Replication: Contrastive Learning and Data Augmentation in Traffic Classification Using a Flowpic Input Representation

Alessandro Finamore
Huawei Technologies SASU, France
alessandro.finamore@huawei.com

Chao Wang
Huawei Technologies SASU, France
EURECOM, France
wang.chao3@huawei.com

Jonatan Krolikowski
Huawei Technologies SASU, France
jonatan.krolikowski@huawei.com

Jose M. Navarro
Huawei Technologies SASU, France
jose.manuel.navarro@huawei.com

Fuxing Chen
Huawei Technologies SASU, France
chenfuxing@huawei.com

Dario Rossi
Huawei Technologies SASU, France
dario.rossi@huawei.com

ABSTRACT

Over the last years we witnessed a renewed interest toward Traffic Classification (TC) captivated by the rise of Deep Learning (DL). Yet, the vast majority of TC literature lacks code artifacts, performance assessments across datasets and reference comparisons against Machine Learning (ML) methods. Among those works, a recent study from IMC'22 [16] is worth of attention since it adopts recent DL methodologies (namely, *few-shot* learning, *self-supervision* via *contrastive learning* and *data augmentation*) appealing for networking as they enable to learn from a few samples and transfer across datasets. The main result of [16] on the `UCDAVIS19`, `ISCX-VPN` and `ISCX-Tor` datasets is that, with such DL methodologies, 100 input samples are enough to achieve very high accuracy using an input representation called "flowpic" (i.e., a per-flow 2d histograms of the packets size evolution over time).

In this paper (i) we *reproduce* [16] on the same datasets and (ii) we *replicate* its most salient aspect (the importance of data augmentation) on three additional public datasets (`MIRAGE-19`, `MIRAGE-22` and `UTMOBILENET21`). While we confirm most of the original results, we also found a ≈20% accuracy drop on some of the investigated scenarios due to a data shift in the original dataset that we uncovered. Additionally, our study validates that the data augmentation strategies studied in [16] perform well on other datasets too. In the spirit of reproducibility and replicability we make all artifacts (code and data) available to the research community at https://tcbenchstack.github.io/tcbench/.

CCS CONCEPTS

• **Computing methodologies → Machine learning**; • **Networks → Network measurement**.

KEYWORDS

Traffic Classification, Deep Learning, Data augmentation, Contrastive learning.

ACM Reference Format:
Alessandro Finamore, Chao Wang, Jonatan Krolikowski, Jose M. Navarro, Fuxing Chen, and Dario Rossi. 2023. Replication: Contrastive Learning and Data Augmentation in Traffic Classification Using a Flowpic Input Representation. In *Proceedings of the 2023 ACM Internet Measurement Conference (IMC '23), October 24–26, 2023, Montreal, QC, Canada.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3618257.3624820

1 INTRODUCTION

Traffic classification (TC) is a long investigated topic in the networking community with seminal works dating back nearly two decades ago [27] which have been instrumental for bringing Machine Learning (ML) tools into networks operation and management. Since then, the TC field has been flourishing with literature and it is regularly surveyed [28, 30]. The recent hype of Deep Learning (DL) has expanded the interest on the field with several contributions from flagship ACM and IEEE conferences, including IMC [16].

Despite the progress made, reproducing (and replicating) research can still be a challenge [18], especially for TC. This is often rooted back in the well known self-awareness that "a scientific publication is not the scholarship itself, it is merely advertising of the scholarship" [5]. For the networking field, reproducibility has become more of a commonplace in the last decade, thanks to the emergence of tools (such as specialized containers [14]), community-wide awareness (such as dedicated workshops [34]) and policies (such as ACM badging [10]). Considering TC, difficulties are knowingly aggravated by data availability (yet, see Sec.2.3 for a positive outlook) and pertinence (i.e., due to datasets bias [19] and ageing, which mandates replication across several datasets).

Following a different direction than previous literature, this paper aims to *reproduce* and *replicate* research results from a recent TC study. In other words, our final goal is not only to investigate the scientific aspects behind the methodologies under study per-se, but to enable the research community to take advantage of our code base and artifacts such as models, logs, and curated datasets with a complementary website to document and navigate the artifacts (see App. B).

More in detail, we aim to reproduce the most important aspects of an interesting recent work on TC that appeared as a *short paper* in IMC'22 program [16]. This study uses recent promising DL techniques (notably, *few-shot* learning, *self-supervision* via *contrastive learning* and *data augmentation*) that we believe to be worthy of community-wide interest as they relate to practical problems that

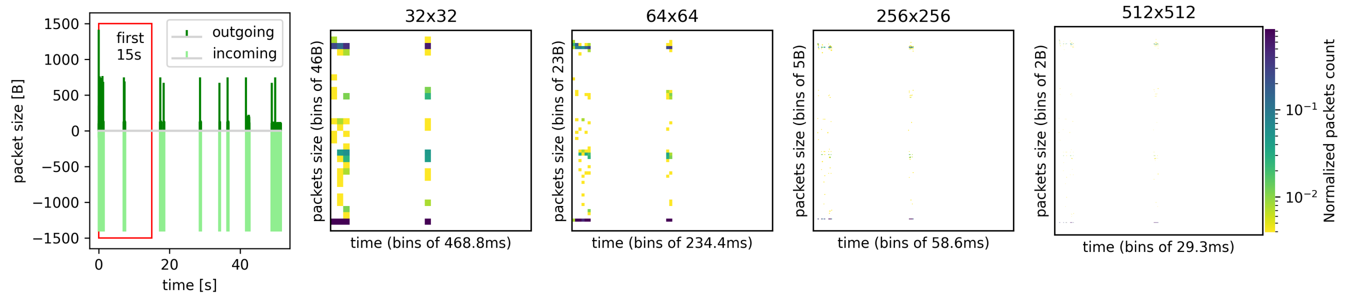


Figure 1: Example of a packet time series transformed into a flowpic representation for a randomly selected YouTube flow in the UCDAVIS19 dataset. Heatmaps are in a log scale normalized between the max and min value for each flowpic, with higher packets count values having darker shades (images better viewed digitally).

plague ML and DL application for TC (e.g., poor model generalization and label scarcity). Summarizing our main findings:

- Regarding *reproducibility*, from a *qualitative* viewpoint, we were able to reproduce most of the results of [16], e.g., confirming the interest in few-shot learning, self-supervision via contrastive loss and data augmentation; from a *quantitative* viewpoint, we incurred in unexpected results with large discrepancies that we were able to drill down and explain. Furthermore, with respect to the original publication, we calculated confidence intervals on our results to achieve better statistical validity, which may require tempering some of the observations in [16].
- Concerning *replicability*, we find qualitative agreement and confirm that the data augmentation policies selected in [16] behave consistently on other datasets.
- Philosophically, this work could be read as a chapter of the famous Queneau's book "Exercises in style"[31], cast to tell a traffic classification story. In particular, some aspects of the target paper (that we discuss in detail later) are missing despite being key for effectively reproducing the study.[1] Hence, readers must be aware that our resulting exercise is one of the many styles to tell the same story [31].

In the following, we first provide background about the target study (Sec. 2). We then lay out our replicability and reproducibility goals, providing information about artifacts (Sec. 3). We continue by discussing our experimental protocol and results (Sec. 4) before concluding with final remarks (Sec. 5). Details that we believe to be needed to make the paper self contained are deferred to App. C–F.

2 BACKGROUND AND MOTIVATION

Due to its nature, the present study falls in the broad area of reproducibility that started becoming a popular subject in networking a decade ago [14]. Given its breadth, it is out of the scope of this paper to review the whole reproducibility discipline. Conversely, we focus on the selected target paper, starting with an overview of its scope and contributions which we expand with relevant related work.

2.1 Target paper

In this work we replicate [16], which in the remainder of this paper we will also refer to as Ref-Paper or *Horowicz* et al. In this *short* study, *Horowicz* et al. quantify the performance of classification tasks when using only up to 100 training samples (i.e., *few shot learning* scenarios) yet increasing the training dataset with synthetic samples created with data augmentation functions. Authors consider both a supervised and an unsupervised setting, the latter represented by the popular *contrastive learning* approach named SimCLR [6], which starts from pre-training a model in an unsupervised fashion and later fine-tunes it to address a target task using a small number of labeled samples. Rather than using packet time series, *Horowicz* et al. use a *flowpic input representation*, i.e., a 2d summary of a network flow dynamics. Overall, we identify three contributions from the Ref-Paper: (*i*) a benchmark of the effect on the performance of Convolutional Neural Network (CNN) models across 7 *data augmentations* (Sec. 2.3), tested on two datasets, UCDAVIS19 and ISCX (Sec. 3.4); (*ii*) an evaluation of SimCLR (Sec. 2.4) and its sensitivity to the number of samples used during fine-tuning; (*iii*) an ablation of the fine-tuning performance when using alternative input formats to flowpic (Sec. 2.2). In the remainder of this section we expand on each of those contributions.

2.2 Input data representation

Target work. The flowpic representation used in the Ref- Paper was originally introduced by (part of) the same authors at an INFOCOM'19 workshop [33]. In Figure 1 we show a YouTube flow (extracted randomly from the UCDAVIS19 dataset) as well as its related flowpic at different resolutions. The left most plot shows the packet time series. Notice the expected bursty nature typical of video streaming services. The Ref-Paper computes a flowpic using only the first 15s of the time series. Specifically, both the 15s and the packets size range (0-1500) are split into bins based on the resolution of the target flowpic.[2] For instance a 32×32 flowpic leads to 469.8ms time bins and 46B packet size bins. Then, the count of

[1]We carried out our study based on what is reported in [16]. In fact, due to the double blind policy of the submission, we reached out multiple times to the original authors only in the weeks related to the shepherding of this article but we received only short and delayed responses. It is worth mentioning that we found a git repository [9] related to [16] but it presents several important limitations (see App. D) and cannot be used to replicate [16].

[2]Traffic directionality is not considered when composing the flowpic in the Ref-Paper although the representation could be reformulated to take it into account.

the packets occurring in each time window are tallied based on the defined packet size bins. In other words, each time window provides a frequency histogram of the packet sizes, and by vertically stacking all the histograms we obtain a "picture" of the flow dynamics. For instance, at the 32×32 resolution, the vertical stripes match the packet bursts of the original time series. This sort of patterns make the flowpic representation appealing for CNN-based DL architectures as convolutional layers are explicitly designed to extract features to detect such patterns. Yet, the higher the flowpic resolution, the sparser the representation, and the higher their computational process. While flowpic was introduced with a 1500×1500 resolution, in the Ref-Paper this is compared against a 32×32 resolution, i.e., a mini-flowpic.

Related work. Despite being well suited for CNN architectures, the flowpic representation is not a mainstream choice for TC as it requires to observe multiple seconds of traffic. This can enforce a late/post-mortem classification (i.e., after the flow ends), which, while still useful for monitoring, might not fit network management needs—prioritization, scheduling and shaping benefit from classification after the first few packets, so waiting for multiple seconds to take action can be sub-optimal. Conversely, the most common input features in the traffic classification literature are *packet time series* (e.g., the size, direction, inter-arrival time of the first 10 packets of a flow) and *payload bytes* (e.g., the first 784 L4 payload bytes). Since time series (and payload) features enable early classification, they have been the go-to choice since seminal works [4, 7, 26]. Additionally, time series and payload input can be combined in "multi modal" architectures [2, 3, 24] that have become popular in networking and other fields—rather than selecting either one representation or the other, a DL model can be designed to learn from different input formats at the same time. While we acknowledge it would be interesting to benchmark architectures and input representation, it is beyond the scope of our reproducibility/replicability study.

2.3 Label scarcity and data augmentation

Target work. Supervised learning requires large labeled datasets. As these are notoriously difficult to share and labeling is costly, the ability to learn from as few labeled samples as possible is particularly appealing.

In this direction, the Ref-Paper considered two mid-sized datasets—UCDAVIS19 [32], which contains 5 classes with up to ≈1,000 samples per class, and ISCX-VPN and ISCX-VPN [29], which were processed and combined to obtain 10 classes with a few flows each—and investigated the classification performance when *learning from only 100 samples.* To do so, *Horowicz* et al. considered *data augmentation* techniques applied to either flowpics (e.g., rotation) or to the packets time series (e.g., altering inter-arrival times) from which the flowpics are then computed. The Ref-Paper shows that simple data augmentations can indeed be beneficial even when using a few samples (with authors preferring time series transformations over image-related ones). These are interesting findings worth reproducing—on flowpic in the context of this work—and we believe they should be extended to packet time-series too in a future work.

Related work. Concerning *label scarcity*, until recent times only small-sized datasets for TC were publicly available—tens of classes and a few thousands of flows at best [29, 32]. This situation changed with the release of significantly larger datasets—hundreds of classes and millions of flows [23, 24, 36]. Whereas this encouraging trend is slowly making TC public datasets reach the scale of computer vision datasets, reducing dependency from labels is still a desirable goal and an open question for the ML research community as a whole.

Concerning *data augmentation* we find that, while it is widely adopted in computer vision (from the very first work at the root of the hype of CNNs in the field [21]), only a handful of studies use it in TC literature. Particularly relevant is [32] where the authors, beside introducing the UCDAVIS19 dataset, use as input packet time series, which are sampled into "subflows". Specifically, given the packet series (size, direction and inter arrival time), the authors propose to sample values based on different policies (e.g., selecting one packet every N from a random starting point); hence, from one flow they obtain multiple "subflows" which semantically correspond to a coarser-grained "view" of the original flow. The authors use UCDAVIS19 and ISCX-VPN with a two-step learning process: first, they *pre-train* a model in an unsupervised manner, targeting a 24-way regression problem from the generated subflows, i.e., they create a model that, given a subflow as input, can provide 24 metrics extracted from the flow; then such model is *fine-tuned* to obtain the target classifier using up to 20 labeled samples per class. In other words, [32] uses the two-step training approach of the Ref-Paper but does not adopt contrastive learning. Another recent study about data augmentation is [17] where the authors use a GAN-based approach that learns to augment packet time series during training. Conversely, in the Ref-Paper the augmentations are designed based on domain knowledge. Overall, while we believe that a broader and more systematic (i.e., across multiple datasets and inputs) comparison of data augmentation techniques in the TC field should be of community-wide interest, in our study we use the same point of view of the Ref-Paper.

2.4 Contrastive learning

Basic principles. Contrastive learning is particularly relevant and among the main reasons for our selection of the Ref-Paper. However, to understand this we first need to review some basic principles of DL model training.

At a high level, a supervised DL classifier is typically a composition of a feature extractor and a linear classifier. During training, the feature extractor learns how to project the input data into a latent space to group samples of the same class and distance them from other classes samples. Relying on such geometrical separations, a simple linear classifier suffices to identify classes but it is important to underline that such geometrical properties in the latent space are *implicitly learned*, i.e., the traditional training loss has no knowledge of the latent space as it observes points after the final linear layer, not in the latent space.

Conversely, contrastive learning is a special type of self-supervision which aims to *explicitly* enforce geometrical properties in the latent space by means of data augmentations. Figure 2 sketches the principles behind the technique. Input samples are transformed into

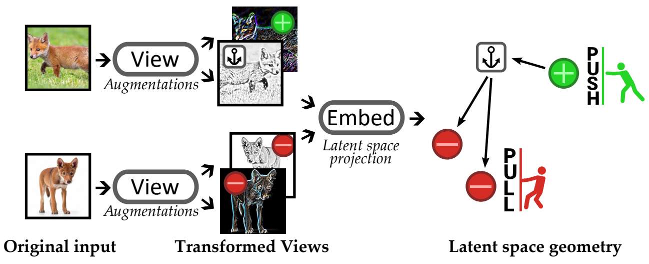


Figure 2: Contrastive learning principles.

"views" using augmentation functions. Then views are compared using a "contrastive game": a given view (an anchor) is compared in the latent space against other views of the same original image (positive samples) and all other views (negative samples). The contrastive loss function (namely InfoNCE for SimCLR) aims at pushing an anchor closer to its positives and distancing it from negatives. This training is *unsupervised* and a given anchor is forced to be similar to other views of the same image—in a sense, views of the same image form "their own class" even if negative samples can be of the same underlying class of the anchor. Hence, this is a harder problem than when using supervision and it is intentionally designed to push the learning of the representation. Figure 2 depicts one specific configuration of (anchor, positive, negatives) but during training all possible permutations are computed. The figure also shows the setting defined by SimCLR, but other variations are possible (e.g., the anchor can be the original image, and some contrastive learning algorithms do not use negative samples [11]). Once the feature extractor (a.k.a., the representation) is pre-trained, it can be extended by fine tuning a linear layer and obtaining the final classifier. Overall, the more powerful the representation, the lower the number of samples required for fine-tuning.

Target work. Contrastive learning's appeal comes from (1) the ability to pre-train a feature extractor in an unsupervised manner and (2) the possibility of fine-tuning with just a few labeled samples. Specifically, *Horowicz* et al. pre-train using 100 unlabeled samples transformed based on packet time series transformations (Change RTT and Time shift)[3] using SimCLR and fine-tune using only up to 10 labeled samples. Results show that, in some scenarios, classification performance is almost on-par with fully supervised training.

Related work. The closest related work to the Ref-Paper is [35], where the authors applied another off-the-shelf contrastive learning method (Bootstrap Your Own Latent - BYOL [11] which, unlike SimCLR, does not rely on negative samples) by pre-training on augmented data and fine-tuning on a few samples. The authors relied on the same dataset as in the Ref-Paper but adopted packet time series as their input (rather than flowpics), leveraging the data transformations proposed by [32] and a ResNet18 architecture. Overall, [35] shows comparable performance with respect to the Ref-Paper. A few more recent studies investigated contrastive learning on raw packet bytes as input [25, 37] and compared it against transfer learning and meta-learning [13], highlighting its recent relevance.

[3]The augmentations used in [16] are inspired by the ones used in [32].

3 METHODOLOGY

3.1 Experimental goals

As previously stated, this work aims (i) to *reproduce* the main results of the Ref-Paper, employing the same methodologies and data, as well as (ii) to *replicate* the Ref-Paper's results by considering three other datasets to confirm the validity of its findings. In more detail, our specific goals are:

- **(G0)** Provide a baseline analysis of the TC problem using classic Machine Learning (ML) models.
- **(G1)** Reproduce the benchmarking of the proposed data augmentation techniques for flowpics as a preliminary selection step for their use in contrastive learning. We aim for a quantitative reproduction **(G1.1)** and a qualitative reproduction **(G1.2)** of the augmentations ranks.
- **(G2)** Reproduce the SimCLR-related results with special attention to their internal details (e.g., use of dropout, projection layer size, training set size and impact of the combination of augmentations used).
- **(G3)** Replicate the benchmarking of data augmentation techniques on another set of public datasets.

By addressing **G0** we aim to quantify the classification task's degree of difficulty using ML methods and to assess if the use of more sophisticated modeling techniques is justified. Additionally, since the Ref-Paper does not provide confidence intervals nor perform any kind of statistical analysis of their results, **G1** and **G2** aim to not only merely reproduce the original results but, also, to add a layer of statistical significance to them. Finally, with **G3** we assess whether (statistically relevant) findings can be extended to other datasets.

3.2 Experimental protocol

We closely followed the configurations and scenarios from the Ref-Paper which we complemented with ablation studies and modeling campaigns. In this section we provide a summary of the main aspects of [16]. Details are deferred to the related evaluation sections where quote blocks as

> *example quote block*

highlight details from the Ref-Paper that require discussion.

DL architectures. We adopted the same CNN-based networks of the Ref-Paper, namely a LeNet5 [22] (i.e., a *mini*-flowpic) and a larger version of it (i.e., a *full*-flowpic).[4] We also explored the impact of dropout layers and the size of SimCRL projection layers (see 4.4.2). In addition to our code artifacts, the printout of the networks is reported in App. C.

Data augmentation. Next to applying no augmentation, we adopted the 6 augmentations used in the Ref-Paper—3 packet time series transformations (Change RTT, Time Shift and Packet Loss)

[4]The terminology in the Ref-Paper is overloaded as mini- and full-flowpic also refer to the resolution of the flowpic created.

Table 1: Outline of the Ref-Paper contributions, their points of improvement and our contributions.

Ref-Paper contributions	**Points of improvement**	**Our contribution**
Test on 7 data augmentations on `UCDAVIS19` and `ISCX`	Issues with `ISCX`; no statistical analysis of classif. perf.	Extra 3 datasets; added statistical analysis
Evaluate SimCLR and its sensitivity to fine-tuning dataset size	Lower performance with respect to the Ref-Paper	Expansion of training set size
Performance comparison on fine-tuning with alternative inputs	Analysis expansion to other parameters	Fine-tuning sensitivity to dropout and augmentation

and 3 image transformations (Rotation, Horizontal Flip, and Color jitter)—with the same hyper-parameters (see [16] for details).

Training steps. As in the Ref-Paper, we compared two DL modeling techniques: *fully supervised* training and *SimCLR + fewshot fine-tune* training. For the former, samples are augmented before starting the training. For the latter, given a labeled dataset and a selected augmentation function, each sample is processed to create 2 views of it using the Change RTT and Time shift transformations. Both views are created when forming the mini batches used during training. First, a representation of the dataset is learned by *pre-training* a model via SimCLR, contrasting pairs of augmented "views" of a sample. Then, a new model is formed by freezing the pre-trained representation and combining it with a classifier layer which is fine-tuned based on a few labeled samples. As in the Ref-Paper, we use Change RTT and Time Shift as data augmentation functions, yet we complement the analysis testing other augmentation pairs too. Augmentations are used only during pre-training.

Comparing contributions. To compare our study with [16], we provide a summary of the Ref-Paper contributions in Table 1, highlighting the points of improvement we identified (some of them described in the coming sections) and the actions we took to expand on them, apart from the basic reproducibility and replicability efforts described above. We remark that this table does not cover every contribution in this paper (e.g., it does not mention the added ML baseline), but rather contrasts what the original paper covered and how we increased the scope of the original contributions.

3.3 System and Artifacts

We performed 13 modeling campaigns, each consisting of the application of a target configuration across multiple random seeds and data splits (see Sec. 4 for details) for a total of 2,760 individual experiments. This entailed the implementation of a modeling framework able to properly track hyper-parameters, performance metrics and other configurations as well as output artifacts (e.g., models, summary report, logs). For this tracking we relied on `AimStack`[5] which we complemented based on our needs (e.g., the framework has only minimal support for tracking output files). Both our modeling framework and the whole modeling campaign outputs are provided as artifacts (see App. B).

All experiments were run on Linux servers equipped with multiple nVIDIA Tesla V100. Individual modeling experiment duration span from a few minutes to hours and the overall set of campaigns takes multiple weeks to run even in a distributed setting.

3.4 Datasets

To address our goals we used the four datasets summarized in Table 2. `UCDAVIS19` is used in the Ref-Paper while we selected the others because of their interesting and complementary properties with respect to `UCDAVIS19`: (*i*) they are collected in similar setups—research projects related to mobile traffic monitoring— (*ii*) they cover a larger number of classes and users behavior—`MIRAGE-19` and `UTMOBILENET21` are gathered from volunteering students interacting with instrumented phones while `MIRAGE-22` focuses on video meeting services—and (*iii*) they are imbalanced—the ρ values in the table reflect the ratio between the number of samples of the largest and smallest class in a dataset; notice the larger imbalance of the three datasets compared to `UCDAVIS19`, which is an expected property of network traffic. More importantly, all these datasets provide per-packet time series for the whole flows duration, which is a key requirement for composing flowpic representations. For instance, we cannot use the larger `AppClassNet` [36] and `CESNET-TLS` [24] datasets because they only provide the packet time series for the first 20-30 packets of each flow.

Table 2: Summary of datasets properties.

Name	Partition	Filter	Classes	Flows				Pkts
				all	*min*	*max*	ρ	*mean*
[32] `UCDAVIS19`	pretraining	*none*	5	6,439	592	1,915	3.2	6,653
	human			83	15	20	1.3	7,666
	script			150	30	30	1.0	7,131
[1] `MIRAGE-19`	*n.a.*	*none*	20$^{(*)}$	122,007	1,986	11,737	5.9	23
		>10pkts		64,172	1,013	7,505	7.4	17
[12] `MIRAGE-22`	*n.a.*	*none*	9	59,071	2,252	18,882	8.4	3,068
		>10pkts		26,773	970	4,437	4.6	6,598
		>1,000pkts		4,569	190	2,220	11.7	38,321
[15] `UTMOBILENET21`	*4-into-1*	*none*	17	34,378	159	5,591	35.2	664
		>10pkts	10	9,460	130	2,496	19.2	2,366

ρ : ratio between max and min number of flows—the larger the value, the higher the class imbalance; (*) Despite being advertised of having traffic from 40 apps, the *public* version of the dataset only contains 20 apps.

Data curation. Each dataset is a collection of files (in either CSV or JSON format) which we reprocessed into "monolithic" parquet files (a well known serialization format used in data science) encoding packet time series as `numpy` arrays.

As detailed in the table, `UCDAVIS19` is pre-partitioned (and pre-filtered) by the authors of the dataset to create a large set of samples for unsupervised training (namely pretraining) and two smaller testing set partitions, namely `script` and `human`.[6] As such, we found no need to alter the dataset beside the mere conversion to parquet.

Conversely, for the other three datasets we filtered out flows with less than 10 packets and removed classes with less than 100 samples. To replicate the setting provided in `UCDAVIS19`, for `MIRAGE-19` and

[5]https://github.com/aimhubio/aim

[6]According to [32], both pretraining and `script` correspond to automated collection of data, while `human` is gathered monitoring traffic when real users were interacting with the selected 5 services.

MIRAGE-22 we also first removed TCP ACK packets from time series and then discarded flows related to background traffic.[7] We also highlight that UTMOBILENET21 authors split the dataset into 4 partitions ("Action-Specific", "Deterministic Automated", "Randomized Automated" and "Wild Test") but we collated them into one.

The right-most column of the table details the average number of packets in a flow. Notice how UCDAVIS19 has very long flows while MIRAGE-19 is the dataset with shortest ones. To further focus on very long flows, we also created a version of MIRAGE-22 with flows having more than 1,000 packets.

Lastly, through our curation we also created reference train/test splits for the datasets. Specifically, since in the Ref-Paper the training dataset needs to have 100 samples, for UCDAVIS19 we create 5 folds (the smallest class in the dataset has 592 flows) of 100 samples per-class each. However, for the other datasets we opted for having 5 random splits each having a random selection of 80% of samples for training (and the rest for testing). To ease replicability, we contribute the code used for our curation (which can be applied directly on original version of each dataset) as well as our curated parquet files (see App. B).

Reproducibility. Beside UCDAVIS19, the Ref-Paper also considers the ISCX-VPN and ISCX-Tor datasets, but we discarded them after some preliminary investigations. In fact, as acknowledged by *Horowicz* et al. and as well known in the literature, these datasets (even when combined) contain only tens of viable flows for the analysis. Hence, to use them, one would need to create multiple 15s windows from the same flow to reach the 100 samples required for training, which seems artificious. More important, a recent work [19] carefully exposes fallacies for these datasets which are rooted in some form of data bias.[8] While underlining these issues, we do not want to discredit the datasets but rather to justify our choice of discarding them from our study.

Replicability. Quantitatively reproducing research results on a dataset is a necessary starting point but not be the ultimate goal. As we argued earlier, datasets age quickly in the TC field and new applications regularly emerge. Thus, replication on novel datasets is equally important. Qualitative agreement on a larger span of datasets brings the additional value of extending the validity of the findings. For these reasons, we employ MIRAGE-19, MIRAGE-22 and UTMOBILENET21 to replicate insights related to the comparison of data augmentation functions in the supervised setting.

4 EVALUATION

Unless differently stated, the results reported in this section are collected using the UCDAVIS19 dataset (training on the pretraining partition and testing of the two predefined human and script partitions).

[7]Traffic is collected on mobile phones with labeling ground-truth provided by netstat. One measurement experiment generates traffic logs for a specific target app. We processed such logs so that traffic of apps and services different from the target app (e.g., netd deamon, SSDP, Android gms) is removed as it represents "background" traffic.

[8]To be fair, the fallacies concern more the way the data bias can be unknowingly exploited to produced biased models. See [19] for more details.

Table 3: (G0) Baseline ML performance without augmentations in a supervised setting.

Input (size)	Model	Origin	Accuracy ± 95%CI	
			script	human
flowpic (32 × 32)	CNN LeNet5	[16]	98.67	92.40
flowpic (32 × 32)	XGBoost	ours	96.80±0.37	73.65±2.14
time series (3 × 10)	XGBoost	ours	94.53±0.56	66.91±1.40

Each *ours* is an aggregations of 15 experiments (5 splits × 3 seeds).

4.1 Providing a simple ML baseline (G0)

4.1.1 Approach. We start with an ML baseline to assess to what extent DL techniques are justified—which would be the case if we observe a large discrepancy between ML and DL performance.

We used a classic XGBoost as our ML model, with default hyperparameter values (100 estimators, max depth 6). As input, we compared a mini-flowpic (a 32×32 image flattened into a 1,024 values array) against the time series of the packet size, direction and intertime of the the first 10 packets of a flow (i.e., 3 features of 10 values each all concatenated into 30 elements arrays). We repeated the experiments 15 times and computed the 95% confidence intervals using a t distribution. Table 3 compares our results against those reported in the Ref-Paper for a LeNet5 CNN model trained without data augmentation (but no confidence intervals are available).

4.1.2 Results. The trained forests have very short trees (an average depth of 1.7 for time series and 1.3 for flowpic input). While trivial to execute, this analysis conveys interesting messages. For the script partition, (*i*) when using a flowpic representation, DL models have a slight advantage (about +2%) over ML models; (*ii*) the advantage of flowpic over a simple time series is more noticeable (about +4%), which could be expected since the amount of information in an early time series (a few packets) is significantly smaller than what encoded in a flowpic (multiple seconds of traffic).

Instead, a different interpretation arises when considering the human partition: (*i*) the results of ML are consistent with the observations in the script partition, i.e., using time series as input yields a score just a few percentage points lower than results using a flowpic input (6.74% difference on average); however, (*ii*) the gap between DL and ML models when using flowpic is unexpectedly large (18.75% on average).

Takeaway. *Based on the Ref-Paper results, our expectations were to have models offering similar performance on both testing partitions. Yet, we observed a large discrepancy for* human *which calls for a deeper analysis that we carry out in the following sections.*

4.2 Reproducing quantitative results of data augmentation (G1.1)

We continue by reproducing results related to Tables 1–2 of [16], which contrast different augmentations applied in a supervised setting.

4.2.1 Approach. Given the unexpected results of the ML baseline, we adopted a very careful approach, that we detail in what follows. *Horowicz* et al. wrote:

Table 4: Comparing data augmentation functions in a supervised training. Values marked as "ours" correspond to the average accuracy across 15 modeling experiments and the related 95-th confidence intervals.

	Test on `script`						**Test on** `human`						**Test on** `leftover` †		
	from [16]			*ours*			*from* [16]			*ours*			*ours*		
flowpic res	32	64	1500	32	64	1500	32	64	1500	32	64	1500	32	64	1500
No augmentation	98.67	99.10	96.22	95.64 ±0.37	95.87 ±0.29	94.93 ±0.72	92.40	85.60	73.30	68.84 ±1.45	69.08 ±1.35	69.32 ±1.63	95.78 ±0.29	96.09 ±0.38	95.79 ±0.51
Rotate	98.60	98.87	94.89	96.31 ±0.44	96.93 ±0.46	95.69 ±0.39	93.73	87.07	77.30	71.65 ±1.98	71.08 ±1.51	68.19 ±0.97	96.74 ±0.35	97.00 ±0.38	95.79 ±0.31
Horizontal flip	98.93	99.27	97.33	95.47 ±0.45	96.00 ±0.59	94.89 ±0.79	94.67	79.33	87.90	69.40 ±1.63	70.52 ±2.03	73.90 ±1.06	95.68 ±0.40	96.32 ±0.59	95.97 ±0.80
Color jitter	96.73	96.40	94.00	97.56 ±0.55	97.16 ±0.62	94.93 ±0.68	82.93	74.93	68.00	68.43 ±2.82	70.20 ±1.99	69.08 ±1.72	96.93 ±0.56	96.46 ±0.46	95.47 ±0.49
Packet loss	98.73	99.60	96.22	96.89 ±0.52	96.84 ±0.63	95.96 ±0.51	90.93	85.60	84.00	70.68 ±1.35	71.33 ±1.45	71.08 ±1.13	96.99 ±0.39	97.25 ±0.39	96.84 ±0.49
Time shift	99.13	99.53	97.56	96.71 ±0.60	97.16 ±0.49	96.89 ±0.27	92.80	87.33	77.30	70.36 ±1.63	71.89 ±1.59	71.08 ±1.33	97.02 ±0.50	97.51 ±0.46	97.67 ±0.29
Change RTT	99.40	100.00	98.44	97.29 ±0.35	97.02 ±0.46	96.93 ±0.31	96.40	88.60	90.70	70.76 ±1.99	71.49 ±1.59	71.97 ±1.08	98.38 ±0.18	97.97 ±0.39	98.19 ±0.22
mean diff ‡				-2.05	-2.26	-0.63				-21.96	-13.27	-9.13			

Each of *our* result is an aggregation of 15 experiments (5 splits × 3 seeds).
† We named "leftover" the samples from the *pretraining* partition not belonging to the 100 samples of a given split. Traditionally this would correspond to the test set.
‡ *mean diff* corresponds to the difference between our assessment and the expected value averaged across augmentations for each given flowpic resolution.

For all experiments, for training set we use only 100 "triggered by script" flows per class, and for test set we follow the experiments by [16] randomly choosing 30 flows for each class for a "triggered by script" test set and 15 flows per class for "triggered by human" test set. [...] For all experiments, we apply each of the augmentations 10 times on the 100 samples per class training set, which increase the training set to 1000 images per class. We also train without any augmentation as baseline experiments and term it "no aug". For all experiments we allocated 20% of the images for validation, and early stopped the training when the validation loss stopped improving

First of all, recall that `UCDAVIS19` is composed of three partitions explicitly named to express the intention of separating a portion of the data used for pre-training from another reserved for testing and fine-tuning (see Table 2). Although the authors use "triggered by script" twice, we interpreted that 100 flows are selected for training from the large pretraining partition, while using the remaining two partitions (`script` and `human`)[9] for testing and fine-tuning.[10]

Secondly, we did not find explicit mentions of how many experiments were performed to gather the results, nor do the tables report confidence intervals. Yet, we assume that several runs were carried out, as it is common practice when performing modeling campaigns to assess the performance across different dataset splits and models initialization.[11] Since the original experiments were done by training with 100 samples per-class (and the classes are imbalanced) doing a traditional k-folds cross validation is not possible. Thus, as from Sec. 3.4, we created k splits by sampling without replacement groups of 100 samples for each class from the pretraining partition. Then, a given set of 100 samples is split randomly s times, with each split corresponding to a 80/20 train/validation split for training. Using these data, we performed a campaign to test the 7 augmentations across k=5 splits each having s=3 train/-validation splits for a total of 105 experiments. This is repeated for the three flowpic resolutions with the same training settings as in the Ref-Paper: static learning rate at 0.001, early stopping on validation loss after 5 steps in which the loss does not improve by more than 0.001, batch size of 32, performance measured via accuracy, flowpic created from the first 15s of a flow.

[9]While `script` is perfectly balanced with 30 flows per class, `human` has three classes with 15 samples, and the remaining two have 18 and 20 samples respectively. Given the very small imbalance we considered irrelevant to resample the partitions to have exactly 15 samples per classes. Thus, we use `script` and `human` as is.
[10]Authors later clarified that they combined pretraining and `script`. However this minor difference does not affect the results of our investigation.
[11]Authors did not provide us more details on this aspect.

4.2.2 *Results.*

Table 4 summarizes our results reporting the mean accuracy and related 95% CI for each scenario. To ease their comparison, we copy the reference results from the Ref-Paper and summarize in the last row the differences across scenarios with a simple arithmetic mean. We complement the evaluation of the Ref-Paper by reporting a new test set corresponding to all pretraining samples not belonging to a selected 100 samples split (i.e., what would be called a test set in a traditional evaluation). As, to the best of our understanding, these samples have been discarded in the Ref-Paper, we refer to this test set as `leftover`.

Overall, we obtained lower performance than what was previously reported. While differences are modest on `script`, we observe a reduction of over 20% on `human`—this is coherent with what we observed for the ML baseline. Notice that no gap appears when comparing `script` with `leftover`.

The gap is (slightly) reduced when using a higher resolution flowpic but the lower performance on `human` (and the larger confidence intervals with respect to `script` and `leftover`) suggests the presence of a hidden problem with this predefined test set. Understanding the reason of this gap is important to verify the validity of our study. However, we defer a significant portion of our study of the performance gap to App. D.1–D.3 and we report only the salient aspects of our investigation in the following sections.

We highlight that, while for 32×32 and 64×64 experiments run in about 1 min, it takes about 30min to run one experiment on 1500×1500. Given this computational cost, motivated by the marginal performance gap across resolutions and as done by *Horowicz* et al., in the remainder of the paper we focus only on the 32×32 resolution.

4.2.3 *Root cause of performance gap.*

We reiterate that we do not apply any pre-processing (e.g., filtering, reshaping) to the `UCDAVIS19` dataset beside consolidating the original CSV files (one for each flow) into a monolithic parquet file. Thus, we conjectured that the root case of the performance problem might be rooted in the data itself.

To start verifying this assumption, the heatmaps in Fig. 3 break down the results in Table 4 by showing the average per-class accuracy across the 105 runs for the 32x32 flowpic resolution. Specifically, we summed all the confusion matrices for `script` and `human`

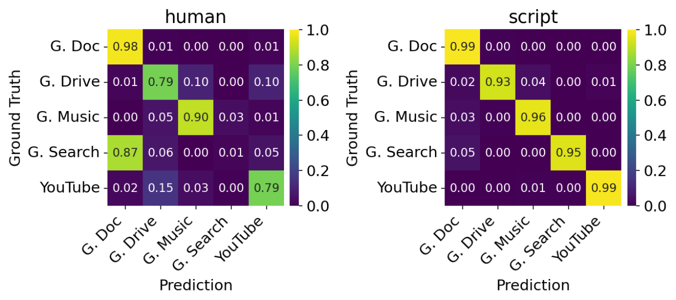


Figure 3: Average confusion matrixes for the 32×32 resolution across all experiments in Table 4.

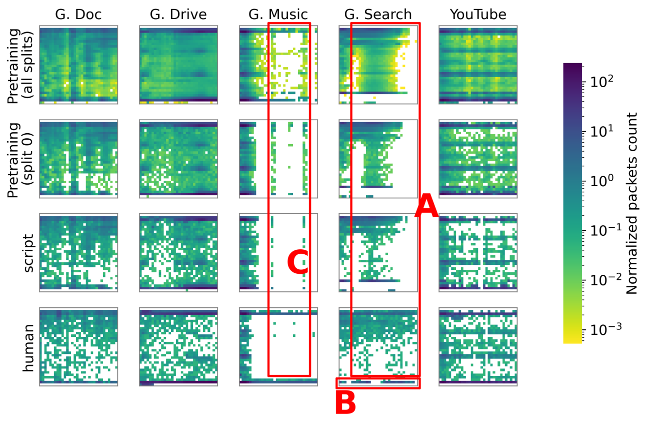


Figure 4: Average 32×32 flowpic for each class across dataset partitions.

and we normalized them by row. For human we observe multiple sources of confusion with *Google doc* and *Google search* having the most evident clash. Conversely, no specific issues can be detected for `script`.

To drill down, Fig. 4 collects an average flowpic per class across the original dataset partitions and one training split. Recall that the horizontal axis of a flowpic corresponds to time (time zero on the left) while the vertical axis corresponds to packet sizes (zero length on the top).

The first row in Fig. 4 corresponds to all flows available in the pretraining partition, while the second one corresponds to a training split, i.e., an aggregation of 100 samples per class. We can clearly see that the reduction of samples has a visual impact, but overall the first two rows are visually very similar. The third and the fourth rows correspond to the `script` and human partition respectively, i.e., they have 30 and ≈15 samples per class. When comparing the last two rows with the first two, we can clearly see differences which we further annotate with rectangles. Notice how *Google search* is expected to have two vertical groups of pixels around the left-axis and the center of the picture. Surprisingly, for human these groups are "shifted" to the right (rectangle A). Moreover, notice how all splits but human saturate the maximum packet size for *Google search*—there is a distinctive horizontal line (around pixels on row 28) for human (rectangle B) while in the other cases there are distinct dark lines at row 32. Interestingly, Fig. 4 also highlights macroscopic differences for *Google music*—vertical "stripes" of pixels are visible in all splits but human (rectangle C). Yet, according to Fig. 3, this seems less of a problem. We conjecture that this might be due to the stark difference between *Google music* and the other services. In other words, despite the different behavior between the partitions, *Google music* is still very different from the other 4 classes (thus it might be easier to classify).

The analysis of the average flowpics supports the idea of a data shift, of which we provide further evidence in the Appendix. Specifically, we support this statement by (*i*) adding more evaluations on `UCDAVIS19` (App. D.1), (*ii*) resorting to content from [32] which introduced the `UCDAVIS19` dataset (App. D.2), and (*iii*) verifying code artifacts from [32] to help us rule out possible mistakes in our approach (App. D.3). Summarizing this extensive material, the existence of a data shift is pointed out by both (*i*) and (*ii*) and we confirm (*iii*) as our verification yields expected results.

Takeaway*. Given the strong evidence provided by our analysis, we concluded that the* human *test split is affected by a data-shift. Yet, we cannot comment on the reason why this was not detected in the* R*EF*-P*APER.*[12]

4.3 Reproducing qualitative ranking of data augmentation (G1.2)

4.3.1 Approach. The original key question behind benchmarking the different augmentations was to understand if, and by how much, they were beneficial with respect to not performing any augmentation. *Horowicz* et al. wrote

> *In all the nine experiments changing the RTT was the best performing augmentation. The improvement varies from 1% for the QUIC script dataset (where the "no aug" accuracy was already 98.7%) up to 17.4% improvement for the most challenging dataset, the QUIC human.*

Without more details on the REF-PAPER it is very difficult to compare against the reported results. We opted instead for performing a statistical analysis of our modeling campaign to understand if Change RTT and Time shift were the best performing augmentations as reported in the REF-PAPER.

The CI values in Table 4 show clear overlaps between different augmentations. To investigate our results, we treat each augmentation as a different classifier and compare them according to the procedures presented in [8]. First, accuracy results are turned into rankings (e.g., if augmentations A, B and C yield an accuracy of 0.9, 0.7 and 0.8, their associated rankings would be 1, 3, and 2) with ties being assigned with the average ranking of the group (e.g., if augmentations A, B and C yield 0.9, 0.9 and 0.8, their associated rankings would be 1.5, 1.5 and 3). This process is repeated across all tested datasets and splits. Then, an average ranking value is extracted per augmentation. These values are compared pairwise using a post-hoc Nemenyi test, which compares these average rankings to decide if the performance difference between augmentations is significant. This decision is made using a Critical Distance (CD) in ranking equal to $CD = q_\alpha\sqrt{\frac{k(k+1)}{6N}}$, where q_α is based on the

[12]Authors did not provide us comments about this aspect.

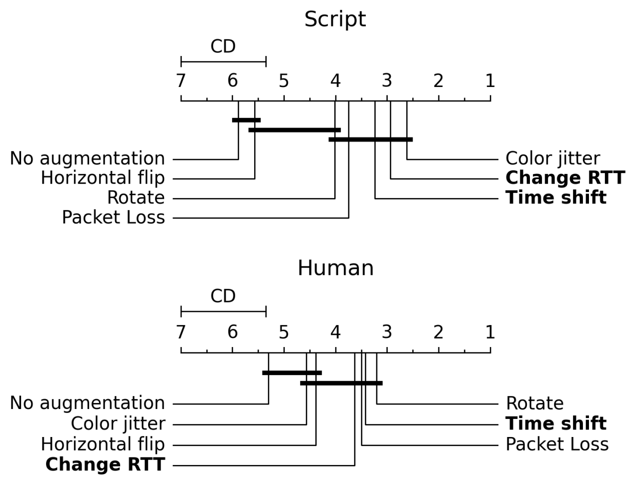


Figure 5: Critical distance plot of the accuracy obtained with each augmentation for the 32×32 and 64×64 resolutions. Augmentations joined by a horizontal line are not statistically different. The lower the ranking (closer to 1, the right side of the plot) the better the performance. Transformations highlighted in bold are selected as the best performing one in the REF-PAPER.

Studentized range statistic divided by $\sqrt{2}$, k is equal to the number of augmentations compared and N is equal to the number of samples used.

4.3.2 Results. Figure 5 displays the results of these comparisons. We combined the 32×32 and 64×64 resolutions as we did not find statistically significant differences between them (see App. F). In our case, with $\alpha = 0.05$, $k = 7$ and $N = 30$ and $q_{0.05} = 2.949$ the critical distance is $CD = 1.644$. The closer an augmentation is to the right side of the plot (a higher average rank), the better the performance.

From our analysis for the `script` partition, we cannot conclude significant differences within three groups, which we sort by increasing performance: {No augmentation and Horizontal flip}; {Horizontal flip and Rotate}; {Rotate, Packet loss, Time shift, Change RTT and Color jitter}. Similar groups exist also for the `human` partition. As annotated in Fig. 5, *Horowicz* et al. selected **Change RTT** and **Time shift** as the best augmentations: whereas these augmentations are in the best performing group both for `script` and `human`, it is easy to gather that other transformations consistently appear in the same (statistically relevant) group.

Takeaway*. On the one hand, the Time shift and Change RTT transformations are in the best performing group, a finding aligned with the ones in the REF-PAPER. On the other hand, from a statistical viewpoint, they are not distinguishable from other options, like Color jitter (for* `script`*) or Rotate (for* `human`*) or Packet Loss (for both).*

4.4 Reproducing constrastive learning results (G2)

4.4.1 Approach. The second goal of our reproducibility study concerns the use of contrastive learning and fine-tuning. A few observations are needed to contextualize the modeling campaign to perform.

Augmentations for SimCLR. First of all, we need to select augmentations for SimCLR. *Horowicz* et al. wrote:

> *we selected to use 'Change RTT' by $\alpha \sim U$ [0.5, 1.5] together with Time Shift by $b \sim U$ [-1,1]. In each training step, a double batch of 32 unlabeled images (taken from the pool of 100 unlabelled samples per class) is loaded after applying the two augmentations above.*

This confirms the traditional SimCLR approach where two views are obtained from each sample in a training mini-batch. However, precisely how the transformations are applied is open to interpretation, e.g., one after the other? If so, in which order? A separate transformation for each view? These are design choices likely depending on the task at hand. For instance, the original SimCLR paper [6] shows that both which transformations are selected and the order in which they are chained are relevant decisions. Since a full ablation study on this aspect is well beyond our scope, we opted for *applying the two transformations in random order for every image in a mini-batch.* Yet, given our ranking analysis showed equivalence among multiple top performing transformations, we also perform a small-scale ablation study considering three other pairs beside the pair selected in the REF-PAPER.[13]

Networks for SimCLR. Even more subtle design choices relate to the application of dropout and the projection layer size used in SimCLR. *Horowicz* et al. wrote:

> *As depicted in Figures 6 and 7, our architectures comprise seven layers, the ReLU activation function is applied to the output of every convectional and fully-connected layer and dropout with probabilities of 0.25 and 0.5 are used in order to reduce overfitting. [...]*

The figures mentioned refer to the "mini" (for 32×32 and 64×64) and "full" (for 1500×1500) architectures. We underline that the mini architecture is identical to the original LeNet5 [22]. However, from the quote we identify a number of layers miscount, as the full version has one layer less than the mini version. In fact, the "flatten" layer is reported only in the full diagram but is actually needed in both versions, and the diagram clearly shows that the full version has one less fully connected layer than the mini version (see Fig. 6-7 in [16]). Moreover, based on the quote it is not clear if dropout was applied to both architectures or just the full version (as the original LeNet5 does not rely on dropout). Given the lower resolution, dropout might be not needed for 32×32.

Regarding the SimCLR projection, *Horowicz* et al. wrote:

> *For the representation extractor $f(\cdot)$ we employed the 5 first layers of the CNN architectures described in A.1 and replaced the last 2 layers with 2 linear layers sized 120 and 30. Thus, resulting with a 120 dimensional representation vector $h = f(flowpic)$ and $z = g(h)$ dimensional similarity vector.*

[13] *Horowicz* et al. clarified with us that the two transformation were chained and to check their repository [9]. Yet, we reiterate that such repository cannot be used to reproduce the results of the REF-PAPER (see App D).

Table 5: Impact of dropout and SimCLR projection layer dimension on fine-tuning (32×32 only, with 10 samples for fine-tuning training).

	test on `script`		*test on* `human`	
Proj. dim	w/ dropout	w/o dropout	w/ dropout	w/o dropout
30	91.81 ±0.38†	92.18 ±0.31	72.12 ±1.37‡	74.69 ±1.13
84	92.02 ±0.36	92.54 ±0.33	73.31 ±1.04	74.35 ±1.38

Each value is an aggr. of 125 exp. (5 splits × 5 SimCRL seeds × 5 fine-tune seeds). The reference value for † from [16] reports in the text (94.5% for 10 samples); for ‡ no specific values are reported but should be ≈80% based on Fig. 4 of [16].

This refers to what is known as the *projection layer* of the feature extractor. In a nutshell, and based on our interpretation of the quote, after the convolutional blocks, the network have a 120-120-30 series of linear layers. However, since the supervised network was using a latent space of size 84, we investigated networks considering both 30 and 84 as final projection layer dimension.

An assessment of these lower level details can be key to obtain a fair comparison against the performance reported in the Ref-Paper. For reference, we report the listing of the architecture used in App. C.

4.4.2 Results. As before, we follow the parameters described in the Ref-Paper, namely batch size of 32, patience of 3 on the top-5 accuracy when training with SimCLR (temperature=0.07, learning rate=0.001) and patience of 5 on train (min delta=0.001) during fine-tuning (learning rate=0.01).

Table 5 details the results of our ablation campaign to understand the impact of dropout and the projection layer.[14] Each value in the table corresponds to the mean and related 95-th percentiles CI across 125 experiments and fine-tuning using 10 training samples. As we expected, we observe poorer performance when testing on `human`, while performance on `script` is just a few points lower than for supervised training. When considering a projection layer of 30 units, we can observe that dropout does not provide a significant difference for `script`; conversely, removing dropout makes a stark difference when testing on `human`. Increasing the projection layer dimension does not provide a significant gain. We conclude than that we can rely on a network *without* dropout (differently from the Ref-Paper) but we confirm the original choice of a projection layer of 30 units.

In Table 5, we also annotate the configuration that (we believe) was used in the Ref-Paper. Specifically, the study reported results (only as figures) characterising performance improvement when increasing the number of samples for fine-tune training, and concluded that the best performance was achieved when using 10 training samples, i.e., the scenario we selected for our evaluation. Yet, while for `script` *Horowicz* et al. wrote:

> Our method achieves 93.4% accuracy with only 3 samples, and 94.5% with 10 samples

no specific values are reported for `human`. However, Figure 4 of the paper clearly shows an accuracy of about 80%.

[14]We did also an ablation of dropout before running results for Table 4. Details are reported in Appendix E. The takeaway is that even for 1500×1500 resolution there are minimal differences introduced by dropout. Yet, results in Table 4 reflect the use of dropout as intended in the original study.

Table 6: Comparing the fine-tuning performance when using different pairs of augmentations for pretraining (32×32 resolution, fine-tuning on 10 samples only).

1st augment.	Change RTT*	Packet loss		Change RTT		Color Jitter
2nd augment.	Time shift*	Color jitter	Rotate	Color Jitter	Rotate	Rotate
test on `script`	92.18±0.31	90.17±0.41	91.94±0.30	91.72±0.36	**92.38**±0.32	91.79±0.34
test on `human`	74.69±1.13	73.67±1.24	71.22±1.20	**75.56**±1.23	74.33±1.26	71.64±1.23

Each value is an aggreg. of 125 exp. (5 splits × 5 SimCLR seeds × 5 fine-tune seeds). (*) pair of augmentations used in [16].

Table 7: Accuracy on 32×32 flowpic when enlarging training set (without dropout).

		`script`	`human`
Supervised	No augmentation	98.37±0.19	72.95±0.96
	Rotate	98.47±0.25	73.73±1.09
	Horizontal flip	98.20±0.15	74.58±1.16
	Color jitter	98.63±0.21	72.47±1.02
	Packet loss	98.63±0.19	73.43±1.25
	Time shift	98.60±0.22	73.25±1.17
	Change RTT	98.33±0.16	72.47±1.04
	SimCLR + fine-tuning	93.90±0.74	80.45±2.37

Each value is an aggregation of 20 experiments (20 different seeds)

While performance are basically on par for `script`, our results for `human` are significantly lower than the previous evaluation. We additionally observe that, for contrastive learning with fine-tuning, a drop of performance from `script` to `human` is also reported in the Ref-Paper—unlike for supervised training as discussed earlier. While the training methodologies are fundamentally different, the underlying dataset and testing methodology are the same (training with the pretraining partition and testing on `script` and `human`). Thus, the consistency between our ML, supervised and contrastive learning campaigns is to be expected, but we cannot comment on why *Horowicz* et al. observed the `script`-vs-`human` gap only for the contrastive learning experiments.[15]

Takeaway*. On the one hand, results are consistent and quantitatively aligned for* `script`*, which confirms the interest for few shot contrastive learning and data augmentation. On the other hand, results for* `human` *are only qualitatively in agreement, which calls for agreeing on a community-wide standard benchmark including multiple datasets.*

4.4.3 Extra results. We conclude our analysis by reporting two complementary analysis with respect to the Ref-Paper. First, we investigated to which extent alternative pairs of augmentations affect the fine-tuning performance. Namely, we considered *Time shift* and *Change RTT* next to *Rotate* and *Color jitter*, selected because they achieved good positions in our ranking analysis. Then we formed groups by either pairing time series with image transformations or pairing the image transformations. Results collected in Table 6 show that, despite the punctual differences between pairs, our observation on Table 4 and the ranking analysis (Sec 4.3) still holds—all pairs are *qualitatively* equivalent.

[15]Authors did not provide more comments to us about this aspect.

Table 8: (G3) Data augmentation in supervised setting on other datasets. The top two transformation strategies for each datasets are in bold for visual purposes (not to imply statistically relevant conclusions).

Augmentation	MIRAGE-22 (≥10pkts)	MIRAGE-22 (≥1000pkts)	UTMOBILENET21 (>10pkts)	MIRAGE-19 (>10pkts)
No augmentation	90.97 ±1.15	83.35 ±3.13	79.82 ±1.53	69.91 ±1.57
Rotate	88.25 ±1.20	**87.32** ±2.24	79.45 ±1.28	60.35 ±1.17
Horizontal flip	91.90 ±0.84	83.82 ±2.26	80.03 ±1.33	69.78 ±1.28
Color jitter	89.77 ±1.16	81.40 ±3.62	78.68 ±2.14	67.00 ±1.11
Packet loss	92.34 ±1.10	87.19 ±2.52	72.07 ±1.73	67.55 ±1.46
Time shift	**92.80** ±1.21	86.73 ±3.88	**81.91** ±2.12	**70.33** ±1.26
Change RTT	**93.75** ±0.83	**91.48** ±2.12	**81.32** ±1.54	**74.28** ±1.22

Each value is aggregation of 15 experiments (5 splits × 3 seeds).

Second, we expand the methodology used so far by quantifying the effect of using a (pre)training set larger than 100 samples. Specifically, we created 5 random 80/20 train/validation split using the full pretraining partition, i.e., the dataset result imbalanced with up to 1,532 training samples for the largest class and 473 for the smallest. Table 7 reports the results of the modeling campaign in both a supervised and contrastive learning settings. As expected, compared to Table 4 and Table 5, enlarging the dataset is effective in improving performance in both settings. In particular, for contrastive learning the gain is smaller for `script` (+1.72% on average) than for `human` (+5.76% on average)—the latent space created via contrastive learning is better at mitigating the data shift.

Takeaway. *The transformations selected in the* Ref-Paper *constitute a good enough choice, although image transformations cannot be fully ruled out based on our assessment. This confirms that identifying the most suitable transformations is tied to the input representation and datasets used, which remains an open problem. Moreover, while a very limited number of samples can be enough for training models, the same scenarios can benefit from more data—the selected augmentations alone are not a final replacement for real input samples.*

4.5 Replicating data augmentation on other datasets (G3)

4.5.1 Approach. Given that we observed only small performance differences among the augmentations, we extended the Ref-Paper by replicating the analysis using other three datasets, namely `MIRAGE-19`, `MIRAGE-22` and `UTMOBILENET21`. Based on the results displayed in Table 7, we opted for a traditional 80/10/10 train/validation/test using all samples available for each class, i.e., we removed the constraint of using 100 samples per class as in Table 4. This is a compromise dictated by the differences among the datasets. In particular, as shown in Table 2, the filtering significantly reduces the number of samples per class, especially for the smallest class. Hence, rather than removing the very small classes, we preferred to use a split preserving the original imbalance of the data. We argue that this is reasonable considering that the question we were targeting was about the *importance of the augmentation functions* which is per-se to be decoupled from datasets samples count. Moreover, we restricted our analysis to the supervised scenario only. It

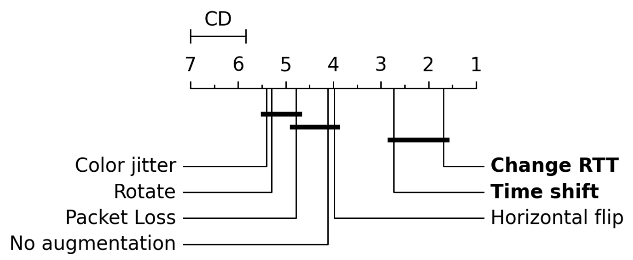


Figure 6: Critical distance plot of the accuracy obtained with each augmentation across the four tested datasets.

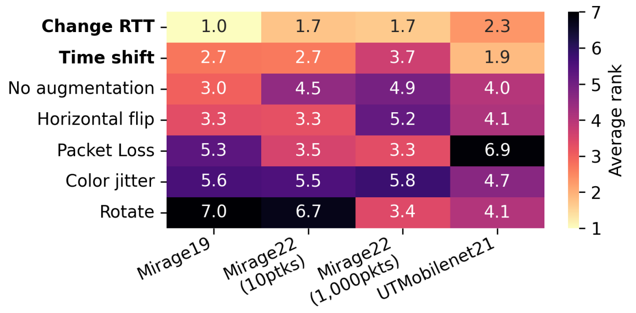


Figure 7: Average rank obtained per augmentation and dataset. Ranks closer to 1 indicate a better performance.

follows that our analysis can be considered as an *upper bound* of what can be achieved when considering less training data and/or via contrastive learning. Since the training and testing datasets are imbalanced in this scenario, we measure performance via an F1 score (rather than using accuracy as done before).

4.5.2 Results. For each dataset we used the architectures and settings as in Sec. 3.2. Table 8 and Figures 6-7 collect our results. Extending the analysis to more datasets allows us to better appreciate differences between the impact of each augmentation. First of all, while the maximum gap between augmentations in Table 4 is (on average) 3.22%, this is now 13.93% (occurring for `MIRAGE-19`). Despite the larger differences, the analysis confirms Change RTT and Time shift as the best performing augmentations across all datasets. Differently from the previous analysis, Fig. 6-7 highlight how the two functions are significantly better than the others, yet still not statistically different from each other.

Takeaway. *Our results confirm the benefit of data augmentations and validate the selection of Change RTT and Time shift as in the* Ref-Paper*.*

5 CONCLUSIONS

In this paper we reproduced and replicated the methodology of [16] which investigated noteworthy DL methodologies (few-shot learning, self-supervision via contrastive learning and data augmentation) on TC. These methods are particularly appealing as they allow for learning from a few samples and transferring models across datasets.

Summarizing our analysis, we have been able to *qualitatively* reproduce most of the original results, so we confirm the interest in few-shot contrastive learning and data augmentation. At the same time, our modeling campaigns found unexpected *quantitative* discrepancies that we rooted in data shifts in the UCDAVIS19 dataset (undetected in the Ref-Paper).

Another remarkable consideration can be gathered by contrasting our reproducibility vs replicability results. Indeed, the reproducibility results on UCDAVIS19 show little statistical significance in the differences among the proposed data augmentation techniques—just by reproducing the study on UCDAVIS19 alone would therefore have not allowed us to validate *Horowicz* et al.'s choices. Conversely, by replicating the methodology on three additional datasets, we gathered evidence that finally validated Change RTT and Time Shift as more beneficial than other augmentations for the flowpic input representation.

We also acknowledge some limitations in our replication. For instance, while we studied augmentations in a supervised setting, we leave as future work their assessment in a contrastive learning setting paired with few shot fine-tuning. Indeed, such a study should consider the variety of contrastive learning approaches including *supervised* contrastive learning methods such as SupCon [20].

Lastly, in the context of network ML studies, we underline the need to agree on a broader set of benchmarks as other communities (e.g., CV and NLP) are doing more systematically, which can only improve the quality of the gathered knowledge. To support this future direction, we make available multiple artifacts in the form of code (besides our modeling framework, we contribute scripts related to the modeling campaign and all post-processing to generate reports and figures inhere contained) and data (both trained models and related logs, as well as the dataset splits used for training and testing). As described in App. B, artifacts are also complemented by a website providing documentation (e.g., guides on how to run the experiments, stats about the datasets). We believe that TC is in need of a reference framework binding datasets with modeling tools. We hope the research community can take advantage of our work and/or be inspired toward improving current practices.

REFERENCES

[1] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, Valerio Persico, and Antonio Pescapè. 2019. MIRAGE: Mobile-app Traffic Capture and Ground-truth Creation. In *IEEE 4th International Conference on Computing, Communication and Security (ICCCS 2019)*.

[2] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapè. 2019. MIMETIC: Mobile encrypted traffic classification using multimodal deep learning. *Computer Networks* 165 (2019), 106944.

[3] Iman Akbari, Mohammad A. Salahuddin, Leni Ven, Noura Limam, Raouf Boutaba, Bertrand Mathieu, Stephanie Moteau, and Stephane Tuffin. 2021. A Look Behind the Curtain: Traffic Classification in an Increasingly Encrypted Web. *Proc. ACM Meas. Anal. Comput. Syst.* 5, 1, Article 04 (feb 2021), 26 pages.

[4] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. 2006. Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review* 36, 2 (2006), 23–26.

[5] Jonathan B Buckheit and David L Donoho. 1995. *Wavelab and reproducible research*. Springer.

[6] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A Simple Framework for Contrastive Learning of Visual Representations. *arXiv preprint arXiv:2002.05709* (2020).

[7] Manuel Crotti, Maurizio Dusi, Francesco Gringoli, and Luca Salgarelli. 2007. Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM Computer Communication Review* 37, 1 (2007), 5–16.

[8] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research* 7 (2006), 1–30.

[9] eyalho. 2022. mini-flowpic-traffic-classification. https://github.com/eyalho/mini-flowpic-traffic-classification?.

[10] Association for Computing Machinery (ACM). 2023. Artifact Review and Badging Version 1.1. https://www.acm.org/publications/policies/artifact-review-and-badging-current.

[11] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, Bilal Piot, koray kavukcuoglu, Remi Munos, and Michal Valko. 2020. Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 21271–21284.

[12] Idio Guarino, Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, Valerio Persico, and Antonio Pescape. 2021. Classification of Communication and Collaboration Apps via Advanced Deep-Learning Approaches. In *2021 IEEE 26th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. 1–6.

[13] Idio Guarino, Chao Wang, Alessandro Finamore, Antonio Pescape; and Dario Rossi. 2023. Many or Few Samples? Comparing Transfer, Contrastive and Meta-Learning in Encrypted Traffic Classification. In *Traffic Measurement and Analysis (TMA)*.

[14] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. 2012. Reproducible network experiments using container-based emulation. In *Proceedings of the 8th ACM International Conference on Emerging networking experiments and technologies (CoNEXT)*. 253–264.

[15] Yuqiang Heng, Vikram Chandrasekhar, and Jeffrey G. Andrews. 2021. UTMobileNetTraffic2021: A Labeled Public Network Traffic Dataset. *IEEE Networking Letters* 3, 3 (2021), 156–160.

[16] Eyal Horowicz, Tal Shapira, and Yuval Shavitt. 2022. A Few Shots Traffic Classification with Mini-FlowPic Augmentations. In *Proceedings of the 22nd ACM Internet Measurement Conference* (Nice, France) *(IMC '22)*. Association for Computing Machinery, New York, NY, USA, 647–654.

[17] Auwal Sani Iliyasu and Huifang Deng. 2020. Semi-Supervised Encrypted Traffic Classification With Deep Convolutional Generative Adversarial Networks. *IEEE Access* 8 (2020), 118–126.

[18] Peter Ivie and Douglas Thain. 2018. Reproducibility in Scientific Computing. *ACM Comput. Surv.*, Article 63 (jul 2018), 36 pages.

[19] Arthur S Jacobs, Roman Beltiukov, Walter Willinger, Ronaldo A Ferreira, Arpit Gupta, and Lisandro Z Granville. 2022. AI/ML for Network Security: The Emperor has no Clothes. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 1537–1551.

[20] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised Contrastive Learning. In *Advances in neural information processing systems (NeurIPS)*.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems (NeurIPS)*.

[22] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[23] Jan Luxemburk, Karel Hynek, Tomáš Čejka, Andrej Lukačovič, and Pavel Šiška. 2023. CESNET-QUIC22: A large one-month QUIC network traffic dataset from backbone lines. *Data in Brief* 46 (2023), 108888.

[24] Jan Luxemburk and Tomáš Čejka. 2023. Fine-grained TLS services classification with reject option. *Computer Networks* 220 (2023), 109467.

[25] Xuying Meng, Yequan Wang, Runxin Ma, Haitong Luo, Xiang Li, and Yujun Zhang. 2022. Packet Representation Learning for Traffic Classification. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Washington DC, USA) *(KDD '22)*. 3546–3554.

[26] Andrew W Moore and Konstantina Papagiannaki. 2005. Toward the accurate identification of network applications. In *Passive and Active Network Measurement: 6th International Workshop, PAM 2005, Boston, MA, USA, March 31-April 1, 2005. Proceedings 6*. Springer, 41–54.

[27] Andrew W Moore and Denis Zuev. 2005. Internet traffic classification using bayesian analysis techniques. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*. 50–60.

[28] Thuy T.T. Nguyen and Grenville Armitage. 2008. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials* 10, 4 (2008), 56–76.

[29] University of New Brunswick. 2016. VPN-nonVPN dataset (ISCXVPN2016). https://www.unb.ca/cic/datasets/vpn.html

[30] Fannia Pacheco, Ernesto Exposito, Mathieu Gineste, Cedric Baudoin, and Jose Aguilar. 2018. Towards the deployment of machine learning solutions in network traffic classification: A systematic survey. *IEEE Communications Surveys & Tutorials* 21, 2 (2018), 1988–2014.

[31] Raymond Queneau. 1947. *Exercices de style*.
[32] Shahbaz Rezaei and Xin Liu. 2019. How to Achieve High Classification Accuracy with Just a Few Labels: A Semi-supervised Approach Using Sampled Packets. In *Industrial Conference Advances in Data Mining - Applications and Theoretical Aspects (ICDM)*.
[33] Tal Shapira and Yuval Shavitt. 2019. FlowPic: Encrypted Internet Traffic Classification is as Easy as Image Recognition. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 680–687.
[34] ACM SIGCOMM. 2017. Reproducibility Workshop. https://conferences.sigcomm.org/sigcomm/2017/workshop-reproducibility.html
[35] Md. Shamim Towhid and Nashid Shahriar. 2022. Encrypted Network Traffic Classification using Self-supervised Learning. In *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*. 366–374.
[36] Chao Wang, Alessandro Finamore, Lixuan Yang, Kevin Fauvel, and Dario Rossi. 2022. AppClassNet: A Commercial-Grade Dataset for Application Identification Research. 52, 3 (sep 2022), 19–27.
[37] Ziyi Zhao, Yingya Guo, Jessie Hui Wang, Haibo Wang, Chengyuan Zhang, and Changqing An. 2022. CL-ETC: A Contrastive Learning Method for Encrypted Traffic Classification. In *2022 IFIP Networking Conference (IFIP Networking)*. 1–9.

A ETHICS

This work makes use of only publicly available data. Although experiments might have included end users, no individuals were monitored in the measurement campaigns. Thus, no ethical concerns are associated with this study.

B ARTIFACTS

All the material created for this paper is made available to the research community. This includes our modeling framework, namely `tcbench`, all data created in our modeling campaigns (the models themselves and all related logs) and the curated datasets described in Sec. 3.4. For more details on the artifacts, how to run our modeling campaigns, notebooks for recreating tables and figures in this paper, data curation, etc., please visit https://tcbenchstack.github.io/tcbench/.

C LAYOUT OF DL NETWORK ARCHITECTURES

We list here our implementation of the network architectures for the 32×32 flowpic resolution (see also Fig.7 in [16]). The Listings 1-5 at the end of this appendix are obtained via the `torchsummary` python package. For code flexibility, our architectures are designed to use Pytorch `nn.Identity()` modules to mask out layers that are not needed from a given architecture. When this masking is applied, our training framework takes care of recreating the network optimizers to reflect the architecture modifications.

D INVESTIGATING ROOT CAUSE OF G1 DISCREPANCIES

We were clearly surprised by the ≈20% classification accuracy performance gap for the human test split. In fact, in the Ref-Paper the two testing partitions have almost on par performance. As mentioned before, when reaching out to *Horowicz* et al. we received delayed and short/partial answers so we based our analysis mostly on the content of the paper. By browsing the web, we also found a git repository [9] that later on was confirmed to have been created by the first author of the Ref-Paper. Unfortunately, this repository only contains code related to the contrastive learning part of the paper (i.e., it only pre-trains a model to later investigate its latent space using a t-SNE projection in two dimensions). Moreover, the network architecture used significantly differs from the one described in the Ref-Paper (e.g., different activation functions, no dropout is used) and also adopts a cosine annealing learning rate scheduler (not mentioned in the original publication). Lastly, the data loading policies are blending flows between the three partitions of the UCDAVIS19 dataset (hence breaching the evaluation protocol defined in the paper) and we found also some of the flows in the test set to be included in the training set. In a nutshell, the repository was of no use to address our questions.

We therefore performed more analysis of the UCDAVIS19 paying particular attention to details reported in [32] which introduces the UCDAVIS19 dataset.

D.1 Our analysis of the UCDAVIS19 dataset

Next to Fig 4, Fig. 8 provides a more compelling argument about the presence of the data shift by showing the Kernel Density Estimation (KDE) of the per-class packet size distribution across all samples in the three partitions of UCDAVIS19. While script is perfectly overlapped with the pretraining split, *Google search* for human has an evident shift, which indeed matches the previous observations in Fig. 4.

D.2 UCDAVIS19 dataset analysis from [32]

We next looked at other sources of information which could help us to exclude problems from our analysis. To the best of our knowledge, the only study that investigates both script and human splits with a reference per-class breakdown is [32], i.e., the same study that introduces the UCDAVIS19 dataset. The authors wrote:

To study whether automatically generated data with script represents human interaction, we capture 15 flows for each class from interactions of real humans in those 5 Google services. We only use this dataset to test the same model described above. Fig. 3(b) illustrates the performance metrics. Interestingly, accuracy of the Google search and Google document have not changed significantly. However, the accuracy of Google drive, Youtube, and Google music drop up to 7%. This depends on how much human interactions can change the traffic pattern, which is class-dependent. Moreover, there are some actions, such as renaming a file or moving files in Google drive, that our scripts do not perform. So, these patterns are not available during re-training. This shows the limitations of datasets and studies [14, 8, 3] that only use scripts to capture data.

Interestingly, they reported no problem with *Google doc* and *Google search*, yet they acknowledged the presence of data-shift due to the way the dataset was collected. However this information alone cannot help us explaining the discrepancies we observe with respect to the Ref-Paper. In fact, [32] relies on packet time series augmented via sampling; in contrast, a flowpic is a "summary" which aggregates patterns over time and does not consider traffic direction—the two studies have intrinsically different input.

Also [35] uses the UCDAVIS19 dataset but authors combine all partitions together, i.e., they do not follow the training/testing protocol of [16, 32].

D.3 Reproduction of [32] on UCDAVIS19

Finally, to rule out errors in our execution, we leveraged [32] code artifacts.[16] To ensure that the UCDAVIS19 dataset we used was intact and correct and to analyze the impact of data shift between script and human partitions, we reproduced some the results of

[16]https://github.com/shrezaei/Semi-supervised-Learning-QUIC-

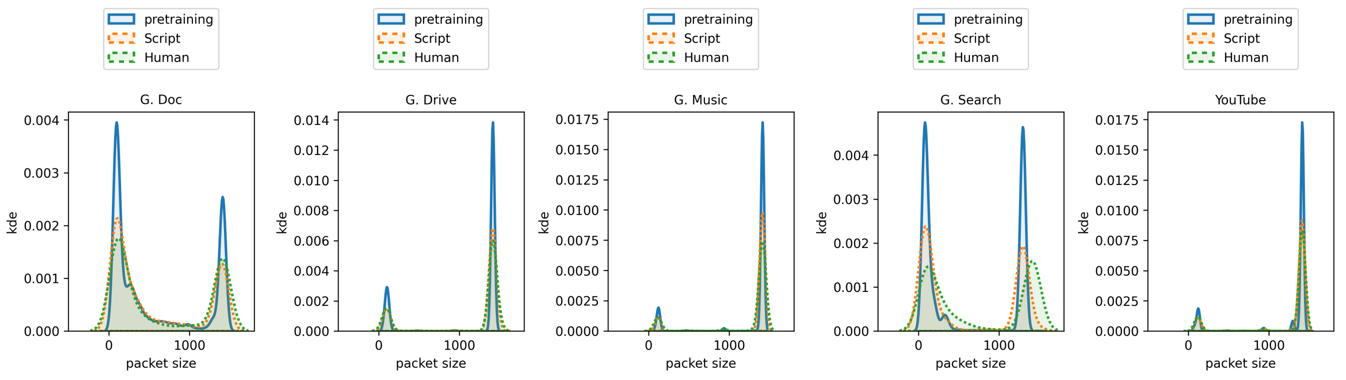


Figure 8: Kernel density estimation of the per-class packet size distributions. Notice the distribution shift for Google search.

Table 9: Macro-average accuracy with different re-training dataset and different sampling methods.

finetune on	from [32] Fig. 9 Sampling† Fixed	Rand	Incre	ours Sampling Fixed	Rand	Incre
script	92.28 (b)	92.28 (c)	95.59 (a)	87.11 ±0.05	94.63 ±0.01	96.22 ±0.04
human	-	-	-	82.60 ±0.02	87.29±0.02	92.56 ±0.01

† "Fixed": Fixed step sampling; "Rand": Random sampling; "Incre": Incremental sampling.

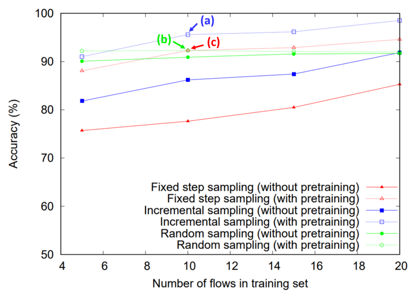
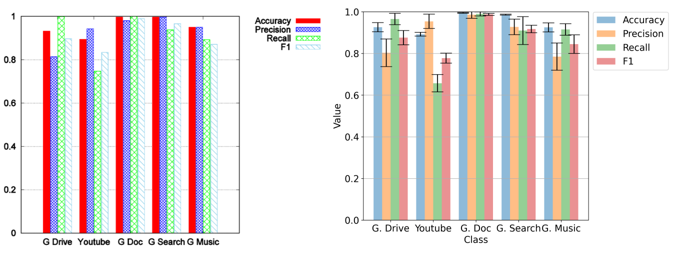


Figure 9: Accuracy on `script` **with different sampling methods[32].**

[32] using the available repository, yet using our curated version of the UCDAVIS19 dataset (which we reiterate was just reworking the original CSV files into a monolithic parquet format). In [32], for each flow, 3 different sampling methods (i.e., random sampling, fixed step sampling, and incremental sampling) are applied respectively up to 100 times to generate multiple short "subflow" time-series, thus augmenting the data set. For self-supervised pre-training on the entire pre-training partition, the authors used a statistical features regression task. For supervised fine-tuning, 3 linear layers are stacked as classifier for the classification task and they are trained with up to 20 labeled flows. While the fine-grained details of the training differ compared to our study and the Ref-Paper, at a high level these three studies share the same aim, i.e., the first pre-train (on the pretraining partition) and then fine-tune (on the two test partitions).



(a) from [32] **(b) ours**

Figure 10: Replicating per-class accuracy on `human`**.**

Table 9 reports the performance when fine-tuning with 10 samples. In [32] the performance is only measured on `script` and is only reported as a figure without numeric annotation (see (a), (b), (c) in Fig. 9, based on which we inferred the values reported on the left side of Table 9). On the right side we reported the results from our modeling campaign using the reference git repository. Overall the accuracy on `script` has differences in the range 0.68-5.17% (much smaller than the ≈20% accuracy gap under investigation), and we can also confirm their results, i.e., *incremental sampling is the best strategy* for the method reported in [32]. For `human` instead we detect a $3.66 - 7.34\%$ drop with respect to `script`. A similar drop is reported in [32] for incremental sampling when fine-tuned on 20 `script` flows and tested on `human` (i.e., in a transfer learning setting) which we were also able to replicate in Fig. 10. Their reasoning for such differences is quoted in App. D.2. Overall, this evaluation is in line with the results of [32] and shows that our preprocessing of UCDAVIS19 is not responsible for the data shift we observed.

E IMPACT OF DROPOUT IN A SUPERVISED SETTING

To assess the impact of dropout in a supervised setting, we performed an ablation study using different test sets, resolutions, and augmentations for 32×32 and 1500×1500 resolutions with the same campaign settings described in Sec. 3.2 (i.e., 15 experiments in each configuration). Fig. 11 shows the results as boxplots (with whiskers

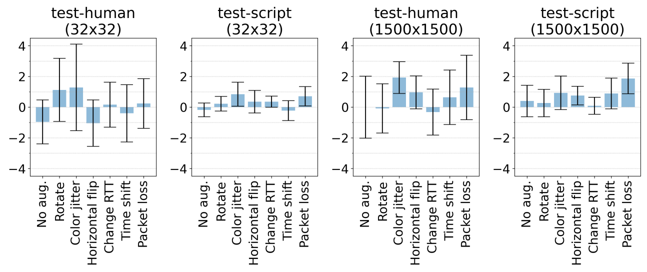


Figure 11: Boxplots of the accuracy difference between models with dropout and without dropout in supervised learning across different augmentations.

at the 95-th percentile) of the difference between the accuracy when using dropout with respect to when not using dropout. In other words, dropout would be justified if the boxplots would fall on the positive size of the y-axis. Conversely, across all scenarios, the boxplots are centered around zero with no evident patterns across augmentations. Overall, we concluded that of dropout does not play a role and its adoption (as required by the Ref-Paper) is weakly motivated.

F COMPARISON OF AUGMENTATIONS PERFORMANCE ACROSS FLOWPIC SIZES

In order to perform the analysis found in section 4.3, three sets of experiments were available, corresponding to the different flowpic resolutions used: 32×32, 64×64 and 1500×1500. If possible, it would be desirable to group the three sets into a single analysis, as that increases the N in the Critical Distance calculation, which reduces the CD's width and allows us to better differentiate between augmentations. However, first we had to ensure that the augmentations performance across sets are similar. To do so, we treated each flowpic resolution as a classifier and compared their paired performance distributions using a posthoc Tukey test, which calculates whether each resolution's performance can be assumed to be significantly different from each other or not. This test's results are shown on Table 10 with the p-values for each comparison. We used a significance level of 0.05, i.e., we can assume significant differences between resolutions if their p-value is smaller than 0.05. There are two populations for which augmentations perform in a similar way: 32×32 and 64×64, with 1500×1500 being clearly different from the other two. Based on these results, we joined the 32×32 and 64×64 populations for our analysis in section 4.3.

Table 10: Performance comparison across augmentations for different flowpic sizes. P-values extracted from Tukey's posthoc test at a 0.05 significance level.

Flowpic resolution	Flowpic resolution	p-value	Is Different?
32×32	64×64	0.57	No
32×32	1500×1500	1.93×10^{-6}	Yes
64×64	1500×1500	1.04×10^{-8}	Yes

Listing 1: Supervised network (with dropout).

```
flowpic_dim: 32
num_classes: 5
with_dropout: True
----------------------------------------------
  Layer (type)        Output Shape    Param #
==============================================
      Conv2d-1     [-1, 6, 28, 28]        156
        ReLU-2     [-1, 6, 28, 28]          0
   MaxPool2d-3     [-1, 6, 14, 14]          0
      Conv2d-4    [-1, 16, 10, 10]      2,416
        ReLU-5    [-1, 16, 10, 10]          0
   Dropout2d-6    [-1, 16, 10, 10]          0
   MaxPool2d-7      [-1, 16, 5, 5]          0
     Flatten-8           [-1, 400]          0
      Linear-9           [-1, 120]     48,120
      ReLU-10            [-1, 120]          0
    Linear-11             [-1, 84]     10,164
      ReLU-12             [-1, 84]          0
  Dropout1d-13            [-1, 84]          0
    Linear-14              [-1, 5]        425
==============================================
Total params: 61,281
Trainable params: 61,281
Non-trainable params: 0
----------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.13
Params size (MB): 0.23
Estimated Total Size (MB): 0.36
----------------------------------------------
```

Listing 2: Supervised network (without dropout).

```
flowpic_dim: 32
num_classes: 5
with_dropout: False
---------------------------------------------
  Layer (type)        Output Shape   Param #
=============================================
      Conv2d-1     [-1, 6, 28, 28]       156
        ReLU-2     [-1, 6, 28, 28]         0
   MaxPool2d-3     [-1, 6, 14, 14]         0
      Conv2d-4    [-1, 16, 10, 10]     2,416
        ReLU-5    [-1, 16, 10, 10]         0
    Identity-6    [-1, 16, 10, 10]         0 <-- masked
   MaxPool2d-7      [-1, 16, 5, 5]         0
     Flatten-8           [-1, 400]         0
      Linear-9           [-1, 120]    48,120
     ReLU-10             [-1, 120]         0
    Linear-11             [-1, 84]    10,164
     ReLU-12              [-1, 84]         0
  Identity-13             [-1, 84]         0 <-- masked
    Linear-14              [-1, 5]       425
=============================================
Total params: 61,281
Trainable params: 61,281
Non-trainable params: 0
---------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.13
Params size (MB): 0.23
Estimated Total Size (MB): 0.36
---------------------------------------------
```

Listing 3: SimCLR pre-train (small projection layer).

```
flowpic_dim: 32
num_classes: 5,
projection_layer_dim: 30
with_dropout: False
---------------------------------------------
  Layer (type)        Output Shape   Param #
=============================================
      Conv2d-1     [-1, 6, 28, 28]       156
        ReLU-2     [-1, 6, 28, 28]         0
   MaxPool2d-3     [-1, 6, 14, 14]         0
      Conv2d-4    [-1, 16, 10, 10]     2,416
        ReLU-5    [-1, 16, 10, 10]         0
    Identity-6    [-1, 16, 10, 10]         0
   MaxPool2d-7      [-1, 16, 5, 5]         0
     Flatten-8           [-1, 400]         0
      Linear-9           [-1, 120]    48,120
     ReLU-10             [-1, 120]         0
    Linear-11            [-1, 120]    14,520 <- proj layer 1
     ReLU-12             [-1, 120]         0
  Identity-13            [-1, 120]         0
    Linear-14             [-1, 30]     3,630 <- smaller proj layer
=============================================
Total params: 68,842
Trainable params: 68,842
Non-trainable params: 0
---------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.13
Params size (MB): 0.26
Estimated Total Size (MB): 0.39
---------------------------------------------
```

Listing 4: SimCLR pre-train (large projection layer).

```
flowpic_dim: 32
num_classes: 5,
projection_layer_dim: 84
with_dropout: False
---------------------------------------------
  Layer (type)        Output Shape   Param #
=============================================
      Conv2d-1     [-1, 6, 28, 28]       156
        ReLU-2     [-1, 6, 28, 28]         0
   MaxPool2d-3     [-1, 6, 14, 14]         0
      Conv2d-4    [-1, 16, 10, 10]     2,416
        ReLU-5    [-1, 16, 10, 10]         0
    Identity-6    [-1, 16, 10, 10]         0
   MaxPool2d-7      [-1, 16, 5, 5]         0
     Flatten-8           [-1, 400]         0
      Linear-9           [-1, 120]    48,120
      ReLU-10            [-1, 120]         0
     Linear-11           [-1, 120]    14,520 <- proj layer 1
       ReLU-12           [-1, 120]         0
   Identity-13           [-1, 120]         0
     Linear-14            [-1, 84]    10,164 <- larger proj layer
=============================================
Total params: 75,376
Trainable params: 75,376
Non-trainable params: 0
---------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.13
Params size (MB): 0.29
Estimated Total Size (MB): 0.42
---------------------------------------------
```

Listing 5: Fine-tune network.

```
flowpic_dim: 32
num_classes: 5,
projection_layer_dim: 30
with_dropout: False
---------------------------------------------
  Layer (type)        Output Shape   Param #
=============================================
      Conv2d-1     [-1, 6, 28, 28]       156
        ReLU-2     [-1, 6, 28, 28]         0
   MaxPool2d-3     [-1, 6, 14, 14]         0
      Conv2d-4    [-1, 16, 10, 10]     2,416
        ReLU-5    [-1, 16, 10, 10]         0
    Identity-6    [-1, 16, 10, 10]         0
   MaxPool2d-7      [-1, 16, 5, 5]         0
     Flatten-8           [-1, 400]         0
      Linear-9           [-1, 120]    48,120
      ReLU-10            [-1, 120]         0
   Identity-11           [-1, 120]         0 <- masked
   Identity-12           [-1, 120]         0 <- masked
   Identity-13           [-1, 120]         0 <- masked
     Linear-14             [-1, 5]       605 <- final classifier
=============================================
Total params: 51,297
Trainable params: 51,297
Non-trainable params: 0
---------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.13
Params size (MB): 0.20
Estimated Total Size (MB): 0.33
---------------------------------------------
```