



**University of
Zurich^{UZH}**

An Explorative Study of Encrypted Traffic Analysis based on Machine Learning

*Cyrill Halter
Zurich, Switzerland
Student ID: 13-938-171*

Supervisor: Bruno Rodrigues
Date of Submission: July 18, 2019

Abstract

Network traffic classification is a fundamental part of many network management and network security related tasks, such as intrusion detection or prevention. The emergence and growth of encrypted network protocols, however, have rendered traditional techniques, such as deep packet inspection, inadequate to perform this essential task reliably. Machine learning techniques offer new opportunities to extract latent information and relationships from network traffic without the need to access its content. This independent study examines the current state of research in this area in an explorative fashion. The ML algorithms used in the field are studied and their pros and cons are assessed. A survey of recent publications is conducted and 12 approaches are compared based on several categories. The machine learning algorithms used and the implementations thereof are described in detail and their advantages and disadvantages are explained. To gain further insight into the problem domain, an experimental classifier is built with the Tensorflow framework and tested using a network flow dataset. The performance is shown to have room for improvement overall but to be comparable to a baseline model, indicating that the problem may lie with the data preparation and cleansing steps taken and not with the classifier.

Contents

1	Introduction	1
2	Background	3
2.1	Supervised vs. Unsupervised ML	3
2.2	Naïve Bayes	4
2.3	Decision Trees	4
2.4	Random Forests	6
2.5	Support Vector Machines	6
2.6	Neural Networks	7
2.7	Convolutional Neural Networks	9
2.8	Recurrent Neural Networks	9
2.9	Autoencoders	10
2.10	Generative Adversarial Network	11
3	Survey of Related Work	13
3.1	Lotfollahi et al., 2017	16
3.2	Wang et al., 2017	16
3.3	Li and Moore, 2007	17
3.4	Lopez-Martin et al., 2017	17
3.5	Vu, Bui, and Nguyen, 2017	18
3.6	Zhang et al., 2015	19
3.7	Taylor et al., 2017	19

3.8	Chen et al., 2017	20
3.9	Höchst et al., 2017	21
3.10	Bocchi et al., 2016	21
3.11	Rezaei and Liu, 2018	22
3.12	Bekerman et al., 2015	22
4	Experimental Classification with TensorFlow	25
4.1	Tech Stack	25
4.2	Dataset	26
4.3	Data Preparation	27
4.4	Classification	28
4.5	Results	30
5	Conclusion & Lessons Learned	33
	List of Figures	34
	List of Tables	35

Chapter 1

Introduction

Network traffic classification involves the task of identifying the applications, protocols and devices used to generate network traffic. Recently, it has become more critical with the rapid growth of current Internet network and online applications. Coupled with advances in software and theory, the range of classification techniques has also increased. Network traffic classification lies at the core of many network management and communication security related tasks. It is considered the first line of defense where a malicious activity can be filtered, identified and detected. Also, it is a central component in evidence collection and analysis.

A traditional approach to investigate network attacks is the use of Deep Packet Inspection (DPI). This technique consists of a thorough examination of the fields contained in the packets that flow within the investigated network. The implementation of strong encryption mechanisms in network communication, however, has complicated this task since the contents of network packets are no longer accessible. In this regard, multiple Machine Learning (ML) tools have been developed with varying degrees of success, the goal being the exploitation of information latently present in the encrypted network communication to classify traffic.

This independent study aims to identify, map and explore techniques and tools suitable to the classification of encrypted network traffic based on behavioral analysis using ML techniques. The goal is to gain an insight into the current state of research in the field of network traffic classification techniques using ML. This comprises a study of the ML algorithms used in the field, as well as a comparison of the concrete implementations of those algorithms in published work, including their advantages and disadvantages. A second step aims to gain hands on experience with the problem domain through the implementation of a specific ML approach.

The rest of this document will be structured as follows: Chapter 2 will explain some of the basic current techniques and building blocks of ML. Chapter 3 will elaborate on the current research into encrypted network traffic classification using ML techniques. Chapter 4 will explain an experimental setup used to explore a specific ML approach in order to gain further insight into the problem domain. Finally, Chapter 5 will conclude the work and explain the lessons learned and insights gained.

Chapter 2

Background

The following chapter will provide an overview over ML approaches used for network traffic classification. While the range of ML techniques is very large, this chapter focuses purely on methods used in the approaches described in Chapter 3. These approaches were selected based on currentness, novelty and quality of the paper. The fundamental principles of each method will be summarized, omitting further technical details and implementation considerations. In addition, the advantages and disadvantages of each approach will be listed in a table.

2.1 Supervised vs. Unsupervised ML

ML methods can be divided broadly into the two categories, supervised and unsupervised [24]. They differ fundamentally in the task that they aim to solve. The main goal of unsupervised methods is to learn the inherent structure of the data at hand. This is done without the need for prior labeling of individual data points. Instead, these methods determine the discriminative components of the data or combinations thereof. Typical use cases for unsupervised methods are exploratory data analysis, dimensionality reduction or clustering.

Supervised methods aim to find structures or relationships in the data that allow to predict some output from an input data point. Mathematically speaking, this corresponds to approximating some objective function that the data implicitly describes. Other than unsupervised approaches, supervised methods require a ground truth dataset, called training data or gold data. It is a set of data that has been annotated a priori with correct output values. The main use cases for supervised ML are classification, the assignment of some categorical label to a datapoint, and regression, the approximation of a function with continuous output.

A third approach, semi-supervised ML, lies at the intersection of the two methods described above. The goal here is the same as for the supervised approach, however, these methods aim to make use of unlabeled data in addition to labeled data. This may be done, for example, to reduce the amount of gold data needed to train a ML algorithm.

2.2 Naïve Bayes

Naïve Bayes (NB) is a classifier model based in probabilistic theory [5]. The name originates from the fact that the naïve assumption of feature independence is made, meaning that only the outcome variable is assumed to be dependent on the individual feature variables and no dependencies exist between the feature variables. This allows us to use Bayes' theorem.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

This can then be applied to the ML problem with feature vector $X = (x_1, x_2, \dots, x_n)$ and outcome variable y .

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

Since the denominator $P(X)$ never changes, we can use proportionality.

$$P(y|X) \propto P(X|y)P(y) = P(y)\prod_{i=1}^n P(x_i|y)$$

The classification task then consists of finding

$$y = \arg \max_y P(y)\prod_{i=1}^n P(x_i|y)$$

The model parameters $P(y)$ and $P(x_i|y)$ can be calculated directly from the training dataset. The advantages and disadvantages of the Naïve Bayes approach are listed in Table 2.1 [26].

Pros	Cons
• Easy to implement • Descriptive • Well-suited for categorical variables	• Requirement for feature independence seldom fulfilled in real-life cases

Table 2.1: Pros and Cons of the Naïve Bayes Algorithm [26]

2.3 Decision Trees

Decision Trees (DT) are a relatively simple learning method that work by segmenting the prediction space recursively [9]. They can be used for regression and classification problems.

A DT is a tree structure where each internal node is labeled with an input feature. Each arc originating from said node is labeled with a value or a range of values that the input feature might have. The leaf nodes correspond to the target variable that should be predicted. A DT can then be trained by splitting the training dataset recursively along the tree until the output is univocal or remains unchanged after further splitting. Figure 2.1 shows a DT for predicting the survival of Titanic passengers, for example. The pros and cons of the DT approach are outlined in Table 2.2 [9] [26].

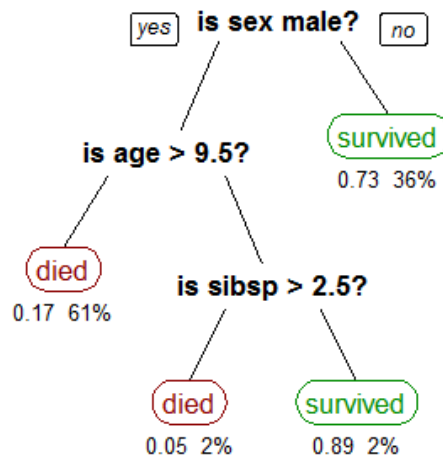


Figure 2.1: A tree showing survival of passengers on the Titanic [29]. *sibsp* is the number of spouses or siblings aboard

Pros	Cons
• Faster than RFs • Descriptive (can be displayed graphically) • Easy to understand • Easy handling of categorical predictors without the need for dummy variables • No assumption on the distribution of the data	• Lower level of predictive accuracy than other approaches • Non-robust to changes in training data • Prone to overfitting

Table 2.2: Pros and Cons of the Decision Tree Algorithm [9] [26]

2.4 Random Forests

Random Forests (RFs) are an extension of the DT approach that combine the idea of bagging with random feature selection to add robustness to DTs and combat the tendency to overfit to the training data [9]. In the bagging approach, multiple DTs are built from different randomly selected subsets of the training data. RFs complement this by also randomly selecting features at each split in a DT. This avoids correlation of the individual DTs that might otherwise arise from the bias towards those features with stronger predictive power. The multiple DTs generated by this method are then used in combination to make a prediction through majority voting. The advantages and disadvantages of RFs are outlined in Table 2.3 [9] [15].

Pros	Cons
• Increased robustness compared to DTs • Decreased proneness to overfitting compared to DTs	• More difficult to understand and interpret than DTs • Computationally more expensive than DTs

Table 2.3: Pros and Cons of the RF Algorithm [9] [15]

2.5 Support Vector Machines

A Support Vector Machine (SVM) is a generalization of a simple and intuitive technique, the maximal margin classifier [9]. A maximal margin classifier works by determining a multidimensional plane, called a hyperplane, to separate classes in a multidimensional feature space. This hyperplane is calculated in such a way that the margin, the distance between the hyperplane and the points closest to the hyperplane, is maximal. Since only these closest points have an influence on the location of the hyperplane, i.e. the hyperplane is *supported* by them, they are called the support vectors. A SVM contributes in two ways. First, it allows for data points to lie on the wrong side of the separating hyperplane if this improves the classification overall. Second, it allows for non-linear separating hyperplanes by adjusting the distance measure used to determine the margin. In practice, this is calculated using the inner product of the support vectors. The way this inner product is computed, called the kernel, can be adjusted depending on the data that needs to be classified. Linear, polynomial and radial kernels, for example, are possible. Table 2.4 describes the pros and cons of SVM classifiers [9] [26].

Pros	Cons
• Memory efficient • Effective in high dimensional spaces • Effective with small datasets • Effective handling of outliers	• High training time for large datasets due to large support vector

Table 2.4: Pros and Cons of the SVM Algorithm [9] [26]

2.6 Neural Networks

Neural Networks (NNs) are a machine learning algorithm inspired by the human brain [17]. Consequently, the central building block of a NN is a neuron. In summary, a neuron is a component that applies a mathematical function, the so called activation function, to multiple inputs and delivers a single output. Figure 2.2 shows the basic composition of a neuron. The activation function used in most NNs nowadays is the Rectified Linear Unit (ReLU) function (*cf.* Figure 2.3), due to its superior performance.

A NN is then simply a collection of neurons that are arranged in layers, such that the output of a neuron in one layer serves as an input of all neurons in the next layer (*cf.* Figure 2.4). Each of these connections has a variable individual weight that is adjusted during the training phase. The activation function can be chosen individually for each layer. The number of layers in a NN is referred to as the *depth*, the number of neurons in a layer as the *width*. NNs with a large depth are often called *deep* NNs. Table 2.5 outlines the pros and cons of NNs [15].

Pros	Cons
• Flexible Design • Able to approximate arbitrary functions • Able to extract discriminative features in the hidden layers	• Very slow training due to computationally expensive backpropagation algorithm • Non-descriptive and very hard to interpret

Table 2.5: Pros and Cons of NNs [15]

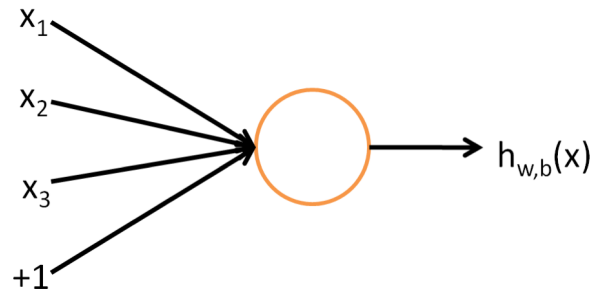


Figure 2.2: The basic composition of a neuron with input values x_1, x_2, x_3 and output value $h_{w,b}(x)$ [17]

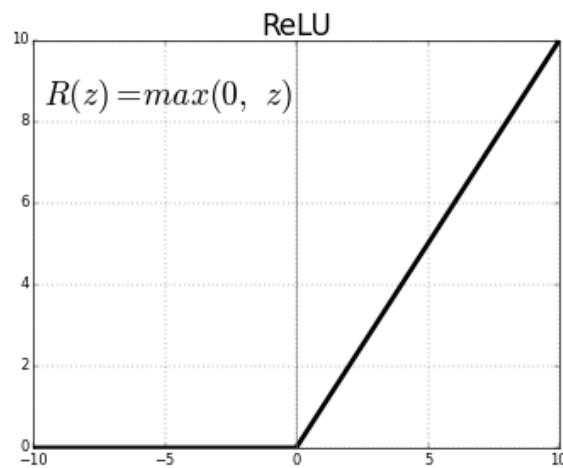


Figure 2.3: The ReLU function, used commonly as an activation function in NNs [17]

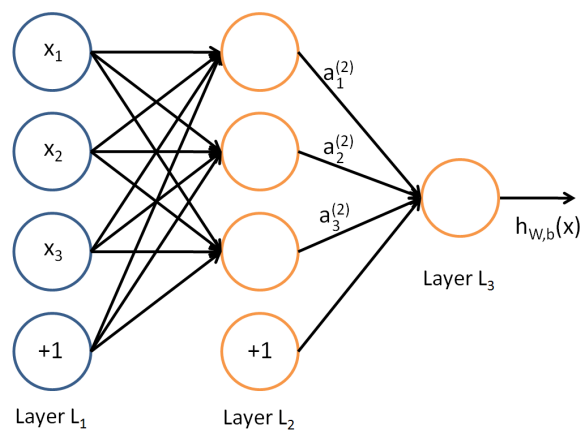


Figure 2.4: A very basic NN with input layer L_1 , one hidden layer L_2 and output layer L_3 [17]. x_1, x_2 and x_3 are the inputs to the NN, $h_{w,b}(x)$ is the output.

2.7 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a subform of NN that have been popularized originally in the realm of computer vision [17]. The main difference to regular NNs is that in CNNs, the input dimensions are smaller than the dimensions of the data. The CNN's input layer is then incrementally shifted over the data and applied to the corresponding subsets of the data. This is called a convolution. To reduce the size of the output data, CNNs employ pooling layers in addition to these convolutional layers. They essentially apply an aggregation function to a subset of the data to produce a smaller output and are applied in the same way as a convolutional layer. Figure 2.5 shows the structure of a two-dimensional classifier CNN with two convolutional and two pooling layers.

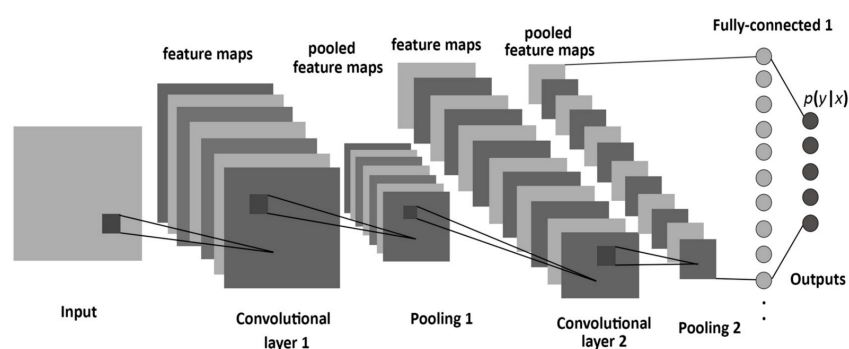


Figure 2.5: A CNN classifier with a two-dimensional input [17]

The reason why CNNs are used frequently in the domain of computer vision or image processing is that the input data may be one-, two- or even higher-dimensional. This makes it particularly suited to the processing of images, but it can also be applied to any other data with some spatial structure. Thanks to the combination of convolutional and pooling layers, the features detected by CNNs also have some shift and scale invariance, making it useful for cases where the location of the discriminative parts in the data isn't clear. The pros and cons of CNNs are outlined in Table 2.6 [17].

Pros	Cons
• Provide some shift and scale invariance • No fixed input size needed	• Only apply to data with spatial structure

Table 2.6: Pros and Cons of CNNs [17]

2.8 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are another subform of NN that were popularized in the domain of language processing. The main difference to regular NNs is that the

same input may not always lead to the same output in a RNN. Instead, the output is dependent on previous inputs apart from the current one. This is achieved including a feedback memory where information can be retained over successive runs of the RNN. Figure 2.6 illustrates this behavior through the representation of an unfolded RNN as a multilayer network. A popular variant of RNN is the Long-Short Term Memory (LSTM) NN. It extends the concept by having multiple feedback memory constructs that are able to store information for varying lengths of time. Table 2.7 outlines the advantages and disadvantages of RNNs [17].

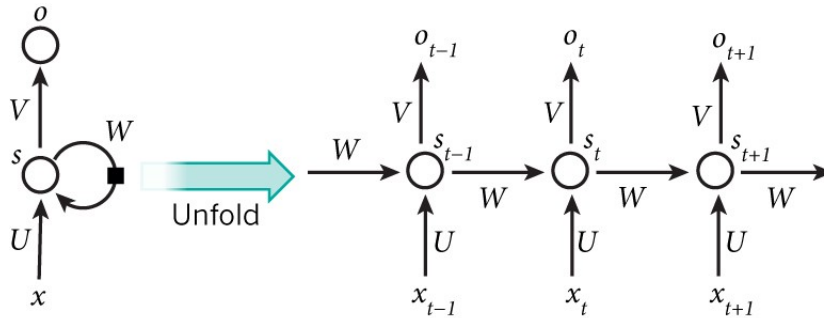


Figure 2.6: An abstract representation of a RNN as a multilayer network, illustrating the method of preserving information across successive runs [17]

Pros	Cons
• Model temporal dependencies on previous observations • Especially LSTM very successful in sequence tagging	• Only apply to data with sequential structure

Table 2.7: Pros and Cons of RNNs [17]

2.9 Autoencoders

An Autoencoder (AE) is an unsupervised subform of NN that is mainly used to extract highly discriminative features from some input dataset [14]. It consists of an encoder component that reduces the size of the input data by gradually reducing the width of the hidden layers, and a decoder component that tries to reconstruct the input from the output of the encoder (*cf.* Figure 2.7). The encoder component thus creates a compressed vector representation of the data. This is, in essence, a dimensionality reduction task similar to what is done with Principle Component Analysis (PCA). Other than PCA, however, AEs are able to use non-linearity in the encoding.

AEs can be used for a variety of tasks apart from dimensionality reduction. A denoising AE, for example, uses measures for intentional corruption of the input data, such as

dropout layers, in order to learn how to reconstruct a clean output from an imperfect input [16]. Variational AEs enforce a continuous encoder output space in order to be able to use the decoder for data generation [23]. The pros and cons of AEs are described in Table 2.8 [3] [11].

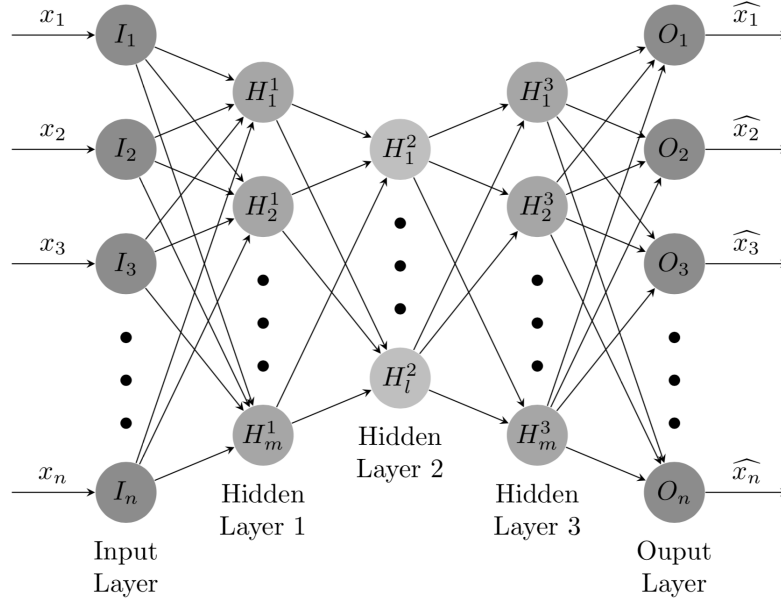


Figure 2.7: Illustration of an AE showing the reduction of dimensionality with encoder layers 1 & 2 and decoder layer 3 [14]

Pros	Cons
• Unsupervised • Can be used for classifier pre-training, data generation and denoising • Generates highly discriminative features as encoder output	• Problematic for unseen types of data

Table 2.8: Pros and Cons of AEs [3] [11]

2.10 Generative Adversarial Network

A Generative Adversarial Network (GAN) is an unsupervised subform of NN [27]. It consists of two main NN components. The generator component uses random noise as input and generates samples. The discriminator component receives as input both the generated samples and the training data and tries to discriminate between the two. These two networks are then trained simultaneously, such that the generator creates increasingly

realistic samples and the discriminator improves its skill of telling whether a sample is real or generated. After training, the generator can be used to supplement data classes that are underrepresented in the training data. Advantages and disadvantages of GANs are listed in Table 2.9 [27].

Pros	Cons
• Unsupervised • Able to generate realistic data	• Purely generative, not useful for classification

Table 2.9: Pros and Cons of GANs [27]

Chapter 3

Survey of Related Work

The following chapter presents a survey of current research in the area of encrypted network traffic analysis based on machine learning techniques. Table 3.1 provides an overview of 12 approaches selected for this study. The remainder of the chapter will detail the ML techniques used in the specific approaches, including the features used, the classification algorithm, as well as contributions and limitations of the selected procedures. The approaches were selected based on currentness, novelty and quality of the paper in an explorative process, that included a broad search for relevant works at first, followed by a filtering phase based on the criteria mentioned.

The grid used in Table 3.1 was selected based on the ML process. Every ML task begins with data acquisition. Data can be obtained from various sources, including research, industry and proprietary collection. It then has to be prepared and pre-processed in order to cleanse it from possible errors and artifacts and bring it into a form that is processable by a ML algorithm. Features are then extracted from the data, such that they represent the areas and relationships relevant for the classification task. A classification algorithm is applied to the features that delivers as its output a prediction about a dependent variable of interest. Finally, the performance of the classifier is evaluated using a test dataset. Apart from these steps, an additional column was added that describes any special characteristics of the respective approach.

Table 3.1: Overview over existing research in the field of encrypted network traffic classification

Paper	Data Source	Data Preparation	Features	Classification Algorithm	Output	Results	Special Characteristics
Lotfollahi et al., 2017	ISCX VPN-nonVPN dataset	Truncation/ padding of packets to 1500 bytes, masking of source and destination IP	Raw packets	Supervised: AE, CNN	Classification into 17 application classes	F_1 score of 0.95 (AE) and 0.98 (CNN)	
Wang et al., 2017	ISCX VPN-nonVPN dataset		Raw application layer or IP layer packets grouped by flows or sessions	Supervised: 1D CNN	Classification into 6 application classes and encrypted/unencrypted (12 classes in total)	Accuracy of 86% for best-performing classifier based on network sessions & all OSI layers	
Li and Moore, 2007	RedIRIS research campus backbone network trace	Omission of flows without initial packet and junk flows	12 highly discriminative TCP flow features	Supervised: DT, NN	Classification into 10 application classes	Accuracy of 99.8 % for best-performing C4.5 classifier with good temporal stability	
Lopez-Martin et al., 2017	RedIRIS research campus backbone network trace	Labeling of flows using the nDPI DPI tool, discarding of packets that could not be labeled	6 network flow features	Supervised: 9 different combinations of CNN and LSTM	Classification into 108 different applications	F_1 score of 96% for best-performing Convolutional and Recurrent Neural Network combination	Neural network architecture optimized experimentally
Vu, Bui, and Nguyen, 2017	NIMS SSH-nonSSH dataset	Generation of additional data for the underrepresented SSH traffic type using a Generative Adversarial Network	22 statistical network flow features	Supervised: SVM, DT, RF	Classification into SSH and nonSSH traffic	F_1 score of 95.43% for best performing Random Forest classifier	Attempt to tackle the imbalance problem
Zhang et al., 2015	Proprietary dataset of network traces, as well as WIDE and KEIO datasets merged together	Labeling of the flows using a DPI tool; removal of flows that could not be labeled; undersampling of classes with a large amount of flows	20 statistical unidirectional flow features	Semi-supervised: Flow clustering using K-means and classification using RF with manual inspection and classification of unknown traffic types	Classification into 7 applications, as well as an <i>unknown</i> class	Overall accuracy of over 95 %; outperforms various other tested methods	Attempt to identify and deal with unknown zero-day traffic
Taylor et al., 2017	Proprietary datasets containing traffic generated by 110 popular free smartphone applications	Noise filtering for parts of the datasets	54 statistical unidirectional burst-based flow features	Supervised: RF	Classification into 110 smartphone applications and one ambiguous class	Accuracy between 65.5% and 73.7% depending on dataset	Specific focus on the identification of smartphone applications
Chen et al., 2017	Proprietary internet traffic dataset		Distribution of 3 flow attributes over flow represented as an image, as well as the original raw flow attributes	Supervised: 2D CNN	Classification into 5 application layer protocols; Classification into 5 applications	Accuracy of 99.84% for protocol classification, 88.24% for application classification	Behavior of flow over time modeled in features
Höchst et al., 2017	Proprietary office network dataset	Labeling of the data using a proprietary Regular-Expression-based DPI tool	9 statistical flow features over exponentially growing time intervals	Unsupervised: Clustering using AE with manual labeling of the resulting clusters	Classification into 7 traffic types	F_1 score of 0.76 for best-performing classifier with 100 clusters and including a scaler for the input values	Attempt to tackle the problem of large amounts of unlabeled data
Bocchi et al., 2016	Proprietary dataset captured from a vantage point at an Internet Service Provider (ISP)	Anonymization of customer IP addresses; Extraction of flows from the packet capture	Features of the seed connectivity graph	Supervised: DT, RF	Classification into malicious and benign classes	Accuracy over 95% for best-performing RF classifier	Features based on hosts' connectivity characteristics

Paper	Data Source	Data Preparation	Features	Classification Algorithm	Output	Results	Special Characteristics
Rezaei and Liu, 2018	Proprietary dataset captured at UC Davis containing traffic from 5 labeled Google services; Waikato dataset containing unlabeled network traces; Ariel dataset containing captured network traffic with operating system and browser labels	For QUIC dataset: elimination of all non-QUIC-protocol traffic; Elimination of short flows	Raw packets sampled from individual flows	Semi-supervised: Pre-trained 1D CNN	Classification into 5 Google applications	Significant improvements of the accuracy of the model with pretraining, even if the pretraining data is from a different dataset	Attempt to tackle the problem of large amounts of unlabeled data
Bekerman et al., 2015	Mixture of proprietary network captures collected at Ben Gurion University and captures recorded by security companies Verint and Emerging Threats	Labeling of the data using a DPI tool and Verint's blacklist labeling; Undersampling of the majority class	22 Features from different layers of the network data stream	Supervised: NB, DT, RF	Classification into benign and malicious classes	Near perfect accuracy for best-performing RF classifier with known classes of malicious traffic; Very high accuracy for unknown classes	Features from different layers of the data stream

3.1 Lotfollahi et al., 2017

Lotfollahi et al.'s features are based on raw network packets on the IP layer as input data. Since most NN systems necessitate a fixed length input, they fix the size of an input item to 1500 bytes, the maximum transmission unit (MTU) of most networks. They keep the 20 byte IP header and the first 1480 bytes of the IP packet contents and truncate packets that exceed this length. To avoid classification of a packet based on the source or destination IP address, the corresponding fields are masked in the input data.

The authors then propose an approach to encrypted network classification using deep learning through NNs. More specifically, they examine two procedures, one using an AE and one using a CNN:

For the AE-based approach, Lotfollahi et al. use an AE consisting of five fully connected layers followed by a softmax layer. To avoid overfitting, they include a dropout rate, the probability for nodes to be randomly set to zero, of 0.05 during the training phase.

For the CNN-based approach, Lotfollahi et al. use a one-dimensional CNN consisting of two convolutional layers, a pooling layer, three fully connected layers and a softmax classifier. They implement the dropout technique to avoid overfitting.

Lotfollahi et al.'s main contribution lies in exploring an approach to classification with minimal pre-processing. Raw network packets are used as features, which omits the need to compute flow statistics or other more complex features. This approach, however, also brings along some limitations. The usage of raw packets results in the inclusion of IP packet contents that may be encrypted. This pseudorandom input data brings large amounts of noise into the ML system and may deteriorate the classification quality overall.

3.2 Wang et al., 2017

Wang et al. evaluate two choices of traffic representation. The first choice refers to the division of the network traffic into discrete units: They either divide traffic into flows, the 5-tuple source IP, source port, destination IP, destination port and transported layer protocol, or sessions, which consider flows in both directions, *i.e.* source IP/port and destination IP/port are interchangeable. The second choice refers to which protocol layers are considered by the classifier. Here, the authors again evaluate two options: Either only layer 7 of the OSI network model, *i.e.* the application layer, or all layers. Raw packets are grouped according to the representation described above and are then used as the input for the classifier. Since the input data size must be uniform, only the first 784 bytes of a flow or session are considered.

Wang et al. argue that a flow or session is essentially sequential data and that they may, therefore, implement a one-dimensional CNN to classify it. They assemble a CNN out of a mix of convolutional, max pooling and fully connected layers with a softmax classifier as the final step.

The main contribution of this work lies in the treatment of flows as sequential data. This approach allows for the utilization of behavioral characteristics of a traffic type over time and can, therefore, provide valuable information for the classification task. The one-dimensional CNN implemented, however, may not fully exploit this information since related information across sequential packets, such as changes of values in an IP header field, are not actually spatially contiguous in the one-dimensional concatenation of packets. Another limitation of the approach is the inclusion of IP packet contents, especially the usage of only the layer 7 packet contents. This makes the approach unsuitable for the classification of encrypted traffic.

3.3 Li and Moore, 2007

Li and Moore focus on efficiency in their approach. The main contribution in this direction lies in the selection of a small set of highly discriminative features. To achieve this, they divide their test data set into 10 different subsets and apply correlation-based filtering to each one, selecting features such that the correlation between the feature and the desired output is maximized. 10 behavior-features are then picked manually according to the following criteria: Each feature should appear in at least 1/3 of the subsets and each subset should contain at least 1/3 of the features. Source and destination port numbers are then also added, resulting in a list of 12 features. The efficiency is increased further by minimizing the size of the observation window while maintaining a high classification accuracy instead of using full flows. The experimentally determine that five packets are sufficient.

The authors then run experiments with various different learning algorithms, including C4.5 DTs, Naive Bayes Trees and Bayesian NNs. They determine that C4.5 delivers the best performance, although the differences to other the algorithms are minute. Since C4.5 also has the lowest testing complexity and their focus is efficiency, however, it is determined to be best.

Li and Moore contribute through their focus on efficiency and the ability of their proposed system to perform in a real-time environment. This is a prerequisite for many real-life applications for network traffic classification. The access to the TCP header required by some of their features, however, makes the approach unsuitable for some network layer encryption schemes.

3.4 Lopez-Martin et al., 2017

Lopez-Martin et al. use bidirectional network flows as a basis for classification. Different combinations of the following flow features are then evaluated: source port, destination port, packet direction, payload size and inter-arrival time. They also experiment with different time window sizes. A two-dimensional pseudo-image is then assembled out of the input data, consisting of the feature vectors in one dimension over the time steps in the other.

The authors evaluate 9 different combinations of CNNs and LSTMs for their ability to classify network flows accurately. Their best performing classifier consists of two convolutional, two batch normalization, two dropout, one LSTM and two fully connected layers. In order to combine these two types of neural networks, the output tensor of the CNN layers gets reshaped into a matrix that is compatible with the LSTM input shape.

Lopez-Martin et al. present an approach that is suitable for the classification of most encrypted traffic, since no access to any layer above the network layer is necessary. They contribute further by modeling network traffic as a two-dimensional pseudoimage. This representation is well-suited for the evaluation using CNNs and models the dynamic behavior of a network flow. Compared to the approach presented by Wang et al., a CNN is able to extract more meaningful information on the dynamic behavior of a flow from this representation. After the features have passed the convolutional layers, however, the sequential structure required by the LSTM may no longer be given. This may limit the effectiveness of their approach overall.

3.5 Vu, Bui, and Nguyen, 2017

Similar to other work, Vu, Bui, and Nguyen split network traffic into flows. 22 statistical features are calculated for each flow and used as a basis for classification.

They then propose an approach to handling the problem of imbalance in network traffic classification data. The imbalance problem refers to the fact that certain types of traffic may be over- or underrepresented in collected network traffic data. This can lead to problems when using said data to train a machine learning model. The authors tackle this issue through the implementation of a generative adversarial network (GAN). More specifically, they use an auxiliary classifier GAN (AC-GAN). It is an extension of the GAN approach that includes a class label in the input for the generator and the output of the discriminator. It can then be used to synthesize data for underrepresented traffic classes in the training data. The augmented dataset is used to train a classifier. The authors experiment with three supervised classification approaches: SVMs, DTs and RFs.

Vu, Bui, and Nguyen contribute by tackling the issue of unbalanced data. This is a prevalent problem inherent to network traffic and should be addressed by any network traffic classification approach. The implementation of a generative NN model allows to use the overrepresented classes to their full extent and to generate additional values for the others. Looking at the results described in the publication, however, reveals that the impact of the AC-GAN-based data generation is minimal. It should also be mentioned here that the approach presented by Vu, Bui, and Nguyen is suitable for the classification of encrypted traffic since their flow features are computed purely from the network layer information available.

3.6 Zhang et al., 2015

Similar to other approaches, Zhang et al. use 20 statistical flow features as the input for their classifier. Their classification approach then consists of three components:

Unknown Discovery: In order to extract zero-day traffic (formerly unseen traffic types), flows are first clustered using the k-means clustering algorithm. If a cluster does not contain any labeled samples, all flows contained in said cluster are given the label *unknown*. A RF classifier is then trained on those flows to gain a more generic representation of the *unknown* class.

BoF-Based Classification: Flows are grouped into *Bags of Flows* (BoFs) based on a simple correlation metric, the presence of the same destination IP, destination port and transport layer protocol within a short time span. They are then then classified using a RF classifier. The majority vote class of the BoF then gets applied to all flows contained in it.

System Update: In order to correctly label unknown traffic, zero-day traffic gets clustered using k-means. A sample of three flows per cluster is extracted and made available for manual inspection. The label chosen by the inspector is then applied to the entire cluster.

Zhang et al.'s approach sets itself apart through the inclusion of unknown detection. With network traffic characteristics being constantly in flux and new network traffic types arising regularly, this lays the foundation for the continued maintenance of their model and enables its sustainability in the long term. It should be noted, however, that the access to the transport layer required by their approach makes it unsuitable for some network layer encryption schemes.

3.7 Taylor et al., 2017

Taylor et al. base their features on flows, defined in their work as follows: Network packets are grouped into *bursts*, characterized by a certain threshold on packet inter-arrival time. Flows are then defined as sets of packets within a burst that have the same destination IP address. A set of 54 aggregate statistics is then extracted from these flows and used as features.

As can be seen in Figure 3.1, the authors implement a two step classification process to account for ambiguity in smartphone generated traffic:

Preliminary Classifier: A RF classifier is run over the preliminary training set and then evaluated using the preliminary testing set. False classifications are identified and re-labeled into the *Ambiguous* class to create the reinforced training set.

Reinforced Classifier: A secondary RF classifier is now trained on the reinforced training set and now includes the *Ambiguous* class. This is the classifier that will then be used for the actual classification task.

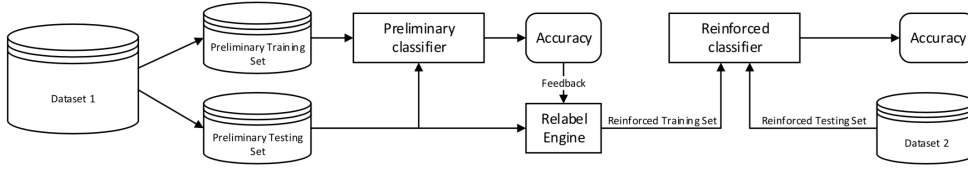


Figure 3.1: Illustration of Taylor et al.’s approach to classification with ambiguity detection [25]

Taylor et al.’s main contribution is the tailoring of their approach to the specificities of traffic generated by smartphone applications. Specifically, they account for ambiguities that may appear, for example, due to shared advertisements libraries. This characteristic may not be unique to smartphone traffic, making their approach interesting for any network traffic classification task. The approach presented is also suitable for the classification of encrypted traffic. The results show that the classification is robust across devices and system versions, however not application versions.

3.8 Chen et al., 2017

Chen et al. use bidirectional flows as the basis for their features. From these flows, they extract a small set of raw flow information: the sequence of packet sizes, packet inter-arrival times and packet directions for the first 9 packets of a flow. They then design their features to describe the static and the dynamic behavior of the flow: With $I_1, \dots, I_j, \dots, I_n$, the sequence of information, they compute the marginal probability distribution $P(I_j)$, as well as the conditional probability distribution $P(I_{j+1}|I_j)$. These distributions are then expressed as a 6-channel image using the Reproducing Kernel Hilbert Space Embedding (RHKS). The server IP address is used as an additional feature.

Thanks to the representation of the features as an image, they can now easily be processed using a CNN. Chen et al. use a CNN consisting of two convolutional, two max pooling and three fully connected layers. The server IP address and the original flow attributes are fed directly to the fully connected layers.

The approach describes an innovative application of a CNN to the traffic classification problem that is also suitable for encrypted traffic. The usage of probability distributions over the duration of a flow also allows the modeling of the dynamic behaviour of a flow. The true effectiveness of the approach, however, remains unclear. Since the original flow attributes, as well as the server IP address are fed into the NN apart from RHKS embedded probability distributions, the actual impact of using this unconventional feature representation cannot be gauged from the results.

3.9 Höchst et al., 2017

Höchst et al. use 8 statistical flow features, as well as the Differentiated Services Codepoint (DSCP) IP header field in their approach. The DSCP field is the successor to the Type-of-Service header field, used to indicate the Quality of Service demands of the traffic. These 9 features are then calculated for 12 incrementally growing time intervals for each flow in both directions in order to more heavily emphasize traffic at the beginning of a flow. This results in a total of 216 features as a basis for classification.

The authors use this feature vector to train an AE. After training, a softmax layer is applied to the encoder output. The largest value from this layer is then used to form a cluster, which should be highly discriminative thanks to the nature of the AE. These clusters are labeled semi-automatically with application types, such as browsing, downloading or video streaming, similar to the process described in [30].

The main contribution of the approach lies in the exploitation of unlabeled network traffic. The semi-supervised classifier automatically labels traffic where some labels exist, but also allows for the detection of unknown or ambiguous classes that can then be inspected manually. The computation of flow statistics for different time intervals also forms an effective but simple approach to modeling the dynamic behavior of a network flow. Thanks to the exclusive usage of information that is available from the IP header, the classifier is also suitable for most encrypted traffic. One limitation in the usability of the approach, however, is that the number of clusters identified by the AE has to be chosen manually.

3.10 Bocchi et al., 2016

Bocchi et al. base their features in a model for the social behavior of a host, the *seed connectivity graph*. They define a seed event as an event that should be classified as malicious or benign. They then create the *host connectivity graph* as follows: For each occurrence of the seed for a single host, they extract the ordered set of events in the temporal window around the host as a snapshot. They extract common patterns in the snapshots on each IP layer represent them as a graph and collapse them. The *seed connectivity graph* is then created by merging the host connectivity graphs for a specific seed event over all hosts. The features used are then properties of the seed connectivity graph, including topology properties, http header parameters, syntax properties extracted from domain names and occurrence properties of specific events. Bocchi et al. then experiment with different DT based classifiers, including J48 – an open-source implementation of the C4.5 DT algorithm –, J48 with bagging and RFs.

The approach sets itself apart by moving past the boundaries of a flow and instead modeling a host's behavior, focusing on the co-occurrence of events through the seed connectivity graph. This captures both dynamic properties of a host, as well as its social behavior, since the observations are no longer limited by a single IP address and port number combination. This opens up new possibilities for traffic classification. The approach does, however, require access to the HTTP header, making it unsuitable for many encryption

schemes. The construction of the seed connectivity graph also requires extensive pre-processing, making this solution challenging for real-time classification.

3.11 Rezaei and Liu, 2018

Rezaei and Liu base their features in sampled packets from bidirectional flows. They experiment with different sampling techniques, including fixed-step sampling, random sampling and sampling with an incrementally increasing time step. A one-dimensional vector with two channels is then created using these packets where the first channel describes the packet inter-arrival time and the second channel describes the packet size and direction.

The authors then implement a one-dimensional CNN with three convolutional, two pooling and one fully connected layer. The network is pre-trained on the sampled packets to estimate 24 statistical flow features in order to benefit from the large amounts of unlabeled data usually present in network traffic classification problems. Two more fully connected layers are then added to the model and it is re-trained on labeled data for classification.

Rezaei and Liu contribute an approach that makes effective use of CNNs ability of recognizing spatially correlated features to model the dynamic behaviour of a flow. The one-dimensional two-channel feature vector fulfills the requirement for a spatial structure for both the case of fixed-step and incremental sampling. This sequentiality is not given, however, in the case of random sampling. They also exploit unlabeled data through a relatively simple pre-training implementation. Since only IP header fields are used, the approach is suitable for most encrypted traffic. The subset of information used to construct the features is, however, very small. Using a larger set of flow statistics may benefit the approaches' effectiveness.

3.12 Bekerman et al., 2015

Bekerman et al. generate features in 4 different resolutions from the internet, transport and application layer:

Transaction: A Transaction describes an interaction between a client and a server. Only HTTP, DNS and SSL transactions are handled in Bekerman et al.'s approach.

Session: A Session is characterized by the 4-tuple source and destination IP, as well as the corresponding port numbers. Specifically, a Session describes either a TCP or a UDP session.

Flow: A Flow is a grouping of sessions during an aggregation period, which is characterized by a maximum idle time between sessions.

Conversation Window: A Conversation Window is a group of flows during an observation period. It can be defined by two IP addresses or by a group of network resources, such as autonomous systems.

For each one of the resolutions, Bekerman et al. extract various features, many of which are dependent on the application layer or the transport layer protocol used. They then experiment with three different ML algorithms: NB, the J48 DT algorithm and RFs, of which, in most cases, RFs perform best.

Bekerman et al. contribute by integrating information from different layers of the OSI model and of the interaction between two hosts to form features. Due to the exclusive focus on certain protocol types, however, the approach is both not applicable to many forms of network traffic, as well as encrypted traffic.

Chapter 4

Experimental Classification with TensorFlow

In order to further explore the possibilities and difficulties of encrypted network traffic classification, an experimental system was designed and implemented using the Tensorflow framework for NN based machine learning. It implements a semi-supervised, AE-based approach to the traffic classification problem. The implementation was inspired by some of the research investigated for Chapter 3. A NN-based approach was selected due to the current popularity of this area of research. It also allows for extensive configuration and is flexible its possible applications. The system was finally tested on a large dataset consisting of collections of flow statistics.

4.1 Tech Stack

Tensorflow

Tensorflow is a system for the implementation and deployment of large-scale machine learning models developed by the *Google Brain* team [6]. A successor to the Google internal Project DistBelief, it was released as an open source library in 2015.

Tensorflow is designed to simplify the real-world use of machine learning systems: It runs in a wide range of different environments, from mobile platforms, such as iOS and Android, to large distributed systems with thousands of GPUs. It combines the flexibility to allow for quick experimentation with the performance necessary for the productive deployment of machine learning models. Tensorflow provides stable APIs for C and Python, however a number of implementations for other programming languages are available.

Keras

Keras is a high-level Python API for the implementation of neural network based machine learning systems capable of running on top of various different backends including

Tensorflow [10]. Keras is designed for ease of use, offering consistent and simple APIs. A neural network model is built out of simple modules, including layers, optimizers, cost functions and activation functions. The Keras code for a simple neural network classifier is shown in Figure 4.1.

```
model = Sequential()
model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5, batch_size=32)
```

Figure 4.1: A simple neural network classifier build using the Keras API [10]

Pandas

Pandas is an open source data manipulation and analysis library for Python [19]. It is designed to work with relational data, such as tabular, time series or labeled matrix data. The two primary data structures, the one-dimensional *Series* and the two dimensional *DataFrame*, provide a wide range of functionality for processing, manipulation, analysis and visualization tailored specifically to the requirements encountered in data analysis tasks.

Scikit-Learn

Scikit-learn is an open source ML library for Python [22]. It provides a wide range of functionality for ML, including implementations of popular algorithms for classification, regression, clustering and dimensionality reduction. Apart from this, scikit-learn also provides useful utilities for problems surrounding the ML task, such as model selection (the search for optimal model parameters), feature extraction and pre-processing (the transformation of data into a representation that is more suitable for processing with ML algorithms).

4.2 Dataset

The dataset used for this experiment was obtained from Kaggle, a platform for data science education and competitions [21]. It was collected from a network section at the Universidad del Cauca, Colombia, over six days in 2017. Similar to [27], [30] or [1], the dataset contains a large number of 87 unidirectional flow properties and statistics that include: source and destination IP addresses, ports, interarrival times and the layer 7

protocol. Most of these values were generated using CICFlowmeter [8]. The application layer protocol was identified using the DPI tool ntopng [18]. In total, the dataset contains 3'577'396 individual data points.

4.3 Data Preparation

The preparation of the dataset included the following steps:

1. **Elimination of rows with missing values:** A neural network is not capable of dealing with missing values in the input vectors. Therefore, incomplete data points have to be addressed in the input dataset. This can be done according to two strategies: Missing fields may either be filled with some computed value (*e.g.* the mean of all values) or eliminated all together. For the sake of simplicity, the latter was chosen for this experiment.
2. **Elimination of the source and destination IP fields:** Since the client IP address does not contain information about the application responsible for the generation of a certain network flow, and it was not possible to distinguish between client and server IP address in the dataset available, the columns containing IP addresses were eliminated.
3. **Elimination of application protocols with less than 50'000 rows:** As it is often the case with network traffic data, the dataset available was highly unbalanced concerning the representation of the different application protocols, ranging from one flow for the most underrepresented application protocol to almost one million flows for the most overrepresented. Since ML needs a certain amount of data to perform properly, all application protocols that had less than 50'000 flows were eliminated, leaving the dataset with 8 application protocols.
4. **Splitting into AE and classifier training data:** Since the training of the model was performed in two phases, the dataset was partitioned into 90% data for the AE-based pre-training and 10% data for the classifier training. The layer 7 protocol column was eliminated for the AE pre-training data. This allows the evaluation of the model's ability to benefit from unlabeled training data.
5. **Splitting into training, test and validation data:** The AE data was split into a 67% partition for training and a 33% partition for validation during the training phase. The classifier data was split into a 45% partition for training, a 22% partition for validation during the training phase and a 33 % partition for a final test of the resulting classifier.
6. **One-hot encoding of application protocol values:** Since NNs are only able to process numerical values and the application protocol values in the dataset are categorical, a so called one-hot encoding needs to be performed. This involves converting the values for the n categories into a n -dimensional vector where the value for the appropriate category is one, the rest is zero. These encoded values

can then be used to train a classifier. The output of the classifier will then also be encoded in the same fashion.

7. **Save prepared data into *Pickle* files:** Since the aforementioned preparation steps are time consuming and resource intensive, the prepared and partitioned dataset is saved into *Pickle* files, a serialized representation for a Python object. This way, multiple experiments can be run on the dataset without having to perform the preparation steps each time. It also enables a more representative comparison between experiments since the dataset does not vary due to possible non-determinism in the preparation steps.

4.4 Classification

The approach devised for this experimentation phase was inspired by some of the research described in Chapter 3: Similar to [20] and [30], the system is semi-supervised in order to test its ability to draw benefits from a large amount of unlabeled data, which is often the reality in problems related to encrypted network traffic. More specifically, the system implements a NN that is pre-trained on unlabeled data in an approach similar to [20]. Like the classifiers described in [7] and [14], the experimental approach implements an AE, however, the way that its capabilities are taken advantage of are different.

An AE-based system was developed that uses as input the aggregate flows statistics prepared in the previous step. An AE was first trained on a large part of the dataset to reproduce the input with an significant reduction in dimensionality in the encoder part of the network, the theory being that this setup would isolate the discriminative parts of the input and would thus pre-train the network to effectively process aggregate flow statistics data. After the pre-training stage, the weights in the encoder part could then be frozen and the decoder part could be replaced by a set of layers for classification. The classification layers could then be trained using labeled data to assign the discriminative features generated by the encoder to a specific application layer protocol.

During the experimentation phase, the following parameters were adjusted: The depth of the encoder and decoder layers, the size of the encoded features, the number of epochs used for training of the AE and the classifier. The proposed implementation will be explained in detail in the following.

Implementation

In summary, the neural network pipeline consists of the following components: Scaler, encoder component, decoder component and classifier component. In addition, it involves some considerations regarding the method of training used. The following values are relevant in the explanation:

- dim_{in} , the input dimension defined by the size of the feature vector

- dim_{enc} , the encoded feature dimension, a parameter
- dim_{class} , the classification output dimension, defined by the number of labels in the dataset
- n_{lay} , the number of layers in the encoder and decoder components, a parameter
- ep_{AE} , the number of training epochs for the AE, a parameter
- ep_{class} , the number of training epochs for the classifier, a parameter

Scaler: In order to avoid bias towards one of the values in the feature vector, it needs to be ensured that none of them outweighs the others in magnitude. To achieve this, all values are scaled to a range between 0 and 1 using min-max scaling. This technique assesses the minimum and the maximum of each value in the feature vector over the entire training dataset. This is then used to transform all values in the training dataset, as well as the test dataset or any other subsequent values passed to the classifier.

The Encoder Component: The encoder component consists of the following layers:

1. An Input layer with dimensions dim_{in} , ReLU activated
2. A Dropout layer with dropout rate 0.2
3. $n_{lay} - 1$ Fully Connected layers, ReLU activated, where layer $i, i \in \{1, \dots, n_{lay} - 1\}$ has dimensions

$$dim_i = dim_{enc} + (dim_{in} - dim_{enc}) * \frac{n_{lay} - i}{n_{lay}}$$

4. An Output layer with dimensions dim_{enc} , ReLU activated

The Decoder Component: The decoder component is structured symmetrically to the encoder component. It consists of the following layers:

1. $n_{lay} - 1$ Fully Connected layers, ReLU activated, where layer $i, i \in \{1, \dots, n_{lay} - 1\}$ has dimensions

$$dim_i = dim_{enc} + (dim_{in} - dim_{enc}) * \frac{i}{n_{lay}}$$

2. An Output layer with dimensions dim_{in} , ReLU Activated

The Classifier Component: The classifier component consists of the following layers:

1. A Fully Connected layer with dimensions dim_{class} , ReLU activated
2. A Dropout layer with dropout rate 0.2
3. An Output layer with dimensions dim_{class} , Softmax activated

Training Considerations: The AE was trained for ep_{AE} epochs. Since the goal of the AE is regression, the Mean Squared Error was used as the loss function. After AE training, the decoder component was replaced with the classifier component. The classifier was then trained for ep_{class} epochs. Since the goal is classification into a categorical variable, the Categorical Cross-Entropy was used as the loss function. In addition, training for both networks was accelerated using the Adam optimizer.

4.5 Results

The computational resources, most importantly CPU power and memory size, were limited in the evaluation of this experiment since it was carried out on a consumer grade laptop computer. Due to this, an exhaustive grid search for optimal parameters was not possible. Instead, the parameters were optimized heuristically, starting out with a reasonable value, then varying them individually to determine the performance gains or losses. This technique led to the set of parameters shown in Table 4.1.

n_{lay}	4
dim_{enc}	10
ep_{AE}	10
ep_{class}	5

Table 4.1: Best-Performing Parameters in the Experimental Classifier

The classifier can then be evaluated in two comparisons: First, the results from the classification of the test dataset were compared to the results from the classification of the training dataset. If the results are similar, this suggests that the classifier is not overfitting to the training dataset. Second, the test dataset classification results were compared to the results generated using a baseline model. This technique often makes a more meaningful statement about the performance of a model than mere performance measures, such as accuracy. In this case, the baseline model was a simple NN classifier with 10 fully connected layers and one dropout layer that was trained with 66% and tested with 33% of the entire labeled dataset. Similar results indicate that the performance of the model did not suffer from the implementation of a semi-supervised approach where only a small part of the of the data is labeled and the large part is used only for unsupervised pre-training. Table 4.2 shows the results with *Categorical Accuracy* being the average accuracy over all classes, *Top 5 Categorical Accuracy* the average accuracy of the 5 best-performing classes. The differences between classification of the test and training dataset show that there may be some overfitting taking place. The accuracy is significantly worse overall, but slightly better for the 5 best-performing classes. This suggests that the model might be biased more strongly towards the majority classes. The results are, however, comparable to those generated by the baseline model, even performing slightly better in accuracy over all classes. This validates the approach of pre-training with an AE system, suggesting instead that the model may be struggling with characteristics of the data itself since even the fully supervised baseline model is not able to generate significantly better results.

Model	Categorical Acc.	Top 5 Categorical Acc.
AE-Based Approach on Test Dataset	31%	96%
AE-Based Approach on Training Dataset	45%	94%
Baseline Model on Test Dataset	27%	100%

Table 4.2: Results of classification using the proposed AE-based approach on the test dataset, the training dataset and a baseline model

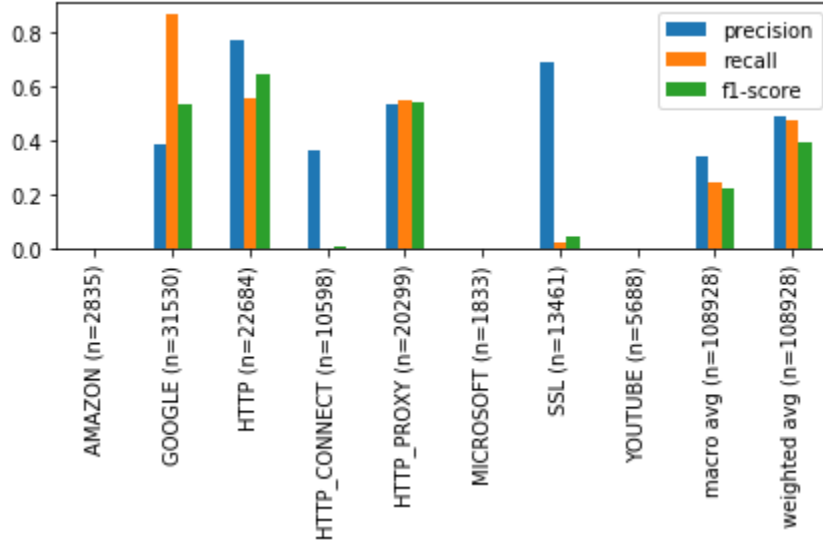


Figure 4.2: Results per class of classification using the proposed AE-based approach on the test dataset

In general, however, further improvement of the model is certainly possible. While it classifies certain classes of network flows well, the performance for other classes, indicated by the overall categorical accuracy of 31%, makes it unusable in practice in its current state. The comparable results generated by the baseline model suggest that the problem may lie with the data preparation steps taken or the data itself, rather than with the classification approach. Figure 4.2 summarizes the classification results per application protocol class. It shows that the more highly represented classes, GOOGLE, HTTP and HTTP_PROXY tend to perform best. The SSL and the HTTP_CONNECT classes that have less support show some precision but are severely lacking in recall, indicating that there are a lot of false negatives in the classification for these classes. The worst performance is generated by the least represented classes, AMAZON, MICROSOFT and YOUTUBE, that do not seem to be captured by the model at all. This suggests that a prior balancing of the dataset may benefit the model's effectiveness.

Chapter 5

Conclusion & Lessons Learned

A survey of existing research in the area of network traffic classification using ML-based techniques was conducted. It showed that, overall, ML shows promising potential for this essential task in network management and security. The publications investigated cover a wide range of approaches with regards to classification, including more traditional algorithms like SVMs and DTs, as well as more novel techniques involving NNs. They also differ with regards to feature engineering, including flow-statistics-based, as well as packet-based features. Each approach has its advantages and limitations that were listed in this work and no single approach stood out in terms of delivering the best performance. This also suggests that the most complex model may not always be the most accurate. It should also be noted that existing approaches are often highly optimized for and tailored to the problem domain and the dataset at hand, indicated by the conspicuously good results reported in most papers. This may make it hard to transfer them to real-life problems and requires adaption.

An experimental classifier was implemented that followed a semi-supervised approach in order to benefit from the large amounts of unlabeled data that usually exist in network traffic classification tasks. The resulting model was evaluated on a network flow statistics dataset. It showed high accuracy for some traffic classes, but performed poorly for others. The evaluation also showed that the approach was moderately robust to overfitting. The evaluation of a fully supervised baseline model on the same dataset produced results that were comparable to those of the semi-supervised approach. This validates the approach in its ability to utilize unlabeled data as parts of its training procedure, and suggests that the limitations in accuracy for some classes may lie with the dataset itself or the data preparation tasks performed. Further experiments in this area could be the exclusion of some parts of the feature vector, the balancing of classes prior to model training or accounting for ambiguity.

The process of building and evaluating the experimental classifier itself revealed some important lessons about ML and data analytics tasks. ML is hard to analyze and hard to debug. Especially with non-descriptive models, such as NNs, the reason for bad performance can often not be isolated or explained. Instead, the building of an accurate model requires a lot of experimentation and heuristic approaches. This optimization of parameters requires either a large amount of computational resources or time.

A domain problem encountered in the real world is often not trivial to translate into one that is solvable using an ML algorithm. This requires extensive consideration of the characteristics of the data at hand, innovative feature engineering in order to profit from it as best as possible, as well as knowledge of the capabilities of different ML approaches. The approaches themselves come with their own limitations that need to be accounted for, such as the fixed input size of the feature vector and floating point formatting of the individual values required by NNs.

List of Figures

2.1	A tree showing survival of passengers on the Titanic [29]. <i>sibsp</i> is the number of spouses or siblings aboard	5
2.2	The basic composition of a neuron with input values x_1, x_2, x_3 and output value $h_{w,b}(x)$ [17]	8
2.3	The ReLU function, used commonly as an activation function in NNs [17] .	8
2.4	A very basic NN with input layer L_1 , one hidden layer L_2 and output layer L_3 [17]. x_1, x_2 and x_3 are the inputs to the NN, $h_{w,b}(x)$ is the output. . . .	8
2.5	A CNN classifier with a two-dimensional input [17]	9
2.6	An abstract representation of a RNN as a multilayer network, illustrating the method of preserving information across successive runs [17]	10
2.7	Illustration of an AE showing the reduction of dimensionality with encoder layers 1 & 2 and decoder layer 3 [14]	11
3.1	Illustration of Taylor et al.'s approach to classification with ambiguity detection [25]	20
4.1	A simple neural network classifier build using the Keras API [10]	26
4.2	Results per class of classification using the proposed AE-based approach on the test dataset	31

List of Tables

2.1	Pros and Cons of the Naïve Bayes Algorithm [26]	4
2.2	Pros and Cons of the Decision Tree Algorithm [9] [26]	5
2.3	Pros and Cons of the RF Algorithm [9] [15]	6
2.4	Pros and Cons of the SVM Algorithm [9] [26]	7
2.5	Pros and Cons of NNs [15]	7
2.6	Pros and Cons of CNNs [17]	9
2.7	Pros and Cons of RNNs [17]	10
2.8	Pros and Cons of AEs [3] [11]	11
2.9	Pros and Cons of GANs [27]	12
3.1	Overview over existing research in the field of encrypted network traffic classification	14
4.1	Best-Performing Parameters in the Experimental Classifier	30
4.2	Results of classification using the proposed AE-based approach on the test dataset, the training dataset and a baseline model	31

Bibliography

- [1] Dmitri Bekerman et al. “Unknown malware detection using network traffic classification”. In: *2015 IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2015, pp. 134–142.
- [2] Enrico Bocchi et al. “MAGMA network behavior classifier for malware traffic”. In: *Computer Networks* 109 (2016), pp. 142–156.
- [3] Ramraj Chandradevan. *AutoEncoders are Essential in Deep Neural Nets*. <https://towardsdatascience.com/autoencoders-are-essential-in-deep-neural-nets-f0365b2d1d7c>. Accessed: 2019-07-11. 2019.
- [4] Zhitang Chen et al. “Seq2img: A sequence-to-image based approach towards ip traffic classification using convolutional neural networks”. In: *2017 IEEE International Conference on Big Data (Big Data)*. IEEE. 2017, pp. 1271–1276.
- [5] Rohinth Gandhi. *Naive Bayes Classifier*. <https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c>. Accessed: 2019-07-08. 2019.
- [6] Sanjay Surendranath Girija. “Tensorflow: Large-scale machine learning on heterogeneous distributed systems”. In: *Software available from tensorflow.org* (2016).
- [7] Jonas Höchst et al. “Unsupervised traffic flow classification using a neural autoencoder”. In: *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*. IEEE. 2017, pp. 523–526.
- [8] ISCX. *Github – CICFlowMeter*. <https://github.com/ISCX/CICFlowMeter>. Accessed: 2019-06-25. 2019.
- [9] Gareth James et al. *An introduction to statistical learning*. Vol. 112. Springer, 2013.
- [10] Keras. *Keras: The Python Deep Learning library*. <https://keras.io>. Accessed: 2019-05-30. 2019.
- [11] Hugo Larochelle et al. “Exploring strategies for training deep neural networks”. In: *Journal of machine learning research* 10.Jan (2009), pp. 1–40.
- [12] Wei Li and Andrew W Moore. “A machine learning approach for efficient traffic classification”. In: *2007 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. IEEE. 2007, pp. 310–317.
- [13] Manuel Lopez-Martin et al. “Network traffic classifier with convolutional and recurrent neural networks for Internet of Things”. In: *IEEE Access* 5 (2017), pp. 18042–18050.
- [14] Mohammad Lotfollahi et al. *Deep Packet: A Novel Approach For Encrypted Traffic Classification Using Deep Learning*. 2017. arXiv: 1709.02656 [cs.LG].
- [15] Farhad Malik. *Machine Learning Algorithms Comparison*. <https://medium.com/fintechexplained/machine-learning-algorithm-comparison-f14ce372b855>. Accessed: 2019-07-11. 2019.

- [16] Dominic Monn. *Denoising Autoencoders explained*. <https://towardsdatascience.com/denoising-autoencoders-explained-dbb82467fc2>. Accessed: 2019-07-11. 2019.
- [17] Vibhor Nigam. *Understanding Neural Networks. From neuron to RNN, CNN, and Deep Learning*. <https://towardsdatascience.com/understanding-neural-networks-from-neuron-to-rnn-cnn-and-deep-learning-cd88e90e0a90>. Accessed: 2019-05-23. 2019.
- [18] ntop. *ntopng – High-Speed Web-based Traffic Analysis and Flow Collection*. <https://www.ntop.org/products/traffic-analysis/ntop/>. Accessed: 2019-06-25. 2019.
- [19] Pandas. *Pandas: Python Data Analysis Library*. <https://pandas.pydata.org>. Accessed: 2019-07-02. 2019.
- [20] Shahbaz Rezaei and Xin Liu. “How to Achieve High Classification Accuracy with Just a Few Labels: A Semi-supervised Approach Using Sampled Packets”. In: *arXiv preprint arXiv:1812.09761* (2018).
- [21] Juan Sebastian Rojas. *Kaggle – IP Network Traffic Flows Labeled with 75 Apps*. <https://www.kaggle.com/jsrojas/ip-network-traffic-flows-labeled-with-87-apps>. Accessed: 2019-06-25. 2019.
- [22] scikit-learn. *scikit-learn: Machine Learning in Python*. <https://scikit-learn.org/stable/>. Accessed: 2019-07-02. 2019.
- [23] Irhum Shafkat. *Intuitively Understanding Variational Autoencoders*. <https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf>. Accessed: 2019-07-11. 2019.
- [24] Devin Soni. *Supervised vs. Unsupervised Learning*. <https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d>. Accessed: 2019-07-11. 2019.
- [25] Vincent F Taylor et al. “Robust smartphone app identification via encrypted network traffic analysis”. In: *IEEE Transactions on Information Forensics and Security* 13.1 (2017), pp. 63–78.
- [26] Danny Varghese. *Comparative Study on Classic Machine learning Algorithms*. <https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222>. Accessed: 2019-07-11. 2019.
- [27] Ly Vu, Cong Thanh Bui, and Quang Uy Nguyen. “A deep learning based method for handling imbalanced problem in network traffic classification”. In: *Proceedings of the Eighth International Symposium on Information and Communication Technology*. ACM. 2017, pp. 333–339.
- [28] Wei Wang et al. “End-to-end encrypted traffic classification with one-dimensional convolution neural networks”. In: *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE. 2017, pp. 43–48.
- [29] Wikipedia. *A tree showing survival of passengers on the Titanic*. https://en.wikipedia.org/wiki/Decision_tree_learning\#/media/File:CART_tree_titanic_survivors.png. Accessed: 2019-03-25. 2019.
- [30] Jun Zhang et al. “Robust network traffic classification”. In: *IEEE/ACM Transactions on Networking (TON)* 23.4 (2015), pp. 1257–1270.