A Study of Deep Learning for Network Traffic Data Forecasting

Benedikt Pfülb[1], Christoph Hardegen[1],
Alexander Gepperth[1] and Sebastian Rieger[1]

University of Applied Sciences Fulda, Leipziger Straße 123, 36037 Fulda, Germany
`http://www.hs-fulda.de`
{`benedikt.pfuelb,christoph.hardegen,alexander.gepperth,sebastian.rieger`}
`@cs.hs-fulda.de`

Abstract. We present a study of deep learning applied to the domain of network traffic data forecasting. This is a very important ingredient for network traffic engineering, e.g., intelligent routing, which can optimize network performance, especially in large networks. In a nutshell, we wish to predict, in advance, the bit rate for a transmission, based on low-dimensional connection metadata ("flows") that is available whenever a communication is initiated. Our study has several genuinely new points: First, it is performed on a large dataset (≈50 million flows), which requires a new training scheme that operates on successive blocks of data since the whole dataset is too large for in-memory processing. Additionally, we are the first to propose and perform a more fine-grained prediction that distinguishes between low, medium and high bit rates instead of just "mice" and "elephant" flows. Lastly, we apply state-of-the-art visualization and clustering techniques to flow data and show that visualizations are insightful despite the heterogeneous and non-metric nature of the data. We developed a processing pipeline to handle the highly non-trivial acquisition process and allow for proper data preprocessing to be able to apply DNNs to network traffic data. We conduct DNN hyper-parameter optimization as well as feature selection experiments, which clearly show that fine-grained network traffic forecasting is feasible, and that domain-dependent data enrichment and augmentation strategies can improve results. An outlook about the fundamental challenges presented by network traffic analysis (high data throughput, unbalanced and dynamic classes, changing statistics, outlier detection) concludes the article.

Keywords: DNN · Incremental Learning · Network Traffic Engineering.

1 Introduction

This article is in the context of computer network traffic forecasting. We focus on using deep neural networks (DNNs). More precisely, we investigate how DNNs can predict, in advance, the approximate bit rate of a computer network communication. This is modeled as a classification task with three classes (low, medium and high). The key idea here is to take this decision based only on the metadata of the communication, which are represented, in their most basic form, as a 5-tuple: source and destination IP address, source and destination port as well as the transport protocol, e.g., TCP or UDP. An example of the flow metadata as well as the classification task is depicted in Fig. 1.

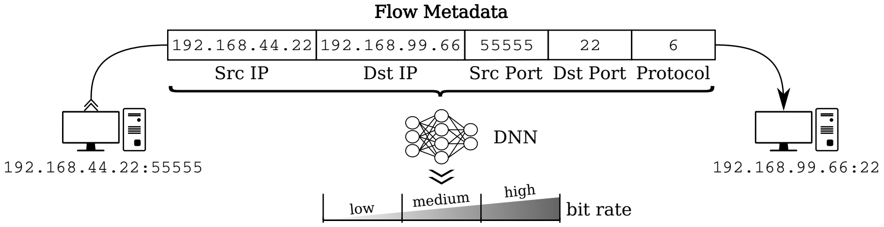


Fig. 1: Overview of the principal task of network traffic forecasting. Upon establishment of an IP-based network communication between two computers, the metadata (5-tuple) is supplied to a trained DNN that forecasts the bit rate (low, medium and high) for this communication. This is done before any data is exchanged. In order to train the DNN, target values have to be obtained after a communication is terminated.

The motivation for investigating this kind of classification problem stems from the field of software-defined networking (SDN). While traditional and still most prevalent network routing algorithms are primarily based on the destination address, SDN-like techniques enable dynamic determination of paths based on traffic characteristics. For example, routers can typically choose between several paths to forward network traffic to a specific destination. On the one hand, in the case of paths with unequal costs, using only the optimal path could cause congestion while alternative paths are underutilized. On the other hand, using the hash of a 5-tuple to decide between multiple equal-cost paths might lead to unequal load balancing because the amount of transmitted data cannot be considered in advance. Also, the link cannot easily be changed during the communication. Therefore, predicting the bit rate of a communication beforehand is of high value for the routing and load balancing process.

1.1 Problem Formulation and Approach of the Article

Challenges The principal immediate challenges for machine learning in network traffic forecasting raised and addressed in this article are as follows:

- data acquisition: Here, one encounters difficulties creating the technical infrastructure (i.e., administrative access to network devices, handling large amounts of data resulting from capturing the network traffic) and the fact that metadata contain sensitive information, requiring an anonymization strategy that preserves information content and relations. Furthermore, the encoding of metadata into a form that is suitable for DNNs and the generation of target values is essential.

- regression problem: Network traffic forecasting is essentially a regression problem as a continuous and highly variable quantity (the bit rate) needs to be predicted, which is a challenging task that must be simplified suitably.
- class imbalance: Communications transmitting very few data are much more frequent than those transferring huge amounts of data[3]. The distribution regarding the bit rate as target value can be expected to change over time.
- concept drift: The statistics of the problem may be time-dependent, e.g., depending on the day of week, the time of day, the season, technical changes, etc. A DNN classifier trained on day X may therefore not be suited to classify metadata collected on day $Y \neq X$. We are therefore dealing with a problem where continual re-training must be conducted while retaining previous knowledge (see [4] for a recent review on this kind of training paradigm).
- big data setting: The amount of flows is so high, and their variability so significant, that DNN training on a representative training set can no longer be performed in-memory. In our scenario, the network devices we accessed to collect data delivered 57 million records in 8 hours (about 15 GB of raw data respectively ∅ 2 000 flows per second, including the 5-tuple).

Approach In order to address these challenges, we first of all treat DNN training as a streaming problem by dividing all collected metadata into blocks of 100 000 flows each. Training and evaluation are then conduced in a semi-streaming fashion, starting with the first block and subsequently passing to following ones, with all relevant preprocessing operations being performed block-wise. Concept drift is thus incorporated into DNN training although it cannot be completely compensated. The class imbalance problem is currently fixed by different class balancing mechanisms, since the whole reference dataset[1] is known prior to DNN training. This will have to be replaced by more generic solutions in the future. Lastly, we transform the regression problem into a classification problem with three classes, thus balancing the need for precision and complexity of the problem.

1.2 Related Work

Network traffic forecasting with machine learning techniques is a field (see [3] for a review) that is receiving increased attention, probably due to the recent advances in machine learning techniques, notably deep learning models. From a machine learning point of view, many recent articles can be grouped according to whether they conduct online or offline learning on streaming network data, what machine learning models they employ in general, what dataset they operate on and whether they systematically investigate the effects of data enrichment. To the best of our knowledge, all related works operate on datasets of around 1 000 000 flows which is significantly smaller than the dataset we use in this study, and thereby avoid “big data” issues like the necessity to perform learning in blocks. Furthermore, related works reduce the network traffic forecasting problem to a binary classification into “mice” and “elephant” flows.

[1] Our anonymized dataset is available upon request.

In [5], the authors apply online and offline learning methods (Multi-Layer Perceptron, Gaussian Process Regression and Online Bayesian Moment Matching). The problem is treated as a two-class classification problem using three different datasets, one self-created (not available) and the others from other authors [1]. No data enrichment is performed, however information about the first three exchanged packets is used in addition to a flow's 5-tuple as a basis for classification, which differs from our approach that does not consider such information. In [9], purely offline learning with two-class decision tree classifiers is performed on the "Wide" dataset and a self-created one (not available) coming from a data center, also without data enrichment. In [8], semi-supervised SVMs are trained in an offline fashion to solve a two-class problem using a simple form of data enrichment. Evaluations are conducted on a dataset of approximately 1 000 000 flows, "captured by the Broadband Communication Research Group in UPC, Barcelona, Spain" (no reference given, no data available). [6] use offline SVM training on two datasets captured on Chinese university campuses (no reference given, not available), and experiment extensively with feature selection schemes, however based only on the basic 5-tuple information. Another interesting albeit not directly related application of machine learning is the routing of flows itself (see [7]).

1.3 Contribution of the Article

Overall, this study shows that fine-grained network traffic forecasting using three classes with DNNs is feasible, and that it can be performed in a "big data" setting, operating on separate data blocks sequentially. We furthermore investigate the effects of data enrichment beyond the basic 5-tuple information, while also dealing with anonymization and privacy issues. Lastly, we show that modern data visualization and clustering techniques can be readily applied to network traffic data in order to gain deeper insights into the structure of the problems and to "debug" machine learning solutions.

2 Flow Data Pipeline

We introduce a flow data pipeline (see Fig. 2) that is responsible for collecting the network traffic and producing a dataset consisting of flows describing communications. Data collection and the first parts of the data preparation (enrichment and anonymization) are entirely performed within our data center to ensure privacy (supported by the administration). The codebase of the pipeline is publicly available in our repository[2].

[2] `https://gitlab.informatik.hs-fulda.de/flow-data-ml`

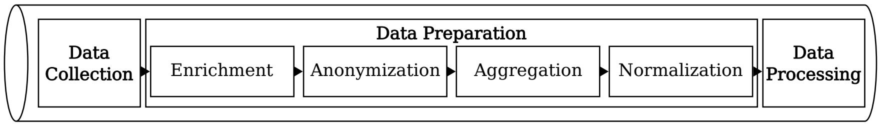


Fig. 2: Overview of the stages of the flow data pipeline: data collection, preparation and processing. At first, flow records are collected. Before IP addresses are anonymized, further related metadata is added during the enrichment phase. Afterwards, the fusion of individual *flow records* is applied to aggregate *flow entries*. Flow data is normalized and stored as a dataset that is used for DNN training in the data processing phase.

2.1 Data Collection

A flow is understood to be the history of a single transmission between two endpoints, from establishment to termination (only metadata). In particular, flows are partly characterized by the 5-tuple. Flows may include additional metadata, e.g., the duration or number of transferred bytes.

Flow data is collected from the networks at Fulda University of Applied Sciences. We export network flow data (57 million flow records) using the NetFlow standard from the two core network devices in our university data center during a continuous time interval of 8 hours on a weekday (02/15/2019 9:00 AM - 5:00 PM). These core components connect multiple subnets from data center, laboratory, WiFi and campus networks. Collecting data from these diverse networks ensures realistic traffic characteristics and patterns to be used for the subsequent data analysis and network flow prediction. For example, collected traffic patterns include internal and external flows originating from client-to-server as well as server-to-server communication.

2.2 Data Preparation

Due to the extremely large amount of collected flow records, these are partitioned into separate *blocks* of 100 000 records each (representing ∅ 1 min), and the operations given below are applied block-wise. We thus obtain the final dataset, which is used for all experiments in this article, containing about 53 million aggregated *flow entries* out of ≈57 million captured *flow records*.

Enrichment Based on the collected 5-tuples, additional context information is derived. For example, groups of internal and external addresses (e.g., IP subnets, VLANs and geographical regions) can be identified from the network addresses. These contexts deliver additional characteristics and patterns for the subsequent analysis and the prediction process.

Anonymization To ensure that collected metadata cannot be traced back to individual network addresses and end users, while still keeping the syntax and semantic of the data intact to prevent distortion of contained characteristics for the subsequent analysis, an appropriate anonymization algorithm was developed. This mechanism anonymizes all address-related metadata, i.e., IP and network addresses. The data center that exports the traffic metadata defines a password, which is cryptographically hashed and used as a seed for randomized permutation tables. A seed ensures consistent anonymization for further data acquisitions. Each octet of an IP address is anonymized individually using these tables. This way, the semantics of an address, e.g., regarding the relevance and order of the octets forming a group of network addresses, will stay intact after the anonymization and can still be used as a characteristic feature for prediction. However, adjacency of addresses will not be preserved in favor of the anonymization due to the seeded randomization of the permutation tables.

Aggregation Exported unidirectional *flow records* that potentially represent only a part of a communication (due to exporter timeouts or cache sizes) are aggregated to ensure coherent *flow entries*. The aggregation of records is based on the 5-tuple and additional traffic characteristics, e.g., flags and predefined time intervals. Duplicated flow records from both exporting network devices are filtered. During this phase, the number of records is reduced to, on average, 7.5 % of the collected flow records. Afterwards, ports greater than 32 767 are replaced by zero because they are chosen randomly by common operating systems.

Normalization We convert raw, heterogeneous features into a format suited for DNNs, e.g., a sequence of floating points, in three different ways: Bit patterns are converted by promoting each bit to a 0.0 or 1.0, float values are interpolated between 0.0 and 1.0 (min-max normalization) and categorical values are encoded as "one-hot" vectors, i.e., a single value of 1.0 put at an unique position, having a length of N, where N represents the number of distinct categories. An example is given in Tab. 1.

Table 1: Exemplary normalization of an IP address, a port and a protocol value using different data formats (bit pattern or float value). Each data type has a feature-dependent size specifying the number of individual float values that are used as input for the DNN. For example, next to its raw format, each octet of an IP address is represented in its original format as bit pattern or as float values.

Feature	**Raw format**	**Bit pattern** (size)	**Float value(s)** (size)
IP address	81.169.238.182	0,1,0,1,0,0,0,1,1,0,1,0,1,0,0,1, 1,1,1,0,1,1,1,0,1,0,1,1,0,1,1,0 (32)	0.3176, 0.6627, 0.9333, 0.7137 (4)
Port	80	0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0 (16)	0.0012 (1)
Protocol	6	0,0,0,0,0,1,1,0 (8)	0.0235 (1)

The output of the normalization and thus of the data preparation process is the actual dataset (2.3 GB). Next to the bit rate, there are other flow features that can be used as class labels and hence for a prediction, e.g., the number of transferred bytes or the duration of a flow. A combination of selected labels is conceivable as well. The datasets structure is summarized in Tab. 2.

2.3 Data Processing

In the data processing phase a fully-connected DNN is trained to predict the bit rate of a communication. During the processing of the created flow dataset, three steps are performed blockwise: At first, a sub-dataset can be extracted by

Table 2: Overview of the dataset features and labels. For each raw flow feature, the supported respectively used (gray highlighting) data formats are shown, and the number of values is given, as well as the point in the flow data pipeline in which the information is added (Src). Features for both source and destination are marked with ⇄.

Feature	**Data format**			**Src**
	Float	Bit	OH	
`month`	1	4	12	DC
`day`	1	5	31	
`hour`	1	5	24	
`minute`	1	6	60	
`second`	1	6	60	
`protocol`	1	8	✗	
`address` ⇄	4	32	✗	
`port` ⇄	1	16	✗	
`network` ⇄	4	32	✗	DE
`prefix_len` ⇄	1	5	✗	
`asn` ⇄	1	16	✗	

Feature	**Data format**			**Src**
	Float	Bit	OH	
`longitude` ⇄	1	✗	✗	DE
`latitude` ⇄	1	✗	✗	
`country_code` ⇄	1	8	240	
`vlan` ⇄	1	12	✗	
`locality` ⇄	✗	1	2	
`flags`	1	8	✗	
Label	**Data format**			**Src**
`duration`	✗	✗	✓	DA
`bytes`	✗	✗	✓	
`bit_rate`	✗	✗	✓	

DC = Data Collection; DE = Data Enrichment; DA = Data Aggregation; OH = One-Hot

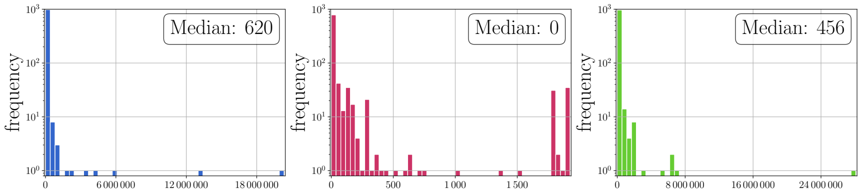
feature selection. Afterwards, data samples are labeled based on predefined class boundaries, which are selected to fit an almost balanced data distribution (presented in Sec. 3.1). Finally, training and testing is done on each individual block sequentially. To evaluate different hyper-parameter setups, we do a parameter optimization. The detailed process and related results are presented in Sec. 4.

3 Exploratory Data Analysis and Visualization

To provide a better understanding of flow data, we explore the distribution of features used for labeling (see Sec. 3.1) and visualize the intrinsic structure of the data (see Sec. 3.2). The analysis is performed on the first 1 000 flow entries (including all features) that are selected from the shuffled test data of the first block. Due to this, the same t-SNE output can be used for all context-related taggings. No significant deviations were observed when performing this analysis on other blocks (every 50th block was compared). All comparisons of t-SNE outputs are done by visual inspection.

3.1 Label Distribution

We analyze the distribution of flow features that can be target values for traffic flow prediction, i.e., the transmitted bytes, the duration or the bit rate calculated from both. Results are shown in Fig. 3. As other authors noted previously [3], these features deviate strongly from a uniform distribution, which makes the determination of suitable class boundaries challenging. The principal conclusion we draw from this is that we must use class balancing (see Sec. 4). Although the data distribution justifies our class boundaries, their practical applicability, e.g., for intelligent routing, is questionable and considered as future work.



(a) number of bytes (b) duration in *seconds* (c) bit rate in $^{bit}/_{sec}$

Fig. 3: Histograms for the possible flow labels of the selected 1 000 elements in the first block. The majority of data samples have both a very small number of transferred bytes (a) and a short duration (b). Median values of 620 bytes respectively 0 seconds (<1 000 ms) substantiate this fact. Hence, the bit rate values (c) are also very unevenly spread over the entire value range (median value is 456).

3.2 Structural Context

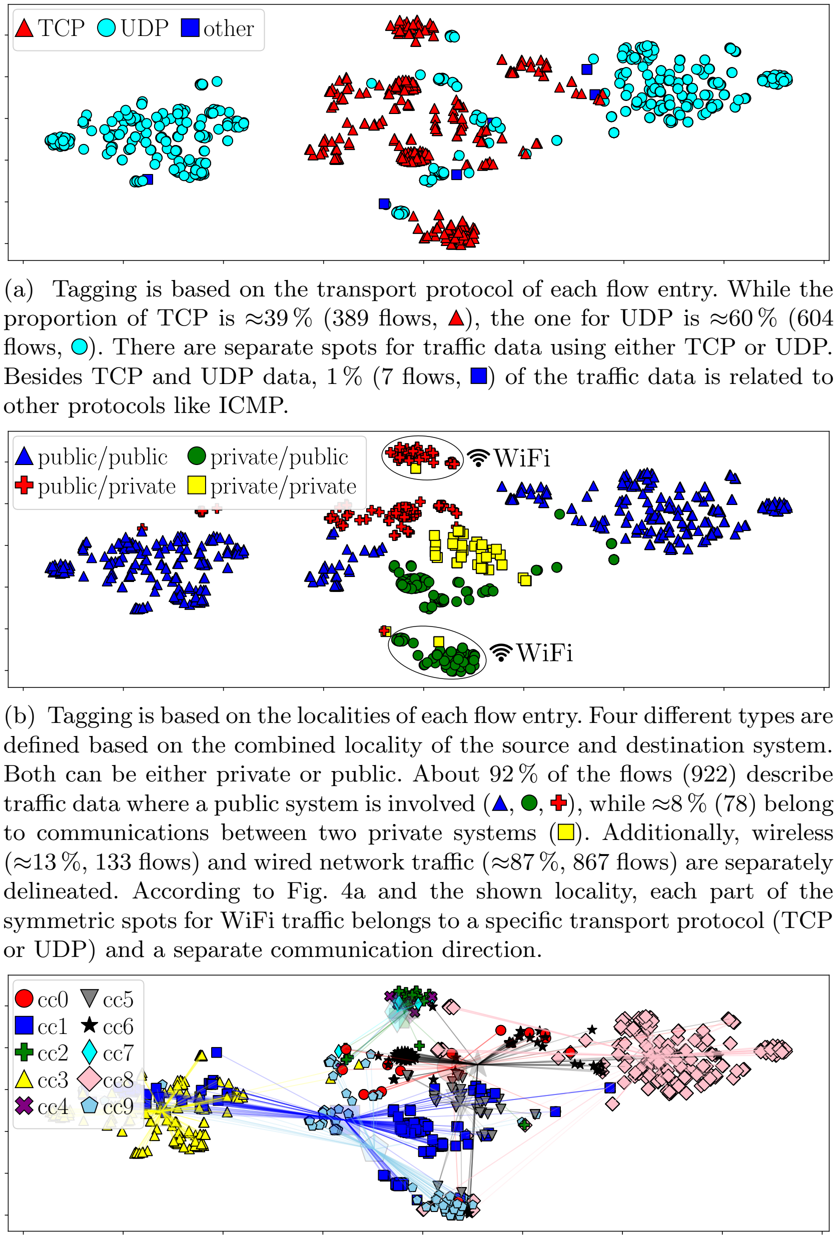
To discover structural relations and similarities between individual flow entries (see Figs. 4 and 5), we use t-Distributed Stochastic Neighbor Embedding (t-SNE) [2], a state-of-the-art method, which maps high-dimensional data samples to a low-dimensional space (2D or 3D) for visualization. We use the t-SNE implementation of the scikit-learn framework, parameter values being an iteration counter of 500, a perplexity of 50 and a learning rate of 200.

Fig. 4a illustrates feature similarities between flow entries that have a common transport protocol. Two symmetric accumulations indicate opposite directions of the same communication. Furthermore, there are examples that do not share the same transport protocol, but t-SNE points out similar feature data.

Tagging of each data sample according to its type of communication, which is the combination of the source and destination locality (either private or public), is shown in Fig. 4b. For each communication type symmetric accumulations can be identified, whereas coherent spots map to individual flow directions.

Additionally, we apply the k-means clustering algorithm on the sub-dataset and use the result for tagging the data samples in the t-SNE output. With k-means, high-dimensional data samples are grouped around a predefined number of iteratively relocated cluster centers. We use the implementation of tensorflow (v1.12) with 10 cluster centers, whereby the initial location of each center is determined randomly and the squared Euclidean distance is used as metric. The tagging of the t-SNE output based on k-means clustering for the data samples is shown in Fig. 4c. According to the t-SNE results, it can be observed that there are samples that belong to the same cluster but have certain feature differences and that there are samples of different clusters sharing feature properties. The actual results depend on the chosen number of cluster centers.

We also perform an outlier detection for each k-means cluster using different metric thresholds (average and median distance as well as both summed up with the standard deviation). See Fig. 5a for an exemplary presentation of detected outliers. With regard to our experiments described in the next section, the outlier detection has no significant influence on network flow prediction.

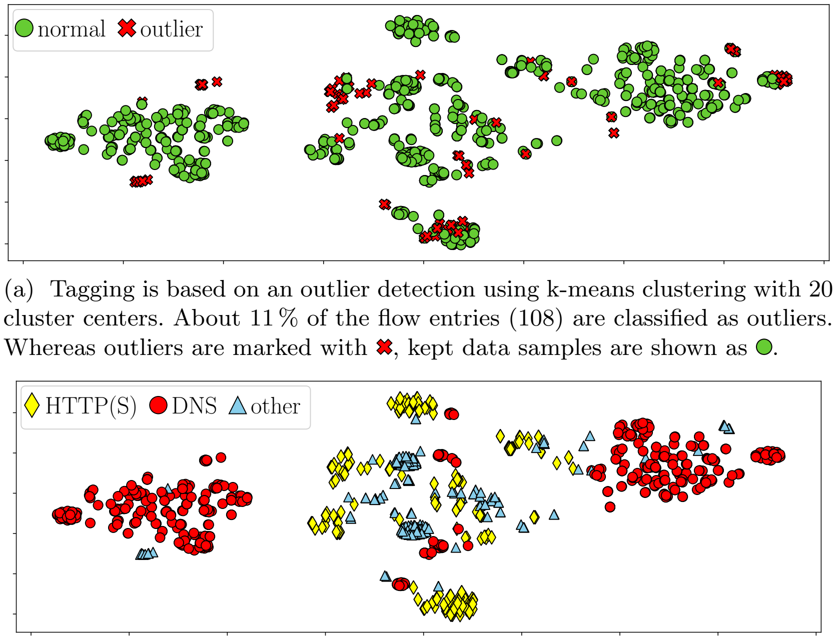


(a) Tagging is based on the transport protocol of each flow entry. While the proportion of TCP is ≈39 % (389 flows, ▲), the one for UDP is ≈60 % (604 flows, ●). There are separate spots for traffic data using either TCP or UDP. Besides TCP and UDP data, 1 % (7 flows, ■) of the traffic data is related to other protocols like ICMP.

(b) Tagging is based on the localities of each flow entry. Four different types are defined based on the combined locality of the source and destination system. Both can be either private or public. About 92 % of the flows (922) describe traffic data where a public system is involved (▲, ●, ✚), while ≈8 % (78) belong to communications between two private systems (■). Additionally, wireless (≈13 %, 133 flows) and wired network traffic (≈87 %, 867 flows) are separately delineated. According to Fig. 4a and the shown locality, each part of the symmetric spots for WiFi traffic belongs to a specific transport protocol (TCP or UDP) and a separate communication direction.

(c) Tagging is based on the output of k-means clustering. Positions of cluster centers (10) are visualized according to t-SNE output. Nearly uniform accumulations of samples can be identified (e.g., ▲, ◆), clusters are spread (e.g., ●, ⬟) and mixtures of samples of different clusters (e.g., ✚, ✖) are recognizable.

Fig. 4: Visualization of the selected 1 000 flow samples. All figures show the same t-SNE results. Tagging is based on the transport protocol (a), locality (b) and clustering (c).



(a) Tagging is based on an outlier detection using k-means clustering with 20 cluster centers. About 11 % of the flow entries (108) are classified as outliers. Whereas outliers are marked with ✖, kept data samples are shown as ●.

(b) Tagging is based on most frequent application protocols. Most samples belong to DNS communications (≈57 %, 571 flows, ●). Next to HTTP(S) (≈24 %, 241 flows, ◊), other application protocols are visualized (≈19 %, 188 flows, ▲). The latter include, for example, authentication, network monitoring and mail.

Fig. 5: Visualization of structures in the selected 1 000 samples. Tagging of the t-SNE results is based on an outlier detection (a) and the applications protocols (b).

According to Fig. 5b DNS and HTTP(S) are the most used application protocols in the dataset. The huge proportion of DNS traffic states the rate of flow entries with a low bit rate respectively short duration.

The data analysis emphasizes relations and feature similarities between individual data samples. All visualizations use the same t-SNE output, but context-related tagging, e.g., regarding used protocols or communication directions, helps to clarify different structural patterns within network flow data.

4 Network Flow Prediction Experiments with DNNs

We employ a fully-connected DNN with L layers of identical sizes S, each hidden layer applying a ReLU transfer function whereas the output layer applies a softmax function. The batch size $bs = 100$ and the number of training epochs $\mathcal{E} = 10$ are fixed for all experiments. DNN training minimizes a standard cross-entropy loss by stochastic gradient descent by means of the Adam Optimizer. The last 10 % of the chronologically ordered data are completely used for testing every 50th iteration.

Choice of evaluation metrics Since we are dealing with a three-class problem, the usual metrics for binary problems are not applicable, such as F1 score, precision, recall, etc. Instead, we present results in the more general form of a confusion matrix, from which we can derive classification accuracy by considering only the diagonal elements. Both of these measures are applicable for classification tasks with an arbitrary number of classes, which can be useful for comparison should we decide to introduce more classes at a later point. In order to allow a more in-depth comparison between the experimental conditions (using the 5-tuple information –vs– using all features), we decided to additionally compute the standard binary performance metrics separately for each of the three classes.

Hyper-parameters Tunable parameters include the learning rate ϵ and the optional application of dropout to input d_i and hidden layers d_h, with different dropout probabilities. The assignment of labels is done based on a class boundary parameter $\mathcal{C}$. This list of boundary values is consistently used for all blocks before a training phase. In order to specify the class balancing method, the parameter $\mathcal{W}$ is introduced. Balancing for training and test data is achieved either by standard class weighting or under-sampling. Furthermore, a feature selector $\mathbb{F}$ provides support for the construction of sub-datasets. O_c specifies the number of cluster centers that are used for outlier detection using k-means clustering. All hyper-parameters mentioned here (L, S, ϵ, d_i, d_h, $\mathcal{C}$, $\mathcal{W}$, $\mathbb{F}$, O_c) are varied to perform a joint parameter optimization.

We train all DNN classifiers on the first 10 blocks sequentially and evaluate the achieved prediction accuracy on each block's test set. In order to obtain the best possible results, we conduct a combinatorial hyper-parameter optimization, leading to a total of 5 400 DNN training and evaluation runs. The explored parameter ranges are summarized in Tab. 3. Depending on the hardware, the computation time of one experiment is between 8 and 15 minutes. Based on the complexity of the DNN and the chosen parameters, the GPU memory usage is between 140 and 264 MB and the RAM utilization varies from 762 to 1 200 MB.

Labeling Because of the unbalanced data distribution (see Sec. 3.1) that makes regression problematic (also addressed in [5]), we treat network flow prediction as a classification problem, using the three exemplary classes "low", "medium"

Table 3: Overview of the variables and tested values for parameter optimization.

Parameter	**Variable**	**Values**
Dropout (input, hidden)	(d_i, d_h)	$\{(1.0, 1.0), (0.9, 0.6), (0.8, 0.5)\}$
Layers	L	$\{3, 4, 5\}$
Neurons per layer	S	$\{200, 400, 600, 800, 1\,000\}$
Learning Rate	ϵ	$\{0.01, 0.001, 0.0001\}$
Features	$\mathbb{F}$	{5-tuple, all}
Class boundaries (bit rate)	$\mathcal{C}$	$\{\{0, 500, 5\,000, \infty\},$ $\{0, 50, 8\,000, \infty\}\}$
Class balancing method	$\mathcal{W}$	{0 (under-sampling), 1 (class weighting)}
Cluster centers (outlier detection)	O_c	$\{0, 20, 60, 100, 500\}$

and "high". The calculated bit rate of each flow is used for computing a class based on thresholding operation (with the two thresholds adapted such that the distribution of classes is approximately flat). Next to the used set of boundaries for class division, Tab. 4 presents related characteristics for each class.

Table 4: Exemplary class partitioning for the prediction of a flow's bit rate. Next to the related intervals, the median and mean value, the average number of elements using class balancing (class weighting or under-sampling) and the data distribution within each class for the first 10 blocks of the dataset are shown (log scale).

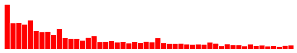
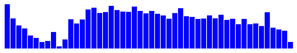
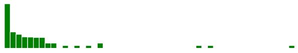
Class	**Interval**	**Median/ Mean**	**Average elements**		**Data distribution**
			class weighting	under-sampling	
0	$[0, 50)$	0/ 2	21 945	21 909	
1	$[50, 8\,000)$	3 904/ 4 004	28 228		
2	$[8\,000, \infty]$	16 960/ 131 736	24 459		

Fig. 6 depicts the distribution of the true labels within the t-SNE output. Whereas some spots primarily have data samples belonging to the same class (c0), other spots are a mixture of different (c1, c2) or all classes. With regard to Fig. 4c, the results of k-means clustering cannot be used to classify the samples adequately. Respectively, it is not sufficient to predict the bit rate of a flow.

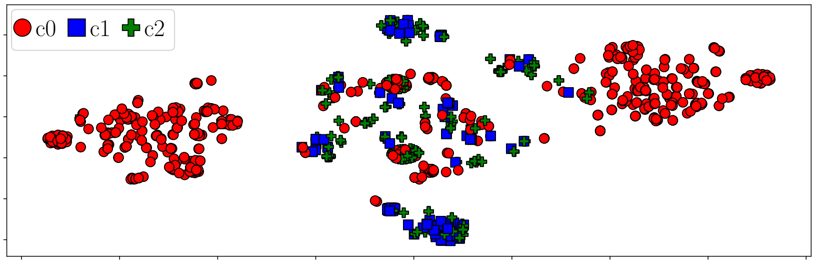


Fig. 6: Visualization of the t-SNE results based on the true labels for the selected 1 000 samples. Tagging is based on the three class labels ((c0 ≈65 %, 652 flows, ●), (c1 ≈12 %, 120 flows, ■), (c2 ≈23 %, 228 flows, ✚)), whereby the exemplary boundaries are used.

The two experiments with the highest accuracy, determined by the parameter optimization, are shown in Fig. 7 and Tab. 5. In the first experiment, training is done on all available flow features (247 inputs), whereas in the second one only the 5-tuple (104 inputs) is used. Fig. 7 depicts the trend of the prediction accuracy. At the beginning of a directly following block, the accuracy value can considerably vary compared to the rate for the previous block but generally stabilizes for each block after a few training iterations. This indicates a slight change in statistics (concept drift) between the individual blocks, which becomes clearer in Fig. 9. We achieve a maximum accuracy of ≈87 % for the first respectively ≈85 % for the second experiment. Regarding these maxima, the data enrichment leads to an accuracy increase of about 2 %. Confusion matrices for both experiments are also outlined in Fig. 7. The distinction between class 1 and 2 is more challenging. Fig. 8 gives an overview of the false classified data samples. With regard to the false labels, prediction errors for coherent spots mainly belong to the same class.

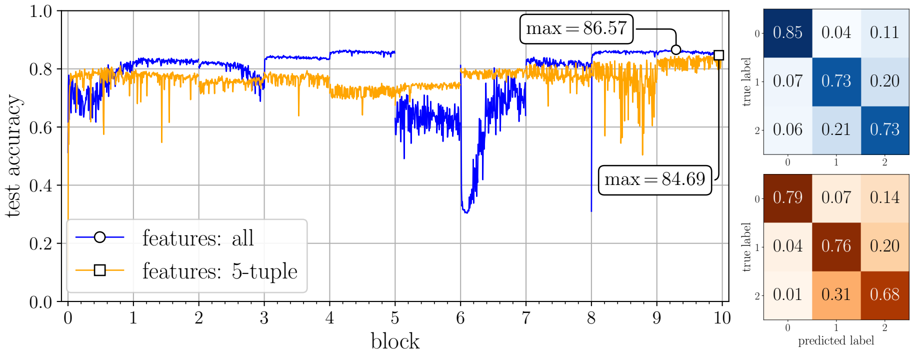


Fig. 7: Testing results for the best experiments determined by the parameter optimization: first ($\mathbb{F} = all$, blue, ○) and second experiment ($\mathbb{F} = 5\text{-}tuple$, orange, □). The trend of the accuracy for the first 10 blocks is depicted. Training and testing is done sequentially on each independent block for 10 epochs. Besides this, the confusion matrices for the last iteration of block 0 in the first (top, blue) and second experiment (bottom, orange) are shown. Hyper-parameters $\mathbb{F} = all$: $\mathcal{C} = [0, 50, 8\,000]$, $(d_i, d_h) = (1.0, 1.0)$, $L = 3$, $S = 1000$, $\epsilon = 0.0001$, $\mathcal{W} = 1$, $C_k = 0$; parameters $\mathbb{F} = 5\text{-}tuple$: $\mathcal{C} = [0, 50, 8\,000]$, $(d_i, d_h) = (0.9, 0.6)$, $L = 5$, $S = 1000$, $\epsilon = 0.001$, $\mathcal{W} = 0$, $C_k = 0$.

Table 5: Common binary classification measures, given separately for each of the three classes in a one-against-all setting. These measures are instructive, particularly when comparing performance between the two experiments (5-tuple only against the full set of features). The values can be computed from the confusion matrices shown in Fig. 7.

exp.	precision	recall/sensitivity	specificity	accuracy
5-tuple class 0	98	84.5	97.3	89.5
5-tuple class 1	43.1	69.8	88.9	86.9
5-tuple class 2	65.8	69.7	85.6	81.0
all feat. class 0	95.4	88	93.5	90.2
all feat. class 1	53.2	65.5	93.1	90.1
all feat. class 2	70.9	76.5	87.5	84.4

5 Discussion and Principal Conclusions

The principal conclusions we can draw from the presented experiments are: First of all, DNNs are a feasible tool for performing fine-grained network traffic flow prediction in a "big data" setting, achieving an accuracy of roughly 87 % even though performed in a streaming fashion on successive and independent blocks of flow data. Previous studies reached accuracies over 90 % but grouped network

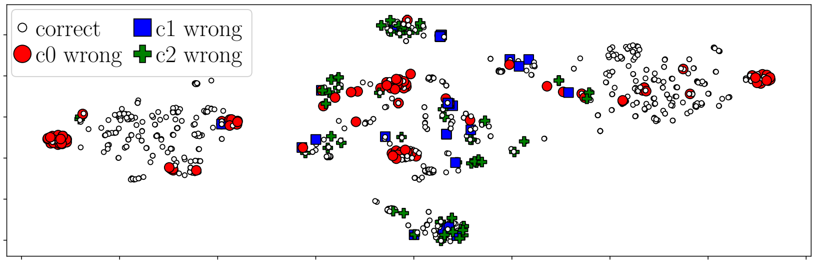


Fig. 8: Visualization of the t-SNE results based on the predicted labels for the selected 1 000 flow samples. Correct classified samples are marked with ○ (correct≈78 %, 778 flows). For all false classified samples ((c0 wrong≈12 %, 120 flows, 🔴), (c1 wrong≈3 %, 33 flows, 🟦) and (c2 wrong≈7 %, 69 flows, ✚)) the true label is shown.

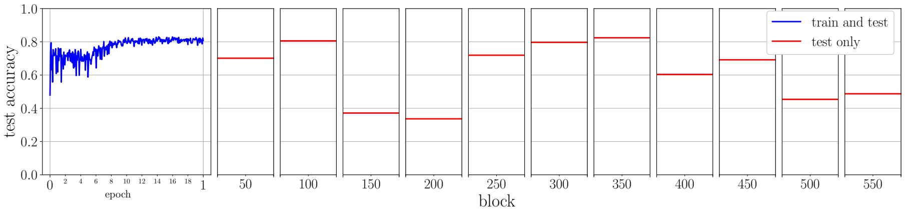


Fig. 9: Overview of the accuracy for different blocks of the dataset. A DNN is trained and tested for 20 epochs on the first block using all available flow features (blue line). Subsequently, only the accuracy is determined for each fiftieth block while measuring on test data for one epoch (red lines). Parameter setup: see first experiment in Fig. 7.

flows in only two classes ("mice" and "elephant" flows), which is considerably less useful for fine-grained network traffic engineering, and, above all, processed all training data in a single block. Secondly, we find that data enrichment can be useful, as it improves classification accuracy by roughly 2 % at manageable computational cost. Thirdly, our visualization and clustering studies show that there is no simple way to improve results by outlier detection, presumably because the data samples do not lend themselves to clustering using Euclidean distance, and a custom distance metric would have to be used here. We establish nevertheless that t-SNE is a useful tool to visualize structures and relations in network flow data. Lastly, we confirm by experiments that there is moderate to strong concept drift in flow data, and that appropriate measures will have to be taken in future works to address this issue.

Comparability and Validity of Results We may ask how generalizable our results are, and the answer is of course complex. In a university campus scenario such as ours, there are numerous factors that may affect the results, like the day of week, the season, the proximity of tests, etc. For example, the WiFi network – including thousands of connected students – represents a dynamic setup that probably cannot be solved easily for a DNN because connections are unique and non-recurring (in contrast to, e.g., communications between servers). Identifying and excluding such "difficult" flows could conceivably improve prediction accuracy and generalizability of our results. As stated in Sec. 1.2, publicly available datasets are relatively small. Larger datasets are not accessible, probably due to privacy issues. Even though our campus network is unique in its structure and thus results on our data do not in any way guarantee that the approach will work in other networks, the same can be said for any of the previous studies on the subject. The only way to show generality would be to have access to several datasets of network flows of comparable size, and to perform the same experiments on all of them. Comparing our results to other studies on the subject is further complicated by the fact that we perform three-class classification whereas previous studies were concerned with two-class scenarios only.

Discussion of the three-class scenario To show that our architecture can replicate previous results, we trained our DNNs on a two-class task with a threshold value of 500 bits per second between "mice" and "elephant" flows and obtain a test accuracy of over 90 %, which is comparable to the results of other studies while taking the abovementioned caveats into consideration. Obviously, introducing an additional class degrades the classification accuracy, simply because guessing has a lower chance of success with one more class to choose from. Whether this lower prediction accuracy is compensated by the benefit of a more fine-grained prediction would have to be tested in simulation, which is what we are currently working on. For this study, we wished to establish that more than two classes can be successfully integrated into a prediction scheme, all the more since the computational cost of predicting more classes is negligible at inference time. When also considering that we perform learning in a streaming setting, which in general degrades performance w.r.t. settings where all data are simultaneously available for training, our results must be considered very competitive.

Justification of using DNNs The principal reason for using DNNs as opposed to other methods proposed in the literature, e.g., Gaussian Process Regression (GPR) [5], is the fact that in future we want to train our classifiers in a streaming fashion: As soon as a new data block has been collected, model re-training is conducted automatically, and the trained model is immediately deployed and used for flow classification. This puts a strong focus on the scalability of the training process w.r.t. the number of data samples. In [5], a training complexity of $\mathcal{O}(n \cdot m^2)$ is reported for GPR, where concrete values for m, or how they are chosen, are unclear. Naively, GPR has a training complexity of $\mathcal{O}(n^3)$, and it is unclear whether the optimizations discussed in [5] can be tuned without human intervention (no code is provided). In contrast, DNNs have a natural training complexity of $\mathcal{O}(n)$ without any optimizations, so they do seem a more natural choice in the "big data" context. We will investigate the performance of other learning algorithms in future work, and compare them to our approach.

Acknowledgements We thank Sven Reißmann from the university data center for assistance with data collection and preparation. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU.

References

1. Benson, T., Akella, A., Maltz, D.A.: Network traffic characteristics of data centers in the wild. In: Proceedings of the 10th ACM SIGCOMM conference on Internet measurement. pp. 267–280. ACM (2010)
2. van der Maaten, L., Hinton, G.: Visualizing Data using t-SNE. Journal of Machine Learning Research (2008). https://doi.org/10.1007/s10479-011-0841-3
3. Nguyen, T.T., Armitage, G.J.: A survey of techniques for internet traffic classification using machine learning. IEEE Communications Surveys and Tutorials **10**(1-4), 56–76 (2008)
4. Pfülb, B., Gepperth, A.: A comprehensive, application-oriented study of catastrophic forgetting in dnns. In: International Conference on Learning Representations (ICLR) (2019), accepted
5. Poupart, P., Chen, Z., Jaini, P., Fung, F., Susanto, H., Geng, Y., Chen, L., Chen, K., Jin, H.: Online flow size prediction for improved network routing. In: 2016 IEEE 24th International Conference on Network Protocols (ICNP). pp. 1–6. IEEE (2016)
6. Shi, H., Li, H., Zhang, D., Cheng, C., Wu, W.: Efficient and robust feature extraction and selection for traffic classification. Computer Networks **119**, 1 – 16 (2017). https://doi.org/https://doi.org/10.1016/j.comnet.2017.03.011, `http://www.sciencedirect.com/science/article/pii/S1389128617300786`
7. Valadarsky, A., Schapira, M., Shahaf, D., Tamar, A.: Learning to route. In: Proceedings of the 16th ACM Workshop on Hot Topics in Networks. pp. 185–191. HotNets-XVI, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3152434.3152441, `http://doi.acm.org/10.1145/3152434.3152441`
8. Wang, P., Lin, S.C., Luo, M.: A framework for QoS-aware traffic classification using semi-supervised machine learning in SDNs. In: 2016 IEEE International Conference on Services Computing (SCC). pp. 760–765. IEEE (2016)
9. Xiao, P., Qu, W., Qi, H., Xu, Y., Li, Z.: An efficient elephant flow detection with cost-sensitive in SDN. In: 2015 1st International Conference on Industrial Networks and Intelligent Systems (INISCom). pp. 24–28. IEEE (2015)