Data Augmentation for Traffic Classification

Chao Wang[1,2][0009−0003−5721−5221], Alessandro Finamore[1][0000−0003−2226−2506], Pietro Michiardi[2][0000−0003−4675−7677], Massimo Gallo[1][0000−0001−8781−0775], and Dario Rossi[1][0000−0003−3936−8876]

[1] Huawei Technologies SASU, France
[2] EURECOM, France

Abstract. Data Augmentation (DA)—enriching training data by adding synthetic samples—is a technique widely adopted in Computer Vision (CV) and Natural Language Processing (NLP) tasks to improve models performance. Yet, DA has struggled to gain traction in networking contexts, particularly in Traffic Classification (TC) tasks. In this work, we fulfill this gap by benchmarking 18 augmentation functions applied to 3 TC datasets using packet time series as input representation and considering a variety of training conditions. Our results show that (*i*) DA can reap benefits previously unexplored, (*ii*) augmentations acting on time series sequence order and masking are better suited for TC than amplitude augmentations and (*iii*) basic models latent space analysis can help understanding the positive/negative effects of augmentations on classification performance.

1 Introduction

Network monitoring is at the core of network operations with Traffic Classification (TC) being key for traffic management. Traditional Deep Packet Inspection (DPI) techniques, i.e., classifying traffic with rules related to packets content, is nowadays more and more challenged by the growth in adoption of TLS/DNSSEC/HTTPS. Despite the quest for alternative solutions to DPI already sparked about two decades ago with the first Machine Learning (ML) models based on packet and flow features, a renewed thrust in addressing TC via data-driven modeling is fueled today by the rise of Deep Learning (DL), with abundant TC literature, periodically surveyed [29, 34], reusing/adapting Computer Vision (CV) training algorithms and model architectures.

Despite the existing literature, we argue that *opportunities laying in the data itself are still unexplored* based on three observations. First, CV and Natural Language Processing (NLP) methods usually leverage "cheap" Data Augmentation (DA) strategies (e.g., image rotation or synonym replacement) to complement training data by increasing samples variety. Empirical studies show that this leads to improved classification accuracy. Yet to the best of our knowledge, only a handful of TC studies considered DA [20, 32, 46] and multiple aspects of DA design space remain unexplored. Second, network traffic datasets are imbalanced due to the natural skew of app/service popularity and traffic dynamics. In turn,

this calls for training strategies emphasizing classification performance improvement for classes with fewer samples. However, the interplay between imbalance and model performance is typically ignored in TC literature. Last, the pursuit of better model generalization and robustness necessitates large-scale datasets with high-quality labeling resulting in expensive data collection processes. In this context, the extent to which DA can alleviate this burden remains unexplored.

In this paper, we fill these gaps by providing a comprehensive evaluation of "hand-crafted" augmentations—transformations designed based on domain knowledge—applied to packets time series typically used as input in TC. Given the broad design space, we defined research goals across multiple dimensions. First of all, we selected a large pool of 18 augmentations across 3 families (amplitude, masking, and sequence) which we benchmark both when used in isolation as well as when multiple augmentations are combined (e.g., via stacking or ensembling). Augmentations are combined with original training data via different batching policies (e.g., replacing training data with augmentation, adding augmented data to each training step, or pre-augmenting the dataset before training). We also included scenarios where imbalanced datasets are re-balanced during training to give more importance to minority classes. Last, we dissected augmentations performance by exploring their geometry in the classifiers latent space to pinpoint root causes driving performance. Our experimental campaigns were carried over 2 mid-sized public datasets, namely `MIRAGE-19` and `MIRAGE-22` (up to 20 classes, 64k flows), and a larger private dataset (100 classes, 2.9M flows). We summarize our major findings as follows:

- We confirm that augmentations improve performance (up to +4.4% weighted F1) and expanding training batches during training (i.e., the Injection policy) is the most effective policy to introduce augmentations. Yet, improvements are dataset dependent and not necessarily linearly related to dataset size or number of classes to model;
- Sequence ordering and masking are more effective augmentation families for TC tasks. Yet, no single augmentation is found consistently superior across datasets, nor domain knowledge suffice to craft effective augmentations, i.e., the quest for effective augmentations is an intrinsic trial-and-error process;
- Effective augmentations introduce good sample variety, i.e., they synthesize samples that are neither too close nor too far from the original training data.

To the best of our knowledge, a broad and systematic study of hand-crafted DA techniques in TC as the one performed in our study is unprecedented. Ultimately, our analysis confirms that DA is currently suffering from a single pain point—exploring the design space via brute force. However, our results suggest a possible road map to achieve better augmentations via generative models which might render obsolete the use of brute force.

In the remainder, we start by introducing DA basic concepts and reviewing relevant ML and TC literature (Sec. 2). We then introduce and discuss our research goals (Sec. 3) and the experimental setting used to address them (Sec. 4). Last, we present our results (Sec. 5) before closing with final remarks (Sec. 6).

2 Background and related work

Data augmentation consists in adding synthetic samples (typically derived from real ones) to the training set to increase its variety. DA has been popularized across many ML disciplines [26, 36, 44] with a large number of variants which we can be broadly grouped into two categories [26]: hand-crafted DA and data synthesis. Hand-crafted DA (also known as data transformations) involves creating new samples by applying predefined rules to existing samples. Instead, data synthesis relates to generating new samples via generative models, e.g., Variational AutoEncoders (VAE), Generative Adversarial Neural networks (GAN), Diffusion Models (DM), etc., trained on existing and typically large datasets.

In this section, we overview the existing DA literature with an emphasis on hand-crafted DA and methods closer to the scope of our work. We begin by introducing relevant CV and time series ML literature. Then, we review TC literature using DA and close with a discussion about general design principles/requirements that we used for defining our research goals outlined in Sec. 3.

2.1 Data augmentation in traditional machine learning tasks

To ground the discussion of different methods merits, we start by revisiting the internal mechanisms of supervised ML/DL models.

Supervised modeling and DA. In a nutshell, a supervised model is a function $\varphi : \mathbf{x} \in \mathcal{X} \rightarrow y \in \mathcal{Y}$ mapping an input $\mathbf{x}$ to its label y. Training such models corresponds to discover a good function $\varphi(\cdot)$ based on a training set. When performing DA, the training set is enlarged by adding new samples $\mathbf{x}' = \text{Aug}(\mathbf{x})$ created by altering original samples $\mathbf{x}$—these transformations act directly in the input space $\mathcal{X}$ and the additional synthetic samples contribute in defining $\varphi(\cdot)$ as much as the original ones. It follows that having a comprehensive understanding of samples/classes properties and their contribution to models training is beneficial for designing *effective* augmentations, i.e., transformations enabling higher classification performance.

Beside operating in the input space, DL models offer also a latent space. In fact, DL models are typically a composition of two functions $\varphi(\mathbf{x_i}) = h(f(\mathbf{x_i})) = y_i$: a feature extractor $f(\cdot)$ and a classifier $h(\cdot)$, normally a single fully connected layer (i.e., a linear classifier) in TC. In other words, an input sample $f(\mathbf{x_i}) = \mathbf{z_i}$ is first projected into an intermediate space, namely the *latent space*, where different classes are expected to occupy different regions. The better such separation, the easier is for the classifier $h(\mathbf{z_i}) = y_i$ to identify the correct label. It follows that this design enables a second form of augmentations based on altering samples in the latent space rather than in the input space.

Last, differently from DA, generative models aims to learn the training set data distribution. In this way generating new synthetic data corresponds to sampling from the learned distribution. In the following we expand on each of these three methodologies.

Input space transformations. In traditional ML, Synthetic Minority Oversampling TEchnique (SMOTE) [7] is a popular augmentation technique. This approach generates new samples by interpolating the nearest neighbors of a given training sample. To address class imbalance, SMOTE is often employed with a sampling mechanism that prioritizes minority classes [16, 17].

In CV, several image transformations have been proposed to improve samples variety while preserving classes semantics. These transformations operate on colors (e.g., contrast and brightness changes, gray scaling) and geometry (e.g., rotation, flipping, and zooming), or via filters (e.g., blurring with Gaussian kernel) and masks (e.g., randomly set to zero a patch of pixels). Furthermore, transformations like CutMix [51] and Mixup [52] not only increase samples variety but also increase *classes variety* by creating synthetic classes from a linear combination of existing ones. The rationale behind this approach is that by introducing new artificial classes sharing similarities with the true classes the classification task becomes intentionally more complex, thereby pushing the training process to extract better data representations. Empirical validations of DA techniques in CV have consistently demonstrated their effectiveness across a diverse range of datasets, tasks, and training paradigms [8, 18, 31]. As a result, DA has become a ubiquitous component in the CV models training pipelines.

Considering time series instead, input transformations can either modify data *amplitude* (e.g., additive Gaussian noise) or manipulate *time* (e.g., composing new time series by combining different segments of existing ones). Similarly to CV, the research community has provided empirical evidence supporting the effectiveness of these transformations in biobehavioral [47] and health [49] domains. However, contrarily to CV, these transformations are less diverse and have been less widely adopted, possibly due to the stronger reliance on domain knowledge—an amplitude change on an electrocardiogram can be more difficult to properly tune compared to simply rotating an image.

Latent space transformations. Differently from traditional ML, DL models offer the ability to shape the feature extractor to create more "abstract" features. For example, Implicit Semantic Data Augmentation (ISDA) [42] first computes class-conditional covariance matrices based on intra-class feature variety; then, it augments features by translating real features along random directions sampled from a Gaussian distribution defined by the class-conditional covariance matrix. To avoid computational inefficiencies caused by explicitly augmenting each sample many times, ISDA computes an upper bound of the expected cross entropy loss on an enlarged feature set and takes this upper bound as the new loss function. Based on ISDA, and focused on data imbalance, Sample-Adaptive Feature Augmentation (SAFA) [19] extracts transferable features from the majority classes and translates features from the minority classes in accordance with the extracted semantic directions for augmentation.

Generative models. In addition to traditional hand-crafted data augmentation techniques, generative models offer an alternative solution to generate samples variety. For instance, [6, 40] use a multi-modal diffusion model trained on an

Internet-scale dataset composed of (image, text) pairs. Then, the model is used to synthesize new samples—text prompts tailored to specific downstream classification tasks are used as conditioning signal to create task-specific samples—to enlarge the training set for a classification task. While these types of generative models can provide high-quality samples variety, their design and application still requires a considerable amount of domain knowledge to be effective.

2.2 Data augmentation in traffic classification

TC tasks usually rely on either packet time series (e.g., packet size, direction, Inter Arrival Time (IAT), etc., of the first 10-30 packets of a flow) or payload bytes (e.g., the first 784 bytes of a flow, possibly gathered by concatenating payload across different packets) arranged as 2d matrices. Recent literature also considers combining both input types into multi-modal architectures [2, 4, 24].

Such input representations and datasets exhibit three notable distinctions when compared to data from other ML/DL disciplines. First, TC datasets show *significant class imbalance*—this is a "native" property of network traffic as different applications enjoy different popularity and traffic dynamics while, for instance, many CV datasets are balanced. Second, TC input representation is typically *"small"* to adhere to desirable system design properties—network traffic should be (*i*) *early classified*, i.e., the application associated to a flow should be identified within the first few packets of a flow, and (*ii*) computational/memory resources required to represent a flow should be minimal as an in-network TC systems need to cope with hundreds of thousands of flow per second. Last, TC input data has *weak semantics*—the underlying application protocols (which may or not be known a priori) may not be easy to interpret even for domain experts when visually inspecting packet time series.

Hand-crafted DA. The combination of the above observations leads to have only a handful of studies adopting DA in TC. Rezaei et al. [32] created synthetic input samples by means of three hand-crafted DA strategies based on sampling multiple short sequences across the duration of a complete flow. Horowicz et al. [20] instead focused on a *flowpic* input representation—a 2d summary of the evolution of packets size throughout the duration of a flow—augmented by first altering the time series collected from the first 15s of a flow and used to compute the flowpic. While both studies show the benefit of DA, these strategies violate the early classification principle as they both consider multiple seconds of traffic, thus they are better suited for post-mortem analysis only. Conversely, Xie et al. [46] recently proposed some packet series hand-crafted DA to tackle data shifts arising when applying a model on network traffic gathered from networks different from the ones used to collect the training dataset. Specifically, inspired by TCP protocol dynamics, authors proposed five packets time series augmentations (e.g., to mimic a packet lost/retransmission one can replicate a value at a later position in the time series) showing that they help to mitigate data shifts. Yet, differently from [20, 32], the study in [46] lacks from an ablation of each individual augmentation's performance.

Generative models. Last, [41, 43, 48] investigate augmentations based on GAN methods when using payload bytes as input for intrusion detection scenarios, i.e., a very special case of TC where the classification task is binary. More recently, [22] compared GAN and diffusion model for generating raw payload bytes traces while [37, 38] instead leveraged GAN or diffusion models to generate 2D representations (namely GASF) of longer traffic flow signals for downstream traffic fingerprinting, anomaly detection, and TC.

2.3 Design space

Search space. Independently from the methodology and application discipline, DA performance can only be assessed via empirical studies, i.e., results are bound to the scenarios and the datasets used. Moreover, to find an efficient strategy one should consider an array of options, each likely subject to a different parametrization. In the case of hand-crafted DA, one can also opt for using *stacking* (i.e., applying a sequence of transformations) or *ensembling* (i.e., applying augmentations by selecting from a pool of candidates according to some sampling logic)—an exhaustive grid search is unfeasible given the large search space. Besides following guidelines to reduce the number of options [10], some studies suggest the use of reinforcement learning to guide the search space exploration [9]. Yet, no standard practice has emerged.

Quantifying good variety. As observed in TC literature [20, 32, 46], domain knowledge is key to design efficient augmentations. Yet, ingenuity might not be enough as models are commonly used as "black boxes", making it extremely challenging to establish a direct link between an augmentation technique and its impact on the final classification performance. For instance, rotation is considered a good image transformation as result of empirical studies. Likewise, generative models are trained on large image datasets but without an explicit connection to a classification task [33]—the design of the augmentation method itself is part of a trial-and-error approach and the definition of metrics quantifying the augmentation quality is still an open question.

One of the aspects to be considered when formulating such metrics is the *variety* introduced by the augmentations. Gontijo-Lopes et al. [11] propose metrics quantifying the distribution shift and diversity introduced by DA contrasting models performance with and without augmentations. Other literature instead focuses on mechanisms that can help defining desirable properties for augmentations. For instance, from the feature learning literature, [35, 53] find that DA induces models to better learn rare/less popular but good features by altering their importance, thus improving model generalization performance. Samyak et al. [21] find that optimization trajectories are different when training on different augmentations and propose to aggregate the weights of models trained on different augmentations to obtain a more uniform distribution of feature patches, encouraging the learning of diverse and robust features.

Training loss. Self-supervision and contrastive learning are DL training strategies that take advantage of augmentations by design. In a nutshell, contrastive learning consists of a 2-steps training process. First, a feature extractor is trained in a *self-supervised* manner with a contrastive loss function that pulls together different augmented "views" of a given sample while distancing them from views of other samples. Then, a classifier head is trained on top of the learned representation in a *supervised* manner using a few labeled samples—the better the feature representation, the lower the number of labeled samples required for training the head. Empirical studies have demonstrated the robustness of the feature representations learned with contrastive learning [8] and a few recent studies investigated contrastive learning also in TC [15, 20, 39, 46].

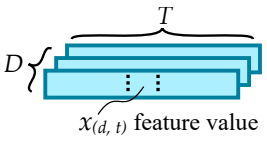
Linking generative models to classifiers. When we consider the specific case of using generative models to augment training data, we face a major challenge—generative models are not designed to target a specific downstream task [37, 38, 43]. While studies like [28] integrated a classifier in GAN training in the pursuit of improving the reliability the model, how to properly link and train a generative model to be sensitive to a downstream classification task is still an open question even in CV literature.

3 Our goals and methodology

Drawing insights from the literature reviewed in Sec. 2, we undertake a set of empirical campaigns to better understand hand-crafted DA when applied in the input space for TC task and address the following research goals:

- **G1.** How to compare the performance of different augmentations? This includes investigating augmentations sensitivity to their hyper-parametrization and dataset properties (e.g., number of samples and classes).
- **G2.** How augmented samples should be added to the training set and how many samples should be added? Is augmenting minority classes beneficial to mitigate class imbalance?
- **G3.** Why some augmentations are more effective than others?
- **G4.** Does combining multiple augmentations provide extra performance improvement?

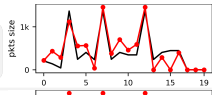
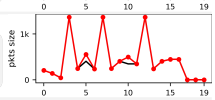
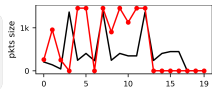
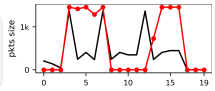
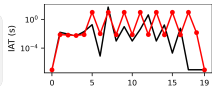
In the remainder of this section, we motivate each goal and introduce the methodology we adopted to address them.



D Number of packet features
T Number of packets per feature
$\sigma^{y}_{(d,\,t)}$ Standard deviation of a generic coordinate $(d,\,t)$ for class y
$\overline{\sigma^{y}_{(d,\,:)}}$ Average standard deviation for dimension d for class y

Fig. 1: Input sample $\mathbf{x}$ shape and related notation.

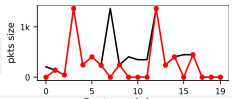
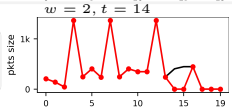
Table 1: Amplitude augmentations.

Name	Pkts Feat. Size	Dir	IAT	Description	Example magnitude $\alpha = 0.5$
Gaussian Noise	◐	○	◐	Add independently sampled Gaussian noise to Size or IAT *Details:* Sample a feature $d \in \{\text{Size}, \text{IAT}\}$ and add Gaussian noise to its values $x_{(d,t)} + \varepsilon_t$ where $\varepsilon_t \sim \mathcal{N}(0, \alpha\{\sigma^{y}_{(d,t)}\}^2)$	
Spike Noise [44]	◐	○	◐	Add independently sampled Gaussian noise to Size or IAT *Details:* Sample a feature $d \in \{\text{Size}, \text{IAT}\}$ and add Gaussian noise to up to 3 of its non-zero values $x_{(d,t)} + \lvert\varepsilon_t\rvert$ where $\varepsilon_t \sim \mathcal{N}(0, \alpha\{\sigma^{y}_{(d,t)}\}^2)$	
Gaussian WrapUp [13]	◐	○	◐	Scale Size or IAT by independently sampled Gaussian values *Details:* Sample a feature $d \in \{\text{Size}, \text{IAT}\}$ and multiply Gaussian noise to its values $x_{(d,t)} \cdot \varepsilon_t$ with $\epsilon_t \sim \mathcal{N}(1 + 0.01\alpha, 0.02\alpha\{\sigma^{y}_{(d,t)}\}^2)$	
Sine WrapUp [30]	◐	○	◐	Scale Size or IAT by sinusoidal noise *Details:* Sample a feature $d \in \{\text{Size}, \text{IAT}\}$ and multiply its values by a sine-like noise $x_{(d,t)} \cdot \epsilon_i$ with $\epsilon_i = [1 + 0.02\alpha \cdot \overline{\sigma^{y}_{(d,:)}} \cdot \sin(\frac{4\pi i}{T} + \theta)]$ and $\theta \sim U[0, 2\pi[$	
Constant WrapUp [20]	○	○	●	Scale IAT by a single randomly sampled value *Details:* Sample a single uniformly sampled value $\epsilon \sim U[a, b]$ and perform $x_i \cdot \epsilon$ to all x_i of IAT with $a = 1 + \overline{\sigma^{y}_{(d,:)}} \cdot (0.06 - 0.02\alpha)$; $b = 1 + \overline{\sigma^{y}_{(d,:)}} \cdot (0.14 + 0.02\alpha)$	

○ feature never used; ◐ feature selected randomly; ● feature always used.
In the figures, black solid lines ▬ for original samples, red ● lines for augmented samples; x-axis for time series index and y-axis the feature value (either packet size or IAT).

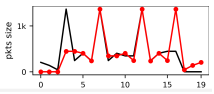
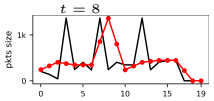
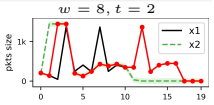
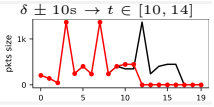
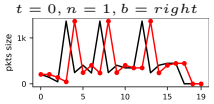
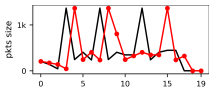
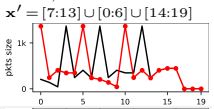
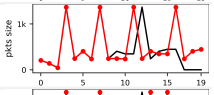
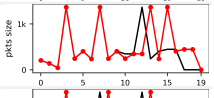
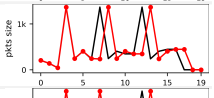
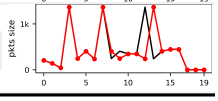
Table 2: Masking augmentations

All three features (Size, DIR and IAT) are affected by all transformations.

Name	Description	Example magnitude $\alpha = 0.5$
Bernoulli Mask [50]	Random masking values *Details:* Independently set to zero feature values by sampling a Bernoulli$(p = 0.6\alpha)$	
Window Mask [30]	Masking the same sequences across all features *Details:* Given a configured maximum size $W = \lfloor 1 + 2.5\alpha \rceil$, sample a window length $w \sim U[1, W]$ and a random starting point $t = U[0, T - w]$ and set to zero all $x_{(:,t)}$ falling in the sampled window	$w = 2, t = 14$ 

In the figures, black solid lines ▬ for original samples, red ● lines for augmented samples.

Table 3: Sequence order augmentations.

Name	Description	Example magnitude $\alpha = 0.5$
Horizontal Flip [30]	Swap values left to right (no magnitude needed)	
Interpolation [30]	Densify time series by injecting average values and then sample a new sequence of length T *Details:* Expand each feature by inserting the average $0.5(x_{(d,t)} + x_{(d,t+1)})$ in-between each pair of values. Then randomly select a starting point t=$U[0, T-1]$ and extract the following T values for all features $x_{(:,t:t+T)}$ (no magnitude needed).	$t = 8$ 
CutMix [51]	Swap segments of two different samples *Details:* Given a training mini-batch, define pairs of samples $(\mathbf{x1}, \mathbf{x2})$ by sampling without replacement. Then sample a *segment* of length $w \sim U[0, T-1]$ starting at $t \sim U[0, T-1-w]$ and swap the segment of each feature between **x1** and **x2** (no magnitude needed).	$w = 8, t = 2$ 
Packet Loss [20]	Remove values in a random time range (as if packets were not received) *Details:* Defining Δ as time to observe the first T packets, sample $\delta \sim U[0, \Delta]$ and remove values across all features in the interval $\delta \pm (10\alpha + 5)$. Then recompute the IAT and pad with zeroes at the end (if needed).	$\delta \pm 10\text{s} \rightarrow t \in [10, 14]$ 
Translation	Move a segment to left (≈pkt drop) or the right (≈ pkt dup/retran) *Details:* Define N=1+arg $\max_i\{a_i \leq \alpha\}$ where $a_i \in \{0.15, 0.3, 0.5, 0.8\}$ and sample $n \sim U[1, N]$. Then, sample a direction $b \in \{left, right\}$ and a starting point $t \sim U[0, T]$: If $b = left$, left shift each feature values n times starting from t and replace shifted values with zero; if $b = right$, right shift each feature values n times starting from t and replace shifted values with the single value $x_{(d,t)}$	$t = 0, n = 1, b = right$ 
Wrap [30]	Mixing interpolation, drop and no change *Details:* Compose a new sample $\mathbf{x}'$ by manipulating each $x_{(:,t)}$ based on three options with probabilities $P_{interpolate} = P_{discard} = 0.5\alpha$ and $P_{nochange} = 1 - \alpha$. If "nochange" then keep $x_{(:,t)}$; if "interpolate" then keep $x_{(:,t)}$ and $x_{(:,t)} = 0.5(x_{(:,t)} + x_{(:,t+1)})$; if "nochange" then do nothing. Stop when $\lvert\mathbf{x}'\rvert = \mathbf{T}$ or apply tail padding (if needed).	
Permutation [13]	Segment the time series and reorder the segments *Details:* Define N=~~2~~+arg $\max_i\{a_i \leq \alpha\}$ where $a_i \in \{0.15, 0.45, 0.75, 0.9\}$, a sample $n \sim U[2, N]$ and split the range [0:T-1] into n segments of random length. Compose a new sample $\mathbf{x}'$ by concatenating $x_{(:,t)}$ from a random order of segments	$n = 3,$ $\mathbf{x}' = [7{:}13] \cup [0{:}6] \cup [14{:}19]$ 
Dup-RTO [46]	Mimic TCP pkt retrans due to timeout by duplicating values *Details:* Duplicating a range of packets according to a Bernoulli($p = 0.1\alpha$) (see Algo. 1 in [1])	
Dup-FastRetr [46]	Mimic TCP fast retrans by duplicating values *Details:* Duplicating one packet according to a Bernoulli($p = 0.1\alpha$) (see Algo. 2 in [1])	
Perm-RTO [46]	Mimic TCP pkt retrans due to timeout by permuting values *Details:* Delaying a range of packets according to a Bernoulli($p = 0.1\alpha$) (see Algo. 3 in [1])	
Perm-FastRetr [46]	Mimic TCP fast retrans by permuting values *Details:* Delaying one packet according to a Bernoulli($p = 0.1\alpha$) (see Algo. 4 in [1])	

In the figures, black solid lines ▬ for original samples, red ● lines for augmented samples.

3.1 Benchmarking hand-crafted DA (G1)

Figure 1 sketches a typical TC input $\mathbf{x}$, i.e., a multivariate time series with D dimensions (one for each packet feature) each having T values (one for each packet) while $x_{(d,t)}$ is the value of $\mathbf{x}$ at coordinates (d, t) where $d \in \{0..D-1\}$ and $t \in \{0..T-1\}$. In particular, in this work, we consider $D = 3$ packet features, namely packet size, direction, and Inter Arrival Time (IAT), and the first $T = 20$ packets of a flow. We also define $\mathbf{x}' = \text{Aug}(\mathbf{x}, \alpha)$ as an augmentation, i.e., the transformation $\mathbf{x}'$ of sample $\mathbf{x}$ is subject to a *magnitude* $\alpha \in]0, 1[$ controlling the intensity of the transformation (1 = maximum modification).

Augmentations pool. In this study, we considered a set $\mathcal{A}$ of 18 augmentation functions. These functions can be categorized into 3 families: 5 *amplitude* transformations, which introduce different type of jittering to the feature values (Table 1); 2 *masking* transformations, which force certain feature values to zero (Table 2); and 11 *sequence* transformations, which modify the order of feature values (Table 3). It is important to note that, given a sample $\mathbf{x}$, amplitude augmentations are solely applied to either packet size or IAT while packet direction is never altered since the latter is a binary feature and does not have amplitude (i.e., it can be -1 or 1). On the contrary, masking and sequence augmentations are applied to all features in parallel (e.g., if a transformation requires to swap $t = 1$ with $t = 6$, all features are swapped accordingly $x_{(i,1)} \leftrightarrow x_{(i,6)}$ for $\forall i \in \{0..D-1\}$). For each augmentation, Tables 1-3 report a reference example annotating its parametrization (if any).

By adopting such a large pool of augmentations our empirical campaign offers several advantages. First, we are able to investigate a broader range of design possibilities compared to previous studies. Second, it enables us to contrast different families and assess if any of them is more prone to disrupt class semantics. Considering the latter, TC literature [20, 32, 46] predominantly investigate sequence transformations (typically acting only on packet timestamp) with only [46] experimenting with masking and amplitude variation, yet targeting scenarios where models are exposed to data shifts due to maximum segment size (MSS) changes, i.e., the network properties related to the training set are different from the ones of the test set.

Augmentations magnitude. As described in Tables 1-3, each augmentation has some predefined static parameters[3] while the magnitude α is the single hyper-parameter controlling random sampling mechanisms contributing to defining the final transformed samples. To quantify augmentations sensitivity to α, we contrast two scenarios following CV literature practice: a static value of $\alpha = 0.5$ and a uniformly sampled value $\alpha \sim U[0, 1]$ extracted for each augmented sample.

Datasets size and task complexity. Supervised tasks, especially when modeled via DL, benefit from large datasets. For instance, as previously mentioned, some CV literature pretrains generative models on large datasets and use those

[3] These parameters are tuned via preliminary investigations.

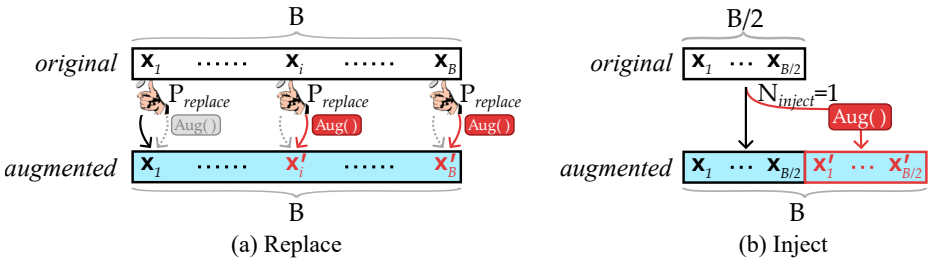


Fig. 2: Training batch creation policies.

models to obtain auxiliary training data for classification tasks. While data availability clearly plays a role, at the same time the task complexity is equivalently important—a task with just a few classes but a lot of data does not necessarily yield higher accuracy than a task with more classes and less data. To understand how augmentations interplay with these dynamics, it is relevant to evaluate augmentations across datasets of different sizes and number of classes.

3.2 Training batches composition (G2)

In order to mitigate any undesirable shifts introduced by artificial samples, it is necessary to balance original and augmented samples. Yet, the way original and augmented samples are combined to form the augmented training set is a design choice. For instance, in TC literature, [20, 32] augment the data before starting the training, while [46] augments mini-batches during the training process. In this work, we apply augmentation samples to a training mini-batch of size B, with the two policies sketched in Fig. 2. *Replace* substitutes an original sample $\mathbf{x_i}$ with its augmentation by sampling from a Bernoulli(P=$P_{replace}$) random variable—during one training epoch, approximately a $P_{replace}$ fraction of the original data is "hidden". Instead, *Inject* increases the batch size by augmenting each sample N_{inject} times (e.g., in Fig. 2 the original batch size is doubled by setting $N_{inject} = 1$).

3.3 Latent space geometry (G3)

Augmented samples play a crucial role in model training, just like the original training samples from which they are derived. To understand the impact of augmentations on the improvement or detriment of classification performance, we propose to examine the latent space of the classifier. In order to conduct a comprehensive analysis, we need to consider two aspects applicable to any supervised classification task.

Augmentation-vs-Test. ML methods operate on the assumption that training data serves as a "proxy" for test samples, i.e., the patterns learned on training data "generalize" to testing data as the two sets of data resemble each other properties. In this context, augmentations can be considered as a means for

fostering data generalization by incorporating samples that resemble even more testing data compared to what is available in training data. However, it is important to empirically quantify this effect by measuring, for instance, the distance between augmented and test samples. In other words, we aim to quantify up to which extent augmented samples are better at mimicking test samples compared to the original data.

Augmentation-vs-Train. The performance of a feature extractor greatly depends on how well the feature extractor separates different classes in the latent space. Data augmentations play a role in shaping intra/inter-class relationships created by the feature extractor in the latent space. For instance, an augmentation that generates samples far away from the region of a class can *disrupt class semantics*—the augmentation is introducing a new behavior/mode making it hard for the classifier to be effective. At the same time, however, expanding the region of a class can be a beneficial design choice—augmentations that enable a better definition of class boundaries simplify the task of the classifier. Understanding such dynamics requires empirical observations, for instance, by comparing the distance between original training data and augmented data. In other words, we aim to verify if augmentations yielding good performance are in a "sweet spot": they create samples that are neither too close (i.e., introduce too little variety) nor too far (i.e., disrupt class semantics) from original samples.

3.4 Combining augmentations (G4)

To address **G1**, each trained model is associated to an individual augmentation. However, in CV it is very common to combine multiple augmentations [8]. Hence, we aim to complement **G1** by measuring the performance of three different policies augmenting mini-batches based on a set $\mathcal{A}' \subset \mathcal{A}$ composed of top-performing augmentations based on the **G1** benchmark: the *Ensemble* policy uniformly samples one of the augmentations in $\mathcal{A}'$ independently for each mini-batch sample; the *RandomStack* policy randomly shuffles $\mathcal{A}'$ independently for each mini-batch sample before applying all augmentations; finally, the *MaskedStack* policy uses a predefined order for $\mathcal{A}'$ but each augmentation is associated to a masking probability, i.e., each sample in the mini-batch independently selects a subset of augmentations of the predefined order.

4 Experimental settings

4.1 Datasets

To address our research goals we considered the datasets summarized in Table 4.

`MIRAGE-19` [3] is a *public* dataset gathering traffic logs from 20 popular Android apps[4] collected at the ARCLAB laboratory of the University of Napoli Federico

[4] Despite being advertised with having traffic from 40 apps, the *public* version of the dataset only contains 20 apps.

Table 4: Summary of datasets properties.

Name	Classes	Curation	Flows per-class				Pkts
			all	*min*	*max*	ρ	*mean*
MIRAGE-19 [3]	20	none	122 k	1,986	11,737	5.9	23
		>10pkts	64 k	1,013	7,505	7.4	17
MIRAGE-22 [14]	9	none	59 k	2,252	18,882	8.4	3,068
		>10pkts	26 k	970	4,437	4.6	6,598
Enterprise	100	none	2.9 M	501,221	5,715	87.7	2,312

ρ : ratio between max and min number of flows per-class—the larger the value, the higher the imbalance;

II. Multiple measurement campaigns were operated by instrumenting 3 Android devices handed off to ≈300 volunteers (students and researchers) for interacting with the selected apps for short sessions. Each session resulted in a pcap file and an `strace` log mapping each socket to the corresponding Android application name. Pcaps were then post-processed to obtain bidirectional flow logs by grouping all packets belonging to the same 5-tuple (srcIP, srcPort, dstIP, dstPort, L4proto) and extracting both aggregate metrics (e.g., total bytes, packets, etc.), per-packet time series (packet size, direction, TCP flags, etc.), raw packets payload bytes (encoded as a list of integer values) and mapping a ground-truth label by means of the `strace` logs.

MIRAGE-22 [14] is another *public* dataset collected by the same research team and with the same instrumentation as MIRAGE-19 which targets 9 video meeting applications used to perform webinars (i.e., meetings with multiple attendees and a single broadcaster), audio calls (i.e., meetings with two participants using audio-only), video calls (i.e., meetings with two participants using both audio and video), and video conferences, (i.e., meetings involving more than two participants broadcasting audio and video).

Enterprise is instead a *private*[5] dataset collected by monitoring network flows from vantage points deployed in residential access and enterprise campus networks. For each flow, the logs report multiple aggregate metrics (number of bytes, packets, TCP flags counters, round trip time statistics, etc.), and the packet time series of packet size, direction and IAT for the first 50 packets of each flow. Moreover, each flow record is also enriched with an application label provided by a commercial DPI software directly integrated into the monitoring solution and supporting hundreds of applications and services.

Data curation. Table 4 compares different dataset properties. For instance, MIRAGE-19 and MIRAGE-22 are quite different from each other despite being obtained via the same platform. Specifically, MIRAGE-19 gathers around 2× more flows than MIRAGE-22 but those are 100× shorter. As expected, all datasets are subject to class imbalance measured by ρ, i.e., the ratio between maximum and

[5] Due to NDA we are not allowed to share the dataset.

minimum number of samples per class. However, `Enterprise` exhibits a larger class imbalance with respect to the other two datasets. Last, while `Enterprise` did not require specific pre-processing, both `MIRAGE-19` and `MIRAGE-22` required a curation to remove *background traffic*—flows created by netd deamon, SSDP, Android google management services and other services unrelated to the target Android apps—and flows having less than 10 packets.

Data folds and normalization. As described in Sec. 3.1, each flow is modeled via a multivariate time series $\mathbf{x}$ consisting of $D = 3$ features (packets size, direction, and IAT) related to the first $T = 20$ packets (applying zero padding in the tail where needed). From the curated datasets we created 80 random 70/15/15 train/validation/test folds. We then processed each train+val split to extract statistics that we used for normalizing the data and to drive the augmentation process. Specifically, we computed both per-coordinate (d, t) and global (i.e., flattening all flows time series into a single array) mean and standard deviation for each class—these statistics provided us the $\sigma^y_{(d,t)}$ and $\sigma^y_{(:,t)}$ needed for the augmentations (see Fig. 1 and Tables 1-3). For IAT, we also computed the global 99th percentile across all classes q^{99}_{iat}. Given a multi-variate input $\mathbf{x}$, we first clip packet size values in the range $[0, 1460]$ and IAT values in the range [1e-7, q^{99}_{iat}]. Due to high skew of IAT distributions, we also log10-scaled the IAT feature values.[6] Last, all features are standardized to provide values $x_{(d,t)} \in [0, 1]$.

Model architecture and training. We rely on a 1d-CNN based neural network architecture with a backbone including 2 ResNet blocks followed by a linear head resulting in a compact architecture of ≈100k parameters. (see Fig. 7 and Listing 1.1 in the appendix for details). Models are trained for a maximum of 500 epochs with a batch size B=1,024 via an AdamW optimizer with a weight decay of 0.0001 and a cosine annealing learning rate scheduler initialized at 0.001. Training is subject to early stopping by monitoring if the validation accuracy does not improve by 0.02 within 20 epochs. We coded our modeling framework using PyTorch and PyTorch Lightning and ran our modeling campaigns on Linux servers equipped with multiple NVIDIA Tesla V100 GPUs. We measured the classification performance via the weighted F1 score considering a reference baseline where training is not subject to augmentations.

5 Results

In this section, we discuss the results of our modeling campaigns closely following the research goals introduced in Sec. 3.

5.1 Augmentations benchmark (G1)

We start by presenting the overall performance of the selected augmentations. Specifically, Table 5 collects results obtained by applying augmentations via

[6] We did not log-scale packet sizes values as we found this can reduce accuracy based on preliminary empirical assessments.

Table 5: Augmentations benchmark **(G1).**

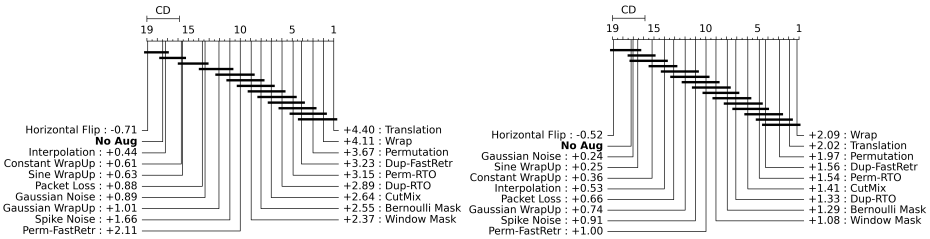
	Augmentation	MIRAGE-19	MIRAGE-22	Enterprise
Baseline	None	75.43±.10	94.92±.07	92.43±.33
Amplitude	Constant WrapUp	0.61±.12	0.36±.09	-0.02±.15
	Gaussian Noise	0.89±.11	0.24±.09	0.15±.14
	Gaussian WrapUp	1.01±.13	0.74±.09	0.24±.12
	Spike Noise	1.66±.12	0.91±.09	0.93±.13
	Sine WrapUp	0.63±.11	0.25±.09	-0.06±.16
Masking	Bernoulli Mask	2.55±.12	1.29±.09	1.25±.16
	Window Mask	2.37±.13	1.08±.09	1.18±.16
Sequence	CutMix	2.65±.13	1.40±.10	-0.21±.10
	Dup-FastRetr	3.23±.13	1.56±.09	0.83±.15
	Dup-RTO	2.89±.13	1.33±.09	0.91±.15
	Horizontal Flip	-0.71±.11	-0.52±.09	-0.88±.15
	Interpolation	0.44±.12	0.53±.10	-0.61±.14
	Packet Loss	0.88±.12	0.66±.09	0.60±.22
	Permutation	3.67±.13	1.97±.09	0.89±.08
	Perm-RTO	3.15±.12	1.54±.09	0.88±.12
	Perm-FastRetr	2.11±.12	1.00±.09	0.74±.26
	Translation	4.40±.13	2.02±.09	0.95±.15
	Wrap	4.11±.13	2.09±.08	0.57±.12

The top-3 best and worst augmentations are color-coded.

Inject with $N_{inject} = 1$ (i.e., each original sample is augmented once)[7] and sampling uniformly the magnitude $\alpha \sim U[0, 1]$. Table 5 shows the average weighted F1 score across 80 runs and related 95th-percentile confidence intervals.

Reference baseline. We highlight that our reference baseline performance for MIRAGE-19 and MIRAGE-22 are *qualitatively* aligned with previous literature that used those datasets. For instance, Table 1 in [14] reports a weighted F1 of 97.89 for a 1d-CNN model when using the first 2,048 payload bytes as input for MIRAGE-22; Figure 1 in [5] instead shows a weighted F1 of ≈75% for 100 packets time series input for MIRAGE-19. Notice however that since these studies use training configurations not exactly identical to ours, a direct comparison with our results should be taken with caution. Yet, despite these differences, we confirm MIRAGE-19 to be a more challenging classification task compared to MIRAGE-22. However, we argue that such a difference is unlikely depending only on the different number of classes (MIRAGE-19 has 20 classes while MIRAGE-22 only 9). This is evident by observing that Enterprise yields very high performance despite having 10× more classes than the other two datasets. We conjecture instead the presence of "cross-app traffic" such as flows generated by libraries/services common across

[7] Since we train the reference baseline with a batch size B=1024, when adding augmentations we instead adopt B=512 (which doubles via injection).



(a) MIRAGE-19 (b) MIRAGE-22

Fig. 3: Augmentations rank and critical distance **(G1).**

multiple apps from the same provider (e.g., apps or services provided by Google or Facebook) and/or the presence of ads traffic,[8] but the datasets raw data is not sufficiently detailed to investigate our hypothesis.

Takeaways. *While the classification tasks complexity is well captured by models performance, it does not necessarily relate to the number of classes or dataset size. These effects are visible only when studying multiple datasets at once, but unfortunately a lot of TC studies focus on individual datasets.*

Augmentations rank. Overall, all augmentations are beneficial except for Horizontal Flip which, as we shall see in Sec. 5.3, breaks class semantics. As expected, not all augmentations provide the same gain and their effectiveness may vary across datasets. Specifically, *sequence* and *masking* better suit our TC tasks.

For a more fine-grained performance comparison, we complement Table 5 results by analyzing augmentations rank via a critical distance by following the procedure described in [12]. Specifically, for each of the 80 modeling runs we first ranked the augmentations from best to worst (e.g., if augmentations A, B, and C yield a weighted F1 of 0.9, 0.7, and 0.8, their associated rankings would be 1, 3, and 2) splitting ties using the average ranking of the group (e.g., if augmentations A, B, and C yield a weighted F1 of 0.9, 0.9 and 0.8, their associated rankings would be 1.5, 1.5, and 3). This process is then repeated across the 80 runs and a global rank is obtained by computing the mean rank for each augmentation. Last, these averages are compared pairwise using a post-hoc Nemenyi test to identify which groups of augmentations are statistically equivalent. This decision is made using a Critical Distance $CD = q_\alpha \sqrt{k(k+1)/6N}$, where q_α is based on the 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. Results are then collected in Fig. 3 where each augmentation is highlighted with its average rank (the lower the better) and horizontal bars connect augmentations that are statistically equivalent. For instance, while Table 5 shows that Translate is the best on average, Fig. 3 shows that {Translate, Wrap, Permutation, Dup-

[8] MIRAGE-22 focuses on video meeting apps which are all from different providers and ads free by design.

FastRetr} are statistically equivalent. We remark that Fig. 3 refers to `MIRAGE-19` and `MIRAGE-22` but similar considerations hold for `Enterprise` as well.

Recall that our training process is subject to an early stop mechanism. Interestingly, we observed that augmentations yielding better performance also present a longer number of training epochs (see Fig. 8 in Appendix). This hints that effective augmentations foster better data representations extraction, although some CV studies also show that early stopping might not necessarily be the best option to achieve high accuracy in some scenarios. An in-depth investigation of these training mechanisms is however out of scope for this paper.

Takeaways. *Augmentations bring benefits that, in absolute scale, are comparable to what is observed in CV literature [27]. Our benchmark shows that TC sequencing and masking augmentations are better options than amplitude augmentations. This confirms previous literature that* implicitly *discarded amplitude augmentations. Finally, despite performance ranks can suggest more performant augmentations (e.g., Translation or Bernoulli mask), agreement between datasets seems more qualitative than punctual (e.g., masking is preferred to sequencing for* `Enterprise`*, but the reverse is true for the other two datasets).*

Sensitivity to magnitude. Most of the augmentations we analyzed are subject to a magnitude α hyper-parameter (see Tables 1-3) that is randomly selected for the results in Table 5. To investigate the relationship between classification performance and augmentation magnitude we selected 3 augmentations among the top performing ones {Translation, Wrap, Permutation} and three among the worst performing {Gaussian Noise, Sine WrapUp, Constant WrapUp}.[9] For each augmentation, we performed 10 modeling runs using magnitude $\alpha = 0.5$ and we contrasted these results with the related runs from the previous modeling campaign. Specifically, by grouping all results we obtained a binary random-vs-static performance comparison which we investigated through a Wilcoxon signed rank sum test that indicated *no statistical difference*, i.e., the selection of magnitude is not a distinctive factor to drive the augmentation performance. The same conclusion holds true when repeating the analysis for each individual augmentation rather than grouping them together.

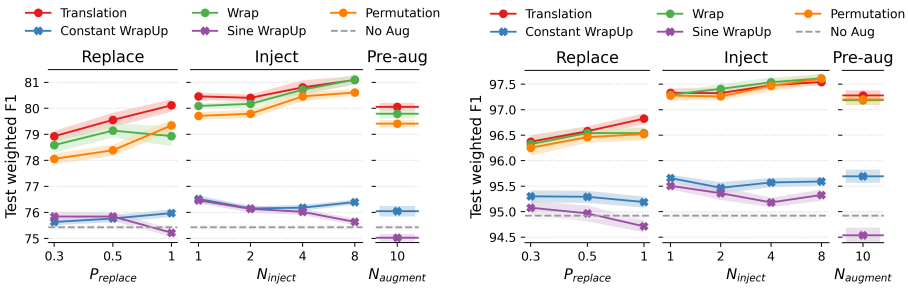
Takeaways. *Although we do not observe any dependency on the augmentation magnitude α, augmentations performance can still be affected by their tuning (as will be discussed further in Sec. 5.3). Unfortunately, this tuning process often relies on a trial-and-error process, making it challenging to operate manually.*

5.2 Training batches composition (G2)

Correctly mixing original with augmented data is an important design choice.

Batching policies. To show this, we considered the three policies introduced in Sec. 3.2: Replace (which randomly substitutes training samples with augmented

[9] We excluded HorizontalFlip as it hurt performance and Interpolation since it does not depend from a magnitude.



(a) MIRAGE-19 (b) MIRAGE-22

Fig. 4: Comparing *Replace*, *Inject* and *Pre-augment* batch creation policies **(G2).**

ones), Inject (which expands batches by adding augmented samples), and Pre-augment (which expands the whole training set before the training start).[10] Batching policies are compared against training without augmentations making sure that each training step has the same batch size B=1,204.[11] Based on Sec. 5.1 results, we limited our comparison to {Translation, Wrap, Permutation} against {Sine WrapUp, Constant WrapUp} as representative of good and poor augmentations across the three datasets under study. We configured Replace with $P_{replace} \in \{0.3, 0.5, 1\}$, Inject with $N_{inject} \in \{1, 2, 4, 8\}$ and augmented each training sample 10 times for Pre-augment. Fig. 4 collects the results with lines showing the average performance while shaded areas correspond to 95th percentile confidence intervals. Overall, top-performing augmentations (● marker) show a positive trend—the higher the volume of augmentations the better the performance—while poor-performing augmentations (× marker) have small deviations from the baseline (dashed line). Based on performance, we can order *Replace* < *Pre-augment* < *Inject*, i.e., the computationally cheaper Pre-augment is on par with the more expensive Replace when $P_{replace} = 1$ but Inject is superior to both alternatives.

Takeaways. *On the one hand, Inject shows a positive trend that perhaps continues beyond* $N_{inject} > 8$.[12] *On the other hand, the performance gain may be too little compared to the computational cost when using many augmentations. For instance,* $N_{inject} = 8$ *requires 3× longer training compared to* $N_{inject} = 1$.

Class-weighted sampling. TC datasets are typically imbalanced (see Table 4). It is then natural to wonder if/how augmentations can help improve performance

[10] Based on our experience on using code-bases related to publications, we were unable to pinpoint if any of those techniques is preferred in CV literature.

[11] For instance, when $N_{inject} = 1$, a training run needs to be configured with B=512 as the mini-batches size doubles via augmentation.

[12] The limit of our experimental campaigns were just bounded by training time and servers availability so it is feasible to go beyond the considered scenarios.

Table 6: Impact of class-weighted sampler on MIRAGE-19 **(G2)**.

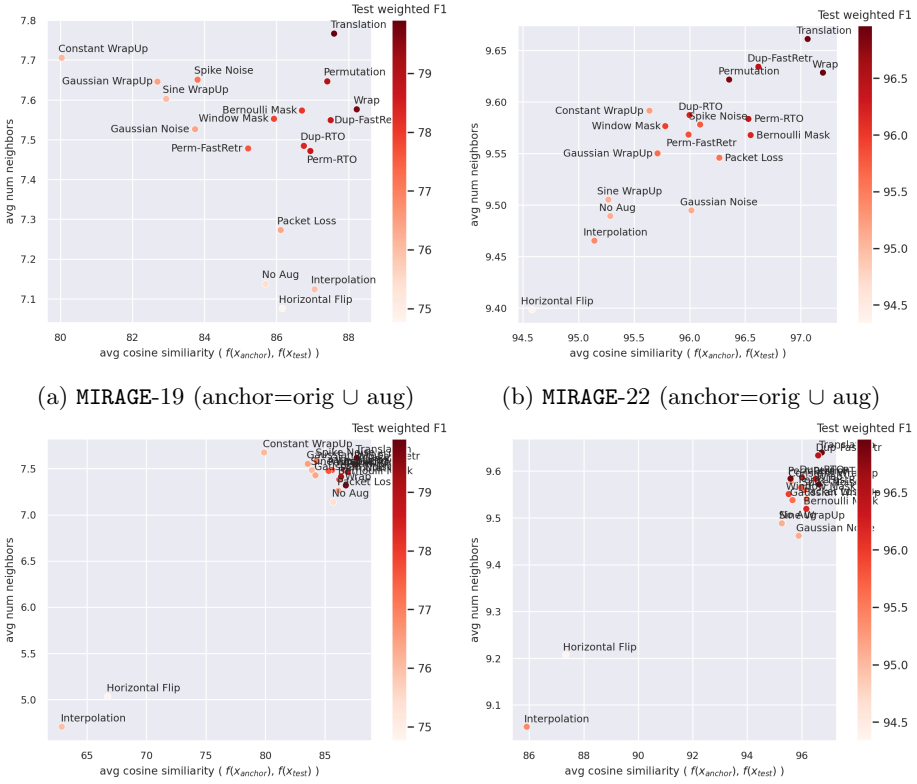
		Majority classes			Minority classes		
	Cls samp.	*Pre*	*Rec*	*weight F1*	*Pre*	*Rec*	*weight F1*
No Aug	with	83.90±.21	81.01±.21	82.36±.14	56.63±.38	60.78±.26	58.18±.21
	without	81.60±.23	82.93±.19	82.16±.12	62.29±.48	58.02±.38	59.78±.27
	diff	2.30±.32	-1.92±.28	0.20±.20	-5.66±.60	2.76±.46	-1.60±.35
Translation	with	89.12±.09	84.26±.11	86.43±.08	60.71±.24	68.64±.17	63.65±.19
	without	85.36±.14	86.73±.10	85.86±.09	69.69±.25	64.14±.25	66.20±.22
	diff	3.77±.06	-2.48±.02	0.57±.02	-8.98±.04	4.50±.09	-2.55±.05

for classes with fewer samples, namely *minority classes*. While the batching policies discussed do not alter the natural distribution of the number of samples per class, alternative techniques like Random Over Sampling (ROS) and Random Under Sampling (RUS) allow to replicate/drop samples for minority/majority classes [23]. A *class-weighted sampler* embodies a more refined version of those mechanisms and composes training mini-batches by selecting samples with a probability inversely proportional to the classes size—each training epoch results in a balanced dataset. When combined with augmentations, this further enhance minority classes variety.

The adoption of a class-weighted sampler seems a good idea in principle. Yet, the enforced balancing in our experience leads to conflicting results. We showcase this in Table 6 where we show Precision, Recall, and weighted F1 for 20 runs trained with/without a weighted sampler and with/without Translation (selected as representative of a good augmentation across datasets). We break down the performance between majority and minority classes and report per-metric differences when using or not the weighted sampler. The table refers to MIRAGE-19 but similar results can be obtained for the other datasets. Ideally, one would hope to observe only positive differences with larger benefits for minority classes. In practice, only the Recall for minority classes improves and overall we observe a poorer weighted F1 (-0.26 across all classes). By investigating misclassifications, we found that majority classes are more confused with minority classes and when introducing augmentations those effects are further magnified.
Takeaways. *Paying too much attention to minority classes can perturb the overall classifier balance, so we discourage the use of class-weighted samplers.*

5.3 Latent space geometry (G3)

Table 5 allows to identify effective augmentations bringing significant benefits in terms of model performance. However, to understand the causes behind the performance gaps we need to investigate how original, augmented, and test samples relate to each other.



(a) `MIRAGE`-19 (anchor=orig ∪ aug)

(b) `MIRAGE`-22 (anchor=orig ∪ aug)

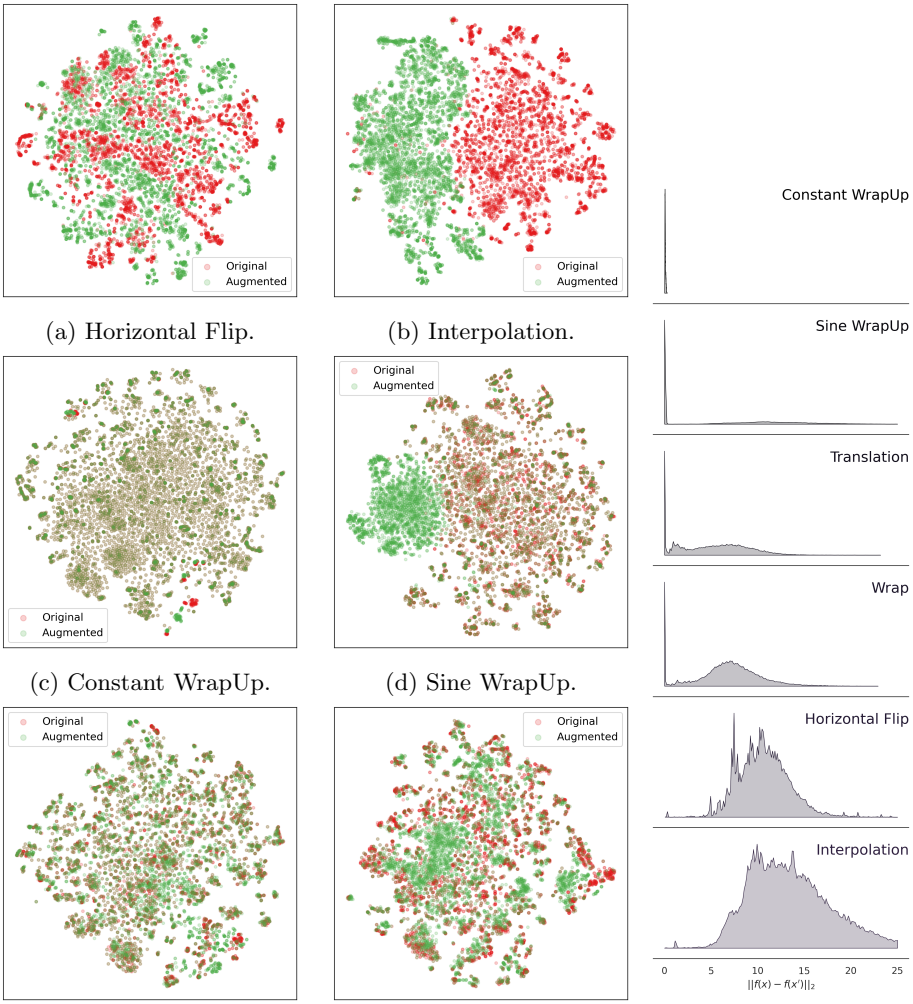
(c) `MIRAGE`-19 (anchor=aug)

(d) `MIRAGE`-22 (anchor=aug)

Fig. 5: Investigating train, augmented and test samples relationships **(G3)**.

Augmented-vs-Test samples. We start our analysis by taking the point of view of the test samples. Specifically, we investigated which type of points are found in the "neighbourhood" of a test sample. To do so, we started creating "true anchors" by projecting both the original training data and 5 augmentations of each training sample—these anchors are "proxy" of what is presented to the model during training. Then we projected the test samples and looked for the closest 10 anchors (based on cosine similarity) of each test sample. Finally, we counted how many of the 10 anchors share the same label as the test samples. Results for each augmentation are reported in Fig. 5 for `MIRAGE`-19 and `MIRAGE`-22 (similar results holds for `Enterprise`) as a scatter plot where the coordinates of each point correspond to the average number of anchors with the correct label found and their average cosine similarity with respect to the test sample. Each augmentation is color-coded with respect to its weighted F1 score.

Despite both metrics vary in a subtle range, such variations suffice to capture multiple effects. First of all, considering the layout of the scatter plot, we expected good transformations to be placed in the top-right corner. This is in-



(a) Horizontal Flip. (b) Interpolation. (c) Constant WrapUp. (d) Sine WrapUp. (e) Translation. (f) Wrap. (g) Euclidean dist. KDE.

Fig. 6: Comparing original and augmented samples in the latent space **(G3)**.

deed the case as presented in Fig. 5 (a-b) where darker colors (higher weighted F1) concentrate in the top-right corner. However, while MIRAGE-22 (Fig. 5(a)) shows a linear correlation between the two metrics, MIRAGE-19 (Fig. 5(a)) shows outliers, most notably Horizontal Flip, Interpolation, and Constant Wrapup.

Fig. 5 (c-d) complement the analysis by showing results when considering only augmented samples as anchors. Differently from before, now Horizontal Flip and Interpolation are found to be the most dissimilar to the test samples—this is signaling that augmentations are possibly disrupting class semantics, i.e., they are introducing unnecessary high variety.

Last, for each test sample we looked at the closest augmented anchor and the closest original sample anchor with the same label. The average ratio of those pairwise distances is centered around 1—augmented samples "mimic" test samples as much as the original samples do.

Takeaways. *Top-performing augmentations do not better mimic test samples compared to original samples. Rather, they help training the feature extractor $f(\cdot)$ so that projected test samples are found in neighborhood of points likely to have the expected label.*

Augmented-vs-Original samples. We complement the previous analysis by investigating original $\mathbf{x}$ and augmented $\mathbf{x}'$ samples relationships. Differently from before, for this analysis original samples are augmented once. Then all points are projected in the latent space $f(\mathbf{x})$ and $f(\mathbf{x}')$ and visualized by means of a 2d t-SNE projection.[13] We also compute the Kernel Density Estimation (KDE) of the Euclidean distance across all pairs. Figure 6 presents the results for 2 top-performing (Translate, Wrap) and 4 poor-performing (Constant Wrapup, Interpolation, Sine Wrapup, Horizontal Flip) augmentations for `MIRAGE`-19. Points in the t-SNE charts are plotted with alpha transparency, hence color saturation highlights prevalence of either augmented or original samples.

Linking back to the previous observations about Horizontal Flip and Interpolation, results now show the more "aggressive" nature of Interpolation—the t-SNE chart is split vertically with the left (right) side occupied by augmented (original) samples only and the Euclidean distance KDEs show heavier tails. By recalling their definition, while it might be easy to realize why Horizontal Flip is a poor choice—a client will never observe the end of the flow before seeing the beginning, hence they are too artificious—it is difficult to assess a priori the effect of Interpolation. Overall, both augmentations break class semantics.

At the opposite side of the performance range we find augmentations like Sine WrapUp and Constant WrapUp. From Fig. 6 we can see that both introduce little-to-no variety—the Euclidean distance distributions are centered around zero. That said, comparing their t-SNE charts we can still observe a major difference between the two transformations which relates to their design. Specifically, Constant WrapUp is applied only to IAT and introduces negligible modifications to the original samples. Conversely, Sine WrapUp is applied on either packet size or IAT. As for Constant WrapUP, the changes to IAT are subtle, while variations of packet size lead to generating an extra "mode" (notice the saturated cluster of points on the left side of the t-SNE plot). In other words, besides the design of the augmentation itself, identifying a good parametrization is very challenging and in this case is also feature-dependent.

Compared to the previous, Translate and Wrap have an in-between behavior—the body of the KDEs show distances neither too far nor too close and the t-SNE charts show a non-perfect overlap with respect to the original samples. Overall, both these augmentations show positive signs of good sample variety.

[13] Our model architecture uses a latent space of 256 dimensions (see Listing 1.1) which the t-SNE representation compresses into a 2d space.

Table 7: Combining augmentations **(G4)**.

	Augmentation	MIRAGE-19	MIRAGE-22
Baseline	No Aug	75.43±.10	94.92±.07
Single	Translation	4.40±.13	2.02±.09
	Wrap	4.11±.13	2.09±.08
	Permutation	3.67±.13	1.97±.09
Combined	Ensemble	4.44±.12	2.18±.09
	RandomStack	4.17±.12	2.18±.09
	MaskedStack ($p = 0.3$)	4.45±.13	2.26±.09
	MaskedStack ($p = 0.5$)	4.60±.15	2.24±.09
	MaskedStack ($p = 0.7$)	4.63±.14	2.18±.10

Takeaways. *Effective transformations operate in a "sweet spot": they neither introduce too little variety—traditional policies like Random Over Sampling (ROS) and Random Under Sampling (RUS) [23] are ineffective—nor they break classes semantic by introducing artificial "modes".*

5.4 Combining augmentations (G4)

We conclude our analysis by analyzing the impact of combining different augmentations. For this analysis, we selected 3 top-performing augmentations and compared their performance when used in isolation against relying on *Ensemble*, *RandomStack* and *MaskedStack* (see Sec. 3.4). Table 7 collects results obtained from 80 modeling runs for each configuration. Overall, mixing multiple augmentations is beneficial but gains are small, i.e., <1%.

Takeaways. *While one would expect that mixing good augmentations can only improve performance, we note that also CV literature is split on the subject. If on the one hand combining augmentations is commonly done in training pipelines, recent literature shows that such combinations bring marginal benefits [27].*

6 Discussion and Conclusions

In this work we presented a benchmark of hand-crafted DA for TC covering multiple dimensions: a total of 18 augmentations across 3 families, with 3 policies for introducing augmentations during training, investigating the classification performance sensitivity with respect to augmentations magnitude and class-weighted sampling across 3 datasets with different sizes and number of classes. Overall, our results confirm what previously observed in CV literature—*augmentations are beneficial even for large datasets, but in absolute terms the gains are dataset-dependent.* While from a qualitative standpoint, sequence and mask augmentations are better suited for TC tasks than amplitude augmentations, no single augmentation is found superior to alternatives and combining them (via stacking or ensembling), even when selecting top-performing ones, marginally improves performance compared to using augmentations in isolation.

Last, by investigating the models latent space geometry, we confirm that *effective augmentations provide good sample variety* by creating samples that are neither too similar nor too different from the original ones which fosters better data representations extraction (as suggested by the longer training time).

Despite the multiple dimensions covered, our work suffers from some limitations. Most notably, it would be desirable to include the larger and more recent `CESNET-TLS22` [25] and `CESNET-QUIC23` [24] datasets but such expansion requires large computational power.[14] Still related to using large datasets, we can also envision more experiments tailored to investigate the relationship between datasets size and augmentations. For instance, one could sample down a large dataset (e.g., by randomly selecting 1% or 10% of the available samples) and investigate if augmentations result more effective with the reduced datasets. In particular, since Inject shows a positive trend with respect to its intensity N_{inject} we hypothesize that by augmenting a small dataset one can achieve the same performance as using larger datasets—showing these effects are clearly relevant for TC as collecting and releasing large datasets is currently a pain point. Last, our campaigns rely only a CNN-based architecture while assessing DA with other architectures (e.g., Transformer-based for time series [45]) is also relevant.

Ultimately, DA modeling campaigns as the one we performed require operating with a grid of configurations and parameters—it is daunting to explore the design space by means of brute forcing all possible scenarios. While domain knowledge can help in pruning the search space, it can also prevent from considering valuable alternatives. For instance, recall that Xie et al. [46] suggest to use augmentations inspired by TCP protocol dynamics. According to our benchmark, these augmentations are indeed among the top performing ones, yet not necessarily the best ones—navigating the search space results in a balancing act between aiming for qualitative and quantitative results.

We identify two viable options to simplify the design space exploration. On the one hand, re-engineering the augmentations so that their parametrization is discovered during training might resolve issues similar to what observed for Sine Wrap (see Sec. 5.3). On the other hand, a more efficient solution would be to rely on generative models avoiding the burden of designing hand-crafted augmentations. More specifically, we envision a first exploration based on conditioning the generative models on the latent space properties learned via hand-crafted DA (e.g., the distance between original and augmented samples should be in the "sweet spot"). Then, we could target the more challenging scenario of training unconditionally and verify if effective representations are automatically learned.

Overall, we believe that the performance observed in our experimental campaigns might still represent a lower bound and extra performance improvements could be achieved via generative models. We call for the research community to join us in our quest for integrating DA and improve TC performance.

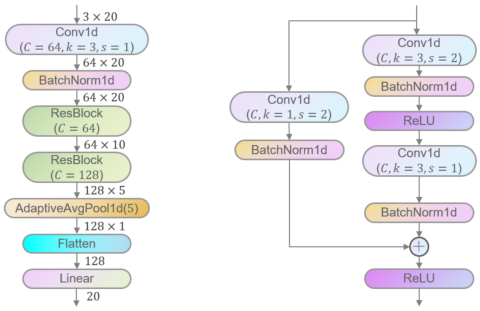
[14] For reference, models trained on `Enterprise` can take up to 6 hours. Since CESNET datasets contains 100× the number of samples of `Enterprise`, performing a thorough exploration of the DA design space is extremely resource demanding.

References

1. Additional material for the paper "Rosetta: Enabling Robust TLS Encrypted Traffic Classification in Diverse Network Environments with TCP-Aware Traffic Augmentation". https://cloud.tsinghua.edu.cn/f/7f250d2ffce8404b845e/?dl=1.
2. Mimetic: Mobile encrypted traffic classification using multimodal deep learning. Computer Networks **165** (2019)
3. Aceto, G., Ciuonzo, D., Montieri, A., Persico, V., Pescapè, A.: Mirage: Mobile-app traffic capture and ground-truth creation. In: IEEE International Conference on Computing, Communication and Security (ICCCS) (2019)
4. Akbari, I., Salahuddin, M.A., Ven, L., Limam, N., Boutaba, R., Mathieu, B., Moteau, S., Tuffin, S.: A look behind the curtain: Traffic classification in an increasingly encrypted web. ACM Measurement and Analysis of Computing Systems **5**(1) (2021)
5. Bovenzi, G., Yang, L., Finamore, A., Aceto, G., Ciuonzo, D., Pescapè, A., Rossi, D.: A first look at class incremental learning in deep learning mobile traffic classification. In: IFIP Traffic Measurement and Analysis (TMA) (2021)
6. Burg, M.F., Wenzel, F., Zietlow, D., Horn, M., Makansi, O., Locatello, F., Russell, C.: A data augmentation perspective on diffusion models and retrieval. arXiv:2304.10253 (2023)
7. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: SMOTE: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research **16**, 321–357 (2002)
8. Chen, T., Kornblith, S., Norouzi, M., Hinton, G.: A simple framework for contrastive learning of visual representations. arXiv:2002.05709 (2020)
9. Cubuk, E.D., Zoph, B., Mane, D., Vasudevan, V., Le, Q.V.: Autoaugment: Learning augmentation policies from data. arXiv:1805.09501 (2019)
10. Cubuk, E.D., Zoph, B., Shlens, J., Le, Q.V.: Randaugment: Practical automated data augmentation with a reduced search space. arXiv:1909.13719 (2019)
11. Cubuk, E.D., Dyer, E.S., Lopes, R.G., Smullin, S.: Tradeoffs in data augmentation: An empirical study. In: International Conference on Learning Representations (ICLR) (2021)
12. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. The Journal of Machine learning research **7**, 1–30 (2006)
13. Eldele, E., Ragab, M., Chen, Z., Wu, M., Kwoh, C.K., Li, X., Guan, C.: Time-series representation learning via temporal and contextual contrasting. arXiv:2106.14112 (2021)
14. Guarino, I., Aceto, G., Ciuonzo, D., Montieri, A., Persico, V., Pescape, A.: Classification of communication and collaboration apps via advanced deep-learning approaches. In: IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD) (2021)
15. Guarino, I., Wang, C., Finamore, A., Pescape, A., Rossi, D.: Many or few samples? comparing transfer, contrastive and meta-learning in encrypted traffic classification. In: IFIP Traffic Measurement and Analysis (TMA) (2023)
16. Han, H., Wang, W., Mao, B.: Borderline-smote: A new over-sampling method in imbalanced data sets learning. Advances in Intelligent Computing (2005)
17. He, H., Bai, Y., Garcia, E.A., Li, S.: Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In: International Joint Conference on Neural Networks (IJCNN) (2008)

18. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. arXiv:1512.03385 (2015)
19. Hong, Y., Zhang, J., Sun, Z., Yan, K.: Safa: Sample-adaptive feature augmentation for long-tailed image classification. In: European Conference on Computer Vision (ECCV) (2022)
20. Horowicz, E., Shapira, T., Shavitt, Y.: A few shots traffic classification with mini-flowpic augmentations. In: ACM Internet Measurement Conference (IMC) (2022)
21. Jain, S., Addepalli, S., Sahu, P.K., Dey, P., Babu, R.V.: Dart: Diversify-aggregate-repeat training improves generalization of neural networks. In: Computer Vision and Pattern Recognition (CVPR) (2023)
22. Jiang, X., Liu, S., Gember-Jacobson, A., Schmitt, P., Bronzino, F., Feamster, N.: Generative, high-fidelity network traces. In: ACM Workshop on Hot Topics in Networks (HotNets) (2023)
23. Johnson, J.M., Khoshgoftaar, T.M.: Survey on deep learning with class imbalance. Journal of Big Data **6** (2019)
24. Luxemburk, J., Hynek, K., Cejka, T.: Encrypted traffic classification: the quic case. In: IFIP Traffic Measument and Analysis (TMA) (2023)
25. Luxemburk, J., Čejka, T.: Fine-grained tls services classification with reject option. Computer Networks **220**, 109467 (2023)
26. Mumuni, A., Mumuni, F.: Data augmentation: A comprehensive survey of modern approaches. Array **16**, 100258 (2022)
27. Müller, S.G., Hutter, F.: Trivialaugment: Tuning-free yet state-of-the-art data augmentation. In: International Conference on Computer Vision (ICCV) (2021)
28. Odena, A., Olah, C., Shlens, J.: Conditional image synthesis with auxiliary classifier gans. arXiv:1610.09585 (2017)
29. Pacheco, F., Exposito, E., Gineste, M., Baudoin, C., Aguilar, J.: Towards the deployment of machine learning solutions in network traffic classification: A systematic survey. IEEE Communications Surveys & Tutorials **21**(2), 1988–2014 (2019)
30. Pöppelbaum, J., Chadha, G.S., Schwung, A.: Contrastive learning based self-supervised time-series analysis. Applied Soft Computing **117**, 108397 (2022)
31. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. arXiv:1506.02640 (2016)
32. Rezaei, S., Liu, X.: How to achieve high classification accuracy with just a few labels: A semi-supervised approach using sampled packets. In: IEEE Industrial Conference Advances in Data Mining - Applications and Theoretical Aspects (ICDM) (2019)
33. Schuhmann, C., Beaumont, R., Vencu, R., Gordon, C.W., Wightman, R., Cherti, M., Coombes, T., Katta, A., Mullis, C., Wortsman, M., Schramowski, P., Kundurthy, S.R., Crowson, K., Schmidt, L., Kaczmarczyk, R., Jitsev, J.: LAION-5b: An open large-scale dataset for training next generation image-text models. In: Neural Information Processing Systems (NeurIPS) - Datasets and Benchmarks Track (2022)
34. Shen, M., Ye, K., Liu, X., Zhu, L., Kang, J., Yu, S., Li, Q., Xu, K.: Machine learning-powered encrypted network traffic analysis: A comprehensive survey. IEEE Communications Surveys & Tutorials **25**(1), 791–824 (2023)
35. Shen, R., Bubeck, S., Gunasekar, S.: Data augmentation as feature manipulation. arXiv:2203.01572 (2022)
36. Shorten, C., Khoshgoftaar, T.M.: A survey on image data augmentation for deep learning. Journal of Big Data **6**(1), 1–48 (2019)

37. Sivaroopan, N., Madarasingha, C., Muramudalige, S., Jourjon, G., Jayasumana, A., Thilakarathna, K.: Synig: Synthetic network traffic generation through time series imaging. In: IEEE Local Computer Networks (LCN) (2023)
38. Sivaroopan, N., Bandara, D., Madarasingha, C., Jourjon, G., Jayasumana, A., Thilakarathna, K.: Netdiffus: Network traffic generation by diffusion models through time-series imaging. arXiv:2310.04429 (2023)
39. Towhid, M.S., Shahriar, N.: Encrypted network traffic classification using self-supervised learning. In: IEEE International Conference on Network Softwarization (NetSoft) (2022)
40. Trabucco, B., Doherty, K., Gurinas, M., Salakhutdinov, R.: Effective data augmentation with diffusion models. arXiv:2302.07944 (2023)
41. Wang, P., Li, S., Ye, F., Wang, Z., Zhang, M.: Packetcgan: Exploratory study of class imbalance for encrypted traffic classification using cgan. In: International Conference on Communications (ICC) (2020)
42. Wang, Y., Pan, X., Song, S., Zhang, H., Wu, C., Huang, G.: Implicit semantic data augmentation for deep networks. arXiv:1909.12220 (2020)
43. Wang, Z., Wang, P., Zhou, X., Li, S., Zhang, M.: Flowgan:unbalanced network encrypted traffic identification method based on gan. In: Conference on Parallel and Distributed Processing with Applications, Big Data and Cloud Computing, Sustainable Computing and Communications, Social Computing and Networking (ISPA/BDCloud/SocialCom/SustainCom) (2019)
44. Wen, Q., Sun, L., Yang, F., Song, X., Gao, J., Wang, X., Xu, H.: Time series data augmentation for deep learning: A survey. In: International Joint Conference on Artificial Intelligence (IJCAI) (2021)
45. Wen, Q., Zhou, T., Zhang, C., Chen, W., Ma, Z., Yan, J., Sun, L.: Transformers in time series: A survey. arXiv:2202.07125 (2023)
46. Xie, R., Cao, J., Dong, E., Xu, M., Sun, K., Li, Q., Shen, L., Zhang, M.: Rosetta: Enabling robust TLS encrypted traffic classification in diverse network environments with TCP-Aware traffic augmentation. In: USENIX Security Symposium (Security) (2023)
47. Yang, H., Yu, H., Sano, A.: Empirical evaluation of data augmentations for biobehavioral time series data with deep learning. arXiv:2210.06701 (2022)
48. Yin, C., Zhu, Y., Liu, S., Fei, J., Zhang, H.: An enhancing framework for botnet detection using generative adversarial networks. In: IEEE International Conference on Artificial Intelligence and Big Data (ICAIBD) (2018)
49. Yu, H., Sano, A.: Semi-supervised learning and data augmentation in wearable-based momentary stress detection in the wild. arXiv:2202.12935 (2022)
50. Yue, Z., Wang, Y., Duan, J., Yang, T., Huang, C., Tong, Y., Xu, B.: Ts2vec: Towards universal representation of time series. In: Proceedings of the Association for the Advancement of Artificial Intelligence Conference (AAAI) (2022)
51. Yun, S., Han, D., Oh, S.J., Chun, S., Choe, J., Yoo, Y.: Cutmix: Regularization strategy to train strong classifiers with localizable features. arXiv:1905.04899 (2019)
52. Zhang, H., Cisse, M., Dauphin, Y.N., Lopez-Paz, D.: mixup: Beyond empirical risk minimization. arXiv:1710.09412 (2018)
53. Zou, D., Cao, Y., Li, Y., Gu, Q.: The benefits of mixup for feature learning. arXiv:2303.08433 (2023)



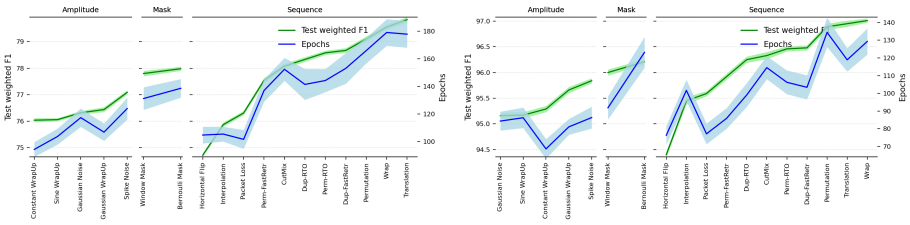
(a) Overall (b) ResBlock

C: number of output channels, k: kernel size; s: stride

Fig. 7: Model architecture.

Listing 1.1: Model architecture printout (MIRAGE-19, 20 classes)

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv1d-1               [-1, 64, 20]             576
       BatchNorm1d-2               [-1, 64, 20]             128
            Conv1d-3               [-1, 64, 10]          12,288
       BatchNorm1d-4               [-1, 64, 10]             128
            Conv1d-5               [-1, 64, 10]          12,288
       BatchNorm1d-6               [-1, 64, 10]             128
            Conv1d-7               [-1, 64, 10]           4,096
       BatchNorm1d-8               [-1, 64, 10]             128
            Conv1d-9              [-1, 128, 5]          24,576
      BatchNorm1d-10              [-1, 128, 5]             256
           Conv1d-11              [-1, 128, 5]          49,152
      BatchNorm1d-12              [-1, 128, 5]             256
           Conv1d-13              [-1, 128, 5]           8,192
      BatchNorm1d-14              [-1, 128, 5]             256
AdaptiveAvgPool1d-15              [-1, 128, 1]               0
           Linear-16                   [-1, 20]           2,580
================================================================
Total params: 115,028
Trainable params: 115,028
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.09
Params size (MB): 0.44
Estimated Total Size (MB): 0.53
----------------------------------------------------------------
```



(a) MIRAGE-19 (b) MIRAGE-22

Fig. 8: Comparing performance improvement and training length.