta-learning, also contrastive learning is designed to facilitate better representation learning. Since it has been shown to be successful for time series [36] outside the TC domain, we ask

Q3: *Does contrastive learning assist in creating more general models also for TC? Is supervised contrastive learning superior to its more traditional version?*

## V. METHODOLOGY

To address our research questions we relied on two publicly available datasets and designed multiple modeling campaigns.

### A. Datasets

MIRAGE19 [37] encompasses per-biflow traffic logs of 40 Android apps collected at the ARCLAB laboratories of the University of Napoli Federico II. It was collected by instrumenting 3 Android devices used by  $\approx 300$  volunteers (students and researchers) interacting with the selected apps for short sessions. Each session resulted in a pcap file and an `strace`<sup>2</sup> log mapping each socket to the corresponding Android package name. Pcaps were then post-processed to obtain biflow logs by grouping all packets belonging to the same 5-tuple (srcIP, srcPort, dstIP, dstPort, L4proto) and extracting both aggregate metrics (e.g., total bytes, packets, etc.), per-packet time series (packet size, direction, TCP flags, etc.), raw packets payload bytes (encoded as list of integer values) and mapping a ground-truth label by means of the `strace` logs.

<sup>2</sup><https://man7.org/linux/man-pages/man1/strace.1.html>

TABLE III: Datasets summary.

Data Partition	MIRAGE19					AppClassNet				
	Num. Classes	Samples			$\rho$	Num. Classes	Samples			$\rho$
		All	Max	Min			All	Max	Min	
$\mathcal{D}_{train}$	24	82k	8.2k	1.3k	6.3	320	9.8M	1M	958	1,044
$\mathcal{D}_{val}$	8	9.4k	1.3k	1.1k	1.2	80	60.5k	956	579	1.6
$\mathcal{D}_{test}$	8	5.1k	904	361	2.5	100	47.4k	578	383	1.5
$\mathcal{D}_{all}$	40	97k	8.2k	361	22.7	500	9.9M	$\approx 1M$	383	2,611

$\rho$  = ratio *Max/Min* samples per class

AppClassNet [38] is a commercial-grade dataset released by Huawei Technologies in 2022 and gathered from real residential and enterprise networks. The dataset encompasses the traffic of 500 applications for a total of 10M biflows each represented by a time series of packet-size and direction of the first 20 packets and, most important, labeled by means of a commercial and proprietary DPI tool. Prior to release, the dataset was anonymized to remove *privacy-sensitive* information (i.e., IP addresses, exact timing, protocol headers field values, and packet-payload) and *business-sensitive* information (i.e., applications labels are encoded as integer values and raw time series values are anonymized).

**Dataset partitioning.** Table III summarizes datasets properties. Defining the popularity of a class as its number of samples, the table reports  $\rho$  measuring the datasets imbalance as the ratio between the most popular and least popular class. As expected, the imbalance is severe; yet such condition is rare in computer vision scenarios, especially in meta-learning settings—typical FSL datasets like Omniglot [39], miniImageNet [40] and CIFAR-FS [41] have all  $\rho=1$ . Adhering to meta-learning protocols we partition the datasets by dis-joining train/val/test. To do so, we use class popularity resulting in  $\mathcal{D}_{train}$  containing the largest classes pool—imbalance here still reflects network traffic scenarios and the large availability of data allows to address (Q1)—while  $\mathcal{D}_{val}$  and  $\mathcal{D}_{test}$  focus on unpopular classes—imbalance here is reduced, better reflecting the typical meta-learning settings (Q2). We argue that such partitioning is preferable to both (i) artificially random under-sampling to enforce a “few-shot” setting and (ii) randomly splitting classes obtaining scenarios where a target class has many samples. Conversely, we aim at target tasks with a naturally reduced number of samples.

**Input type.** All models are created using packet time series as input (Q2). Specifically, for MIRAGE19 we consider 4 features (packet size, direction, inter-arrival time, and TCP window-size<sup>3</sup>) for the first 10 packets; for AppClassNet we consider 2 features (packet size and direction) for the first 20 packets.<sup>4</sup>

### B. DL methods

**Approaches.** We considered a total of 16 methods: 1 for monolithic training, 8 for transfer learning, 3 for meta-

<sup>3</sup>For UDP traffic the time series is padded with 0.

<sup>4</sup>The reason for the different time series length resides in properties of the datasets: MIRAGE19 contains many short leaved flows thus using 10 packets reduces padding.

learning, and 4 for contrastive learning. We use all methods reported in Table I. However, for transfer learning, we modify original names to better express their relationship and semantic: `Baseline++`  $\rightarrow$  `Baseline(ClassEmb)`, `RFS-simple(LR)`  $\rightarrow$  `Baseline(LR)` and `RFS-simple(NN)`  $\rightarrow$  `Baseline(NN)`.<sup>5</sup> Still on transfer learning, we added the two variants `Baseline-TL` and `Baseline(ClassEmb)-TL` that fine-tune via monolithic training rather than episodic training. For contrastive learning, we considered both unsupervised (SimCLR) and supervised (SupCon) variants (Q3) considering models with and without Baseline’s class embedding (4 variants). We also randomly apply 4 transformations: *horizontal flip* reverses the order of packets (1st become last, etc.); *shuffle* randomly reorders packets; *tail-occlusion* masks the second half of an input time series with zeros; *gaussian noise* adds noise sampled from a normal distribution  $\epsilon \sim \mathcal{N}(0, 1)$  to each time series value.

**Architectures.** We used CNN architectures based of 2d convolutional blocks (a convolution layer followed by a batch normalization layer and a ReLU activation): CNN-2 is inspired by [42] and has two CNN blocks with 32 and 64 filters respectively followed by a fully connected layer of 200 units (trunk with 529k parameters); CNN-4 adds other two CNN blocks and a fully connected layer of 500 units (trunk 1.3M parameters).

**Implementation.** We complemented the Pytorch implementation provided by [15, 16] by adding monolithic training, traditional transfer learning, and contrastive learning methods.

### C. ML methods

We compared all DL approaches against a reference Random Forest (RF) provided by `sklearn` [43] and fed with the concatenated packet time series (Q2).

### D. Experimental scenarios

We settled on scenarios intentionally designed to go beyond the traditional FSL settings where meta-training and meta-testing share the same  $N$ -ways, and number of shots is limited to  $S \in \{1, 5\}$ . Trained DL  $M_{source}$  models are used in fixed representation (Sec. II-B). We also experimented with different monolithic models to gather “reference” comparison performance. Overall, the training campaigns run on multiple linux servers equipped with multiple nVidia V100 GPUs.

**Monolithic models.** When using monolithic training  $\mathcal{D}_{val}$  is discarded<sup>6</sup> and the actual training and validation set are obtained from a 9:1 partitioning of  $\mathcal{D}_{train}$ . For RF we used 100 estimators but varying  $max\_dept \in \{\text{unbounded}, 10, 30\}$ . For DL methods we used a batch size of 64 for MIRAGE19 and 1,024 for AppClassNet with a static learning rate of 0.001.

**Episodic training.** For episodic training, we varied the number of shots  $S \in \{5, 15, 50, 100, 200\}$  while keeping  $Q=15$  query

<sup>5</sup>We acknowledge that `Baseline` is a sub-optimal naming convention, but we kept it to preserve the relationship with [15].

<sup>6</sup>Alternatively we could have merged  $\mathcal{D}_{train}$  with  $\mathcal{D}_{val}$ , but transfer learning training would have enjoyed more classes than for meta-learning.

TABLE IV: Models reference performance.

Arch	Train	Test	MIRAGE19		AppClassNet	
			Accuracy	Params.	Accuracy	Params.
RF (a)	<i>all</i>	<i>all</i>	86.29 ± 0.52	2.5M / 56	66.73 ± 0.12	226M / 116
RF (b)	<i>all</i>	4 target	83.30 ± 0.21	2.5M / 56	55.95 ± 0.63	226M / 116
RF (c)	4 target	4 target	96.84 ± 0.09	26k / 22	95.88 ± 0.15	30k / 23
RF-10 (a)	<i>all</i>	<i>all</i>	57.64 ± 0.70	94.1k / 10	8.66 ± 0.08	163k / 10
RF-30 (a)	<i>all</i>	<i>all</i>	86.25 ± 0.47	2.4M / 30	52.55 ± 0.25	90M / 30
CNN-2 (a)	<i>all</i>	<i>all</i>	79.69 ± 0.06	8k	83.91 ± 0.41	101k
CNN-2 (b)	<i>all</i>	4 target	74.47 ± 1.29	529k+ 8k	76.21 ± 1.84	529k+ 101k
CNN-2 (c)	4 target	4 target	84.35 ± 0.95	804	91.26 ± 0.93	804
CNN-4 (a)	<i>all</i>	<i>all</i>	79.83 ± 0.10	20k	84.56 ± 0.31	250k
CNN-4 (b)	<i>all</i>	4 target	73.40 ± 1.33	1.3M+ 20k	75.12 ± 1.57	1.3M+ 250k
CNN-4 (c)	4 target	4 target	84.87 ± 0.71	2k	91.93 ± 2.14	2k

For RF\*,  $\{total\ nodes\} / \{avg\ depth\}$ ; For CNN\*,  $\{trunk\} + \{classifier\}$  params.

samples training for 200 epochs of 100 episodes each. Following the reference implementation, the learning rate was (slightly) different between the three approaches: ProtoNet used a policy halving the learning rate every 10 epochs, while RelationNet and MAML had a fixed learning rate of 0.001 and 0.0001 respectively. We also experimented with varying both training and testing ways (see Sec. VI-C). The only exception is MAML which in the reference implementation was not flexible enough to easily change models classifier during meta-testing.<sup>7</sup>

**Evaluation metric.** We measured classification performance using *balanced accuracy* defined as a weighted average of per-class accuracy based on the inverse of the class popularity [44].

## VI. EVALUATION

### A. Reference monolithic models

We evaluated RF and DL monolithic models in three scenarios: (a) training and testing on all classes, (b) training on all classes but testing a random set of 4 unpopular classes and (c) training and testing on a random set of unpopular classes only. We repeated the experiments 10 times, each with 5-folds cross-validation and 300 random selection of 4 unpopular classes for (b-c). Table IV reports the average balanced accuracy, 95th Confidence Intervals (CI) and models size, i.e., number of nodes and depth for RFs and number of trunk and classifier parameters for CNNs.

**Random forests.** Unbounded RF models yield the best performance across all scenarios. Yet, (a-b) models are “unrealistic upper bound” given their incredibly high complexity. Specifically, pickle serialization created files of 930MB and 416GB, i.e., 422× and 190M× a CNN-2 size for MIRAGE19 and AppClassNet respectively.<sup>8</sup> Reducing trees depth makes models leaner, but still large in absolute terms—with  $max\_depth=30$  we obtain 880MB and 21.1GB for MIRAGE19 and AppClassNet respectively—and aggressively reducing depth

<sup>7</sup>We found the same to be true in other publicly available implementations of MAML; we believe that this constraint can be alleviated but we leave it as future work.

<sup>8</sup>Sizes do not account for gzip, zlib, etc., compression thus roughly reflect the memory required to load those models for inference.

TABLE V: Comparing of transfer, meta- and contrastive learning in 4-way  $\mathcal{T}_{target}$  tasks.

Approach	MIRAGE19					AppClassNet				
	5-shots	15-shots	50-shots	100-shots	200-shots	5-shots	15-shots	50-shots	100-shots	200-shots
Baseline	60.24 ± 0.57	72.62 ± 0.48	82.26 ± 0.41	86.09 ± 0.36	88.72 ± 0.32	77.30 ± 0.65	85.42 ± 0.50	90.11 ± 0.38	91.51 ± 0.38	93.03 ± 0.32
Baseline(ClassEmb)	59.98 ± 0.54	70.78 ± 0.49	79.03 ± 0.42	82.16 ± 0.41	84.48 ± 0.38	76.54 ± 0.61	84.35 ± 0.52	89.16 ± 0.41	91.39 ± 0.34	92.30 ± 0.32
Baseline(LR)	65.50 ± 0.61	75.28 ± 0.48	82.40 ± 0.42	85.87 ± 0.38	87.99 ± 0.34	77.42 ± 0.65	84.03 ± 0.53	88.77 ± 0.41	90.48 ± 0.39	91.83 ± 0.34
Baseline(NN)	65.48 ± 0.56	77.13 ± 0.48	86.41 ± 0.36	90.26 ± 0.30	92.84 ± 0.24	76.93 ± 0.61	84.81 ± 0.51	90.18 ± 0.39	92.25 ± 0.33	93.77 ± 0.29
RFS-distill(LR)	64.62 ± 0.56	75.14 ± 0.47	82.91 ± 0.41	86.10 ± 0.36	88.46 ± 0.33	77.10 ± 0.65	83.73 ± 0.52	88.69 ± 0.42	90.64 ± 0.38	92.14 ± 0.34
RFS-distill(NN)	64.85 ± 0.58	77.37 ± 0.47	86.78 ± 0.36	90.10 ± 0.30	92.51 ± 0.25	76.65 ± 0.66	85.02 ± 0.50	89.77 ± 0.40	92.45 ± 0.34	93.83 ± 0.30
MAML	57.10 ± 0.58	65.19 ± 0.48	70.68 ± 0.44	73.92 ± 0.44	73.97 ± 0.44	61.93 ± 0.71	72.60 ± 0.66	75.57 ± 0.60	77.52 ± 0.57	78.27 ± 0.55
ProtoNet	62.62 ± 0.56	67.07 ± 0.47	69.93 ± 0.41	70.72 ± 0.42	72.09 ± 0.40	69.93 ± 0.74	77.81 ± 0.63	80.31 ± 0.51	81.05 ± 0.52	81.94 ± 0.50
RelationNet	54.28 ± 0.58	58.73 ± 0.50	57.73 ± 0.51	62.99 ± 0.45	61.60 ± 0.47	68.65 ± 0.74	72.22 ± 0.67	77.24 ± 0.59	78.54 ± 0.58	75.09 ± 0.57
SimCLR	63.97 ± 1.01	74.61 ± 0.81	79.26 ± 0.76	80.94 ± 0.58	81.52 ± 0.63	77.88 ± 2.05	86.73 ± 1.43	91.10 ± 1.16	91.18 ± 1.15	91.65 ± 1.11
SupCon	64.72 ± 0.83	77.62 ± 0.69	86.55 ± 0.50	90.07 ± 0.43	91.00 ± 0.39	81.74 ± 1.07	89.29 ± 0.82	91.70 ± 0.64	92.03 ± 0.60	93.33 ± 0.51
SimCLR(ClassEmb)	62.82 ± 0.98	76.53 ± 0.83	86.91 ± 0.57	89.89 ± 0.46	91.01 ± 0.43	78.27 ± 2.17	84.70 ± 1.84	92.05 ± 1.08	92.35 ± 0.98	93.35 ± 0.92
SupCon(ClassEmb)	66.42 ± 0.84	77.69 ± 0.68	87.01 ± 0.48	90.32 ± 0.43	91.87 ± 0.37	81.05 ± 1.09	89.63 ± 0.75	93.75 ± 0.53	95.18 ± 0.48	95.94 ± 0.44

severely affect performance.<sup>9</sup> Considering (c) instead, RF models are a very lean option as they yield the highest accuracy with small footprint ( $\approx 2$ MB).

**CNNs.** Doubling the depth of the CNN provides  $<1\%$  performance improvement, yet both CNN-2 and CNN-4 have a small memory footprint (2.2MB and 5.4MB respectively).

**Takeaways.** *On the one hand, while RF reaps the performance by dynamically growing its “architecture”, this can lead to large memory requirements. Yet, for tasks with 4 classes with  $\leq 1,000$  samples RF is still the best monolithic reference considering that DL models bounds their deployment to (i) GPU/TPU accelerators availability and (ii) non trivial performance optimization. These aspects have been overlooked by TC literature which prefers large architectures even for relatively simple tasks—e.g., [45] uses 7M parameters on a 5-classes task; [46] uses a 3M parameters on a 20-classes task. This call for an in-depth investigation of DL architectures taking into account a variety of datasets which is outside the scope of this work. Conversely, we take the reverse approach with respect to the literature and we rely on CNN-2 for the explicit purpose of investigating how much can be learned with a relatively small architecture.*

### B. Sensitivity to number of shots

We continue the analysis considering transfer learning, meta-learning, and contrastive learning approaches on one specific settings. Recall that for transfer learning  $\mathcal{T}_{source}$  contains all 320 for AppClassNet (*viz.* 24 for MIRAGE19) (Sec.II-B and Sec.V-A). We train  $M_{source}$  models for 200 epochs. For meta-learning, each epoch has 100 episodes with  $N=64$  ways for AppClassNet (*viz.* 16 for MIRAGE19), with  $Q=15$  queries and varying shots  $S \in \{5, 15, 50, 100, 200\}$  for both datasets. For all methods,  $\mathcal{T}_{target}$  is based on episodes of  $N = 4$  ways and  $Q = 15$  queries with the same configurations of  $S$  shots as in training. In each scenario we used a 5-fold cross validation for  $M_{source}$  models fine-tuning 1,000  $M_{target}$  models, each with 4 random classes from  $\mathcal{D}_{test}$ . Table V

<sup>9</sup>We control trees size by means of max-depth for simplicity and we acknowledge that also the estimators should be taken into account to make our assessment even more robust.

collects the average balanced accuracy and related 95th CI; we further visually pinpoints the best (green) and worst (red) method for each category when using 200 shots.

**Meta-learning.** First of all, we observe a sharp benefit when increasing the number of shots (Q2). Recall that the typical setup in computer vision literature limits  $S \in \{1, 5\}$  shots, and also TC literature related to FSL adopts constrained settings (see Table II).<sup>10</sup> Those regimes are the worst performing in our analysis with models trained on MIRAGE19 suffering a  $-7\%$  performance gap compared to AppClassNet models. In other words, a larger variety of classes/samples is beneficial, yet meta-training seems affected by (not obvious) “bottlenecks”.

When increasing the shots, beside the larger labeling effort, two more subtle effects start to appear. First, an increased computational resource cost—to go beyond 100 shots we had to split training across multiple GPUs, not for the models themselves, but due to the resource required to track gradients given the increased episode size. Second, the higher the number of shots, the more an episode resembles a “bloated” micro-batch of traditional monolithic training—an episode is deprived of its “synthetic random task” nature (see Sec.II-C). Overall, and differently from previous literature (see Table II), meta-learning methods fail to obtain solid generalization (Q2).

**Episodic transfer learning.** Conversely, transfer learning methods are the best performing (Q1). In particular, on AppClassNet all methods are within a  $\pm 1\%$  gap, while for MIRAGE19 we observe  $\pm 8.36\%$  between the best and worst performing methods with 200 shots. Recall that most of these methods differ mostly for the classifier  $f_{\varphi}(\cdot)$ . In particular, on AppClassNet we can rank their performance as *logistic regression*  $<$  *class embedding*  $<$  *linear layer*  $<$  *nearest neighbor*. However, on MIRAGE19 class embedding is the worst while nearest neighbor has  $+4\%$  gap compared to the 2nd performing classifier (logistic regression). This hints that the latent space representation learned on MIRAGE19 is worse than the one learned on AppClassNet despite the very different task complexity (32-vs-400 classes)—a nearest

<sup>10</sup>Except ISCX-VPN-nonVPN and USTC-TFC, popular datasets in TC (Table II) enjoy  $>1,000$  avg. samples/class (even when filtering flows with  $>10$  packets), but lack classes variety, thus more shots are viable.



TABLE VI: Transfer learning without episodic training.

Approach	MIRAGE19	AppClassNet	AppClassNet →MIRAGE19
Baseline-TL	86.62 ± 1.15	95.48 ± 0.78	78.69 ± 0.41
Baseline(ClassEmb)-TL	82.85 ± 1.54	94.72 ± 0.85	71.13 ± 0.45

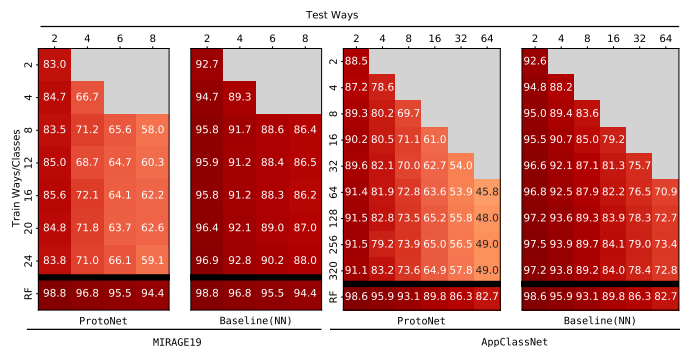
neighbor classifier is more flexible than a linear one, which possibly justifies the better performance on MIRAGE19 if classes are not well separated; yet, a nearest neighbour classifier needs to carry training data with the model in order to have labeled “anchors” to use for the classification. Unfortunately, investigating the latent space by means of silhouette score and similar clustering metrics did not provide conclusive answers about the discrepancies between the datasets. Lastly, while distillation can provide benefits on average, the difference with respect to other methods falls within the CI ranges, but we cannot completely discard those methods which were effective in computer vision literature [16].

**Contrastive learning.** As suspected, by leveraging label information, supervised contrastive learning outperforms the traditional version—e.g., up to +10.35% and 3.98% on MIRAGE19 and AppClassNet respectively (Q3). Moreover, while class embedding where ineffective for transfer learning, they are beneficial for contrastive learning. However, the negative side of those techniques is their computational cost: each  $\mathcal{T}_{source}$  model took  $\approx 1$  day of training! By investigating the code, we deemed those costs to the randomized data augmentation. Thus, we believe that proper code refactoring can further reap contrastive learning benefits.

**Transfer learning without episodes.** To complete the benchmark, Table VI reports the performance when using transfer learning without episodic training. We observe that for MIRAGE19 episodic training is more beneficial than for AppClassNet, even when compared against supervised contrastive learning. As before, we suspect these discrepancies relates to differences in the latent space geometry which are not so obvious to extrapolate.

Table VI also shows that using AppClassNet as source model to fine-tune task related to MIRAGE19 yielded poor performance. Recall that AppClassNet underwent an anonymization process modifying raw features values. Moreover, MIRAGE19 relates to mobile apps traffic while the (private) dataset from which AppClassNet was created relates to wired networks. Based on these observations, we find quite remarkable that this transfer still outperforms meta-learning methods for MIRAGE19.

**Takeaways.** *Differently from previous literature, the analysis show poor performance for meta-learning, and suggest to focus on transfer learning and contrastive learning (Q1-2). The latter of the two poses interesting engineering challenges and the positive results we gathered seem corroborating the idea of relying on self-supervision to further boost representation learning [21]. Notice also how the best DL method is on-par (or close to) the reference RF on 4 classes tasks—this is a positive signal of more general representations (Q1).*

Fig. 2: Sensitivity to train and test ways ( $S = 200$ ,  $Q = 15$ ).

### C. Sensitivity to the number of ways

Meta-learning methods evaluated in Table V were trained with a smaller  $N$ -way that for transfer learning. If on the one hand, each episode is less complex than for transfer learning, it is possible that transfer learning methods enjoy a more “regularized” latent space due to the higher number of classes pushing for more knowledge extraction. Thus, it is natural to wonder if the performance gap is due to the experimental settings. To investigate this, we run a second campaign varying both number of training and testing ways. For this analysis, we focused only on ProtoNet and Baseline(NN) as representative methods. We fixed  $S = 200$  shots and  $Q = 15$  queries while varying the number of train ways  $\{2, 4, 8, 12, 16, 20, 24\}$  and test ways  $\{2, 4, 8\}$  for MIRAGE19 ( $\{2, 4, 8, 16, 32, 64, 128, 256, 320\}$  and  $\{2, 4, 8, 16, 32, 64\}$  for AppClassNet). As before, we trained the models using 200 epochs of 100 episodes each. Fig. 2 heatmaps report the average balanced accuracy across 1,000 test episodes (CI are in line with previous experiments, hence not reported for brevity) as well as the reference RF performance.

Starting from columns values, performance improve when increasing the train ways—e.g., +4.6% for Baseline(NN) on AppClassNet when moving from 2 to 320 classes (160× more classes), yet only +0.4% when moving from 64 to 320 (5× more classes). The same is true for MIRAGE19 too but with milder benefits. The trend is reversed when considering rows values—e.g., on MIRAGE19, −8/9% performance drop for ProtoNet (−2/3% for Baseline(NN)) every time we double  $\mathcal{T}_{target}$  number of classes. Overall, all models are under-performing compared to RF models with unbounded depth. Yet, also RF performance decreases with similar gaps to DL when doubling the target task size.

**Takeaways.** *The analysis suggests that better generalization can indeed be achieved when using a larger pool of classes (Q1). However, this is just a first step in the right direction. Recall indeed that we used fixed representation according to traditional evaluation protocol for representation learning, and we intentionally fixed models architecture too—our results are likely lower bounds.*



## VII. DISCUSSION AND CONCLUSION

In this work, we compared transfer, meta- and contrastive learning against traditional ML tree-based models and DL monolithic training using two publicly available datasets with larger classes variety than in previous TC literature.

On the one hand, tree-based models achieve the highest performance. Although their size explodes for complex tasks, when dealing with  $\leq 10$  classes (and an average of 1,000 samples/class), they are still the most practical solution (Q2). On the other hand, compared with tree-based models, DL models handle more complex tasks, and their performance is almost on-par with tree-base models also for small tasks when using transfer learning methods—to the best of our knowledge, this has been overlooked in previous TC literature.

More important, results show that a large classes variety fosters reuse of DL models (Q1). In particular, and differently from previous TC literature, our results show that meta-learning methods are the worst performing methods (Q2), while transfer learning based on episodic fine-tuning is a better option—training with many samples (and many classes) is the best option to create  $M_{source}$  models; these can then be used to fine-tune  $M_{target}$  models using 100s samples. Moreover, data augmentations and supervised contrastive learning yield the best DL models overall (Q3).

We acknowledge also some limitations. For instance, we follow the traditional protocol of testing representations via fixed representation ( $M_{source}$  is frozen when transferred), and we also adopt on relatively small architecture compared to TC literature, i.e., our results are likely lower bounds. But, even in this constrained setting, supervised contrastive learning offers performance almost on-par with RandomForest (when training tasks with 4 classes) (Q3). We firmly believe that performance can improve via further optimizations (e.g., different architectures, input type, training methodology, and data augmentation policies). However, with this work, we aim to spark a “mind shift” in the TC research community to depart from simple tasks ( $\approx 10$  classes), and refocus on more challenging settings to push the learning. In particular, next to MIRAGE19 and AppClassNet, new large datasets have been recently released [47, 48] and more are likely to come: now we call to the research community to take them into action.

## REFERENCES

- [1] J. R uth, I. Poese, C. Dietzel, and O. Hohlfeld, “A first look at quic in the wild,” in *PAM*, 2018.
- [2] S. Roth, R. van Rijswijk-Deij, and C. Taejoong, “Tracking registrar support for dnssec,” *IMC*, 2019.
- [3] M. Trevisan, I. Drago, P. Schmitt, and F. Bronzino, “Measuring the performance of icloud private relay,” in *PAM*, 2023.
- [4] L. Yang, A. Finamore, F. Jun, and D. Rossi, “Deep learning and zero-day traffic classification: Lessons learned from a commercial-grade dataset,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4103–4118, 2021.
- [5] Y. Ouyang, B. Li, Q. Kong, H. Song, and T. Li, “FS-IDS: A Novel Few-Shot Learning Based Intrusion Detection System for SCADA Networks,” in *ICC*, 2021.
- [6] C. Rong, G. Gou, C. Hou, Z. Li, G. Xiong, and L. Guo, “UMVD-FSL: Unseen Malware Variants Detection Using Few-Shot Learning,” in *IJCNN*, 2021.
- [7] Y. Yu and N. Bian, “An intrusion detection method using few-shot learning,” *IEEE Access*, vol. 8, pp. 49 730–49 740, 2020.
- [8] W. Liang, Y. Hu, X. Zhou, Y. Pan, and K. I.-K. Wang, “Variational few-shot learning for microservice-oriented intrusion detection in distributed industrial iot,” *IEEE Transactions on Industrial Informatics*, vol. 18, no. 8, pp. 5087–5095, 2022.
- [9] W. Zheng, C. Gou, L. Yan, and S. Mo, “Learning to classify: A flow-based relation network for encrypted traffic classification,” in *WWW*, 2020.
- [10] C. Xu, J. Shen, and X. Du, “A method of few-shot network intrusion detection based on meta-learning framework,” *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3540–3552, 2020.
- [11] Z. Zhao, Y. Lai, Y. Wang, W. Jia, and H. He, “A few-shot learning based approach to iot traffic classification,” *IEEE Communications Letters*, vol. 26, no. 3, pp. 537–541, 2022.
- [12] T. Feng, Q. Qi, J. Wang, and J. Liao, “Few-shot class-adaptive anomaly detection with model-agnostic meta-learning,” in *IFIP Networking*, 2021.
- [13] E. Horowicz, T. Shapira, and Y. Shavitt, “A few shots traffic classification with mini-flowpic augmentations,” in *IMC*, 2022.
- [14] Z. Zhao, Y. Guo, J. H. Wang, H. Wang, C. Zhang, and C. An, “Cl-etc: A contrastive learning method for encrypted traffic classification,” in *IFIP Networking*, 2022, pp. 1–9.
- [15] W.-Y. Chen, Y.-C. Liu, Z. Kira, Y.-C. Wang, and J.-B. Huang, “A closer look at few-shot classification,” in *ICLR*, 2019.
- [16] Y. Tian, Y. Wang, D. Krishnan, J. B. Tenenbaum, and P. Isola, “Rethinking few-shot image classification: A good embedding is all you need?” in *ECCV*, 2020.
- [17] K. Weiss, T. M. Khoshgoftaar, and D. Wang, “A survey of transfer learning,” *Journal of Big Data*, vol. 3, 2016.
- [18] M. Crawshaw, “Multi-task learning with deep neural networks: A survey,” *arXiv preprint arXiv:2009.09796*, 2020.
- [19] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, “Revisiting unreasonable effectiveness of data in deep learning era,” in *ICML*, 2017.
- [20] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, “Generalizing from a few examples: A survey on few-shot learning,” vol. 53, no. 3, 2020.
- [21] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, “A simple framework for contrastive learning of visual representations,” in *ICML*, 2020.
- [22] A. v. d. Oord, Y. Li, and O. Vinyals, “Representation learning with contrastive predictive coding,” *arXiv preprint arXiv:1807.03748*, 2018.
- [23] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “Smote: synthetic minority over-sampling technique,” *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.
- [24] H. He, Y. Bai, E. A. Garcia, and S. Li, “ADASYN: Adaptive synthetic sampling approach for imbalanced learning,” in *IEEE international joint conference on neural networks*, 2008.
- [25] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, and J. Tang, “Self-supervised learning: Generative or contrastive,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 1, pp. 857–876, 2023.
- [26] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, “Supervised contrastive learning,” in *NeurIPS*, 2020.
- [27] J. Snell, K. Swersky, and R. Zemel, “Prototypical networks for few-shot learning,” in *NeurIPS*, 2017.
- [28] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales, “Learning to compare: Relation network for few-shot learning,” in *CVPR*, 2018.
- [29] C. Finn, P. Abbeel, and S. Levine, “Model-agnostic meta-learning for fast adaptation of deep networks,” in *ICML*, 2017.
- [30] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” in *NIPS*, 2015.
- [31] S. Rezaei and X. Liu, “Multitask learning for network traffic classification,” in *ICCCN*, 2020.
- [32] G. Sun, L. Liang, T. Chen, F. Xiao, and F. Lang, “Network traffic classification based on transfer learning,” *Computers & Electrical Engineering*, vol. 69, pp. 920–927, 2018.
- [33] W. Dai, Q. Yang, G.-R. Xue, and Y. Yu, “Boosting for transfer learning,” in *ICML*, 2007.
- [34] J.-B. Grill, F. Strub, F. Altch e, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, B. Piot, k. kavukcuoglu, R. Munos, and M. Valko, “Bootstrap your own latent - a new approach to self-supervised learning,” in *NeurIPS*, 2020.
- [35] S. Rezaei and X. Liu, “How to achieve high classification accuracy with just a few labels: A semi-supervised approach using sampled packets,”

in *ICDM*, 2019.

- [36] “Timeclr: A self-supervised contrastive learning framework for univariate time series representation,” *Knowledge-Based Systems*, vol. 245, p. 108606, 2022.
- [37] G. Aceto, D. Ciunzo, A. Montieri, V. Persico, and A. Pescapé, “Mirage: Mobile-app traffic capture and ground-truth creation,” in *ICCCS*, 2019.
- [38] C. Wang, A. Finamore, L. Yang, K. Fauvel, and D. Rossi, “Appclassnet: A commercial-grade dataset for application identification research,” *SIGCOMM Comput. Commun. Rev.*, vol. 52, no. 3, p. 19–27, sep 2022.
- [39] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, “Human-level concept learning through probabilistic program induction,” *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.
- [40] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra *et al.*, “Matching networks for one shot learning,” in *Neurips*, 2016.
- [41] L. Bertinetto, J. F. Henriques, P. Torr, and A. Vedaldi, “Meta-learning with differentiable closed-form solvers,” in *ICML*, 2019.
- [42] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, “Network traffic classifier with convolutional and recurrent neural networks for internet of things,” *IEEE Access*, vol. 5, pp. 18 042–18 050, 2017.
- [43] sklearn, “scikit-learn,” 2022. [Online]. Available: <https://scikit-learn.org/>
- [44] S. documentation, “Balanced accuracy score,” [https://scikit-learn.org/stable/modules/model\\_evaluation.html#balanced-accuracy-score](https://scikit-learn.org/stable/modules/model_evaluation.html#balanced-accuracy-score).
- [45] Z. Chen, K. He, J. Li, and Y. Geng, “Seq2img: A sequence-to-image based approach towards ip traffic classification using convolutional neural networks,” in *Big Data*, 2017.
- [46] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, “Malware traffic classification using convolutional neural network for representation learning,” in *ICOIN*, 2017.
- [47] J. Luxemburk and T. Čejka, “Fine-grained tls services classification with reject option,” *Computer Networks*, vol. 220, p. 109467, 2023.
- [48] J. Luxemburk, K. Hynek, T. Čejka, A. Lukačovič, and P. Šiška, “Cesnet-quick22: A large one-month quick network traffic dataset from backbone lines,” *Data in Brief*, vol. 46, p. 108888, 2023.