TU WIEN Informatics

Study of 2D Representations of Encrypted Network Traffic for Attack Detection with Deep Learning

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Data Science

eingereicht von

Martin Pichler, BSc
Matrikelnummer 01429133

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.-Ing. Tanja Zseby
Mitwirkung: Senior Scientist Dr.techn. Felix Vazquez Iglesias

SIGNATURE INFORMATION	**Signatory**	Tanja Zseby
	Date/Time-UTC	2022-05-15T15:43:33+02:00
	Verification	Information about the verification of the electronic signature can be found at: https://www.signaturpruefung.gv.at
Note	This document is signed with a qualified electronic signature. According to Art. 25 para. 2 of the Regulation (EU) No 910/2014 of 23. July 2014 ("eIDAS-Regulation") it shall have the equivalent legal effect of a handwritten signature.	

Wien, 14. Mai 2022

Martin Pichler | Tanja Zseby

TU WIEN Informatics

Study of 2D Representations of Encrypted Network Traffic for Attack Detection with Deep Learning

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Data Science

by

Martin Pichler, BSc
Registration Number 01429133

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.-Ing. Tanja Zseby
Assistance: Senior Scientist Dr.techn. Felix Vazquez Iglesias

Vienna, 14th May, 2022

Martin Pichler	Tanja Zseby

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Martin Pichler, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 14. Mai 2022

Martin Pichler

Danksagung

Ich habe viel Unterstützung und Hilfe erfahren, ohne der ich es nicht geschafft hätte diese Arbeit fertigzustellen.

Ich möchte mich bei meinem Betreuer Dr. techn. Félix Iglesias bedanken, der seine Expertise, Einblicke und Ideen mit mir geteilt hat. Ohne deiner Hilfe hätte ich die Arbeit nicht fertigstellen können. Ich möchte außerdem allen anderen aus dem Ïnstitute of Telecommunications"die mich Unterstütz haben meinen Dank aussprechen. Insbesondere möchte ich mich bei Professor Tanja Zseby und bei Fares Meghdouri bedanken.

Weiters möchte ich mich bei meinen Eltern für ihre andauernde Unterstützung in meinem Studium bedanken. Ohne euch wäre es mir nicht möglich gewesen mein Studium abzuschließen. Danke! Zudem möchte ich allen meinen Freunden und Kollegen danken, die mir durch stressige Zeiten geholfen haben.

Acknowledgements

I received a lot of support and assistance during the creation of this thesis, whithout which I would not have been able to finish it.

First I want to thank my supervisor Dr. techn. Félix Iglesias, who shared his expertise, insight and ideas with me. Your help during the thesis was invaluable for me and withouth it I could not have finished the thesis. I also want to thank all the other helpful people of the Institute of Telecommunications who offered their support, especially Professor Tanja Zseby and Fares Meghdouri.

Additionally I want to thank my parents for constantly supporting me through all of my studies. I would not be able to finish my thesis if it wasn't for your help and patience. Thank you! Also I want to thank all of my friends and colleagues who helped me through stressful times.

Kurzfassung

In einer immer vernetzteren Welt steigt die Menge an gesendeten Daten ständig an. Gleichzeitig steigt auch der Bedarf an schnellen und robusten Systemen zur Erkennung von Netzwerkangriffen. Solche Angriffe zu erkennen wird durch die hohe Menge an verschlüsselten Daten erheblich erschwert. Deep Learning (DL) zeigt in vielen Anwendungsbereichen herausragenden Ergebnisse, beim Erkennen von Netzwerkangriffen zeigt sich allerdings kein signifikanter Unterschied zu bestehenden Machine Learning (ML) Ansätzen. Wir schlagen eine neue bildbasierte Form zur Darstellung von Datenverkehr vor, welche sich die starke Leistung in der Mustererkennung von Convolutional Neural Networks (CNN) zu nutze machen kann. Dazu verwenden wir modernste synthetische Datensätze und Datensätze welche aus echten Datenverkehr erstellt wurden. In Kombination mit aktuellen Deep Learning Ansätzen wie Siamese Networks (SN) oder Few-Shot Learning untersuchen wir die Lesitung von Detektoren für eine binären Klassifikation und für eine Klassifikation mit mehreren Klassen.. Wir berücksichtigen modernen Datenverkehr in dem wir unsere Datensätze einschränken, und nur Attribute verwenden, welche auch in verschlüsselter Kommunikation vorliegen. Wir kombinieren mehrere Granularitäten von Netzwerkkommunikationen zu einem neuen Datensatz, welchen wir multikey nennen. Mit diesen multikey Ansatz versuchen wir so viel Informationen wie möglich aus den zur Verfügung stehenden Daten auszulesen. Es zeigt sich, dass verschiedene Modelle und Architekturen nur wenig Einfluss auf die Erkennungsleistung haben. Bilder welche wir aus multikey Daten generieren, verbessern die Resultate zu bestehenden Darstellungen von Datenverkehr, liefern aber keine besseren Ergebnisse als andere Machine Learning Modelle, welche auch mit multikey basierten Datensätzen trainiert wurden. Daraus schließen wir, dass die Erkennungsleistung vor allem von der Qualität und Genauigkeit der Daten abhängt. Als Resultat dieser Arbeit präsentieren wir eine bild-basierte Darstellung von Datenverkehr zur Verwendung mit CNNs. Außerdem beschreiben wir einen Schritt für Schritt Prozess zur Evaluierung von Modellen für Netzwerkangriffe und präsentieren mehrere Optimierungen im Bezug auf Siamese Networks.

Abstract

With an ever more connected world, volume of network traffic surges and so does the need for fast and reliable Network Intrusion Detection Systems (NIDS). With most of the Internet's traffic being encrypted, detecting harmful activities becomes more and more challenging. Deep Learning has shown exceptional results in many domains, but was not able to significantly improve NIDS performance over traditional Machine Learning ensembles. We propose new image based representations of network traffic that can utilize the powerful pattern identification performance of Convolutional Neural Networks (CNN). State of the art synthetic intrusion detection datasets and real world captures are used in combination with modern Deep Learning techniques like Siamese Networks (SN) and Few-Shot learning to investigate intrusion detection performance in binary and multiclass classification tasks. To address modern day traffic, we limit ourself to only use network traffic features which are also available in highly encrypted environments. A multikey approach is used to capture as much information as possible in this limited feature space. We show that different models and architectures have only little influence on the detection performance. Using multikey based visual representations, we outperform state of the art CNN-representations, yet equalizing ensembles that also work on multikey-based vectors. From our experiments we can conclude that quality and granularity of data is more important than its form of representation or the applied classification algorithm. As a result we present an optimized way to represent network traffic for deep learning, provide an end-to-end approach for generating datasets and evaluating models, and present multiple performance optimizations for Siamese Networks.

Contents

CHAPTER 1

Introduction

The introduction chapter is the starting point of this thesis. In the background section, the scientific and technical context covered by this thesis is explained and an overview of applied methods and approaches is given to put the reader into context. A series of opinions and observations is given in the motivations sections where we also elaborate on what the main points of interest are and why we consider them as important. A more detailed collection of methods, approaches, expected outcome and research questions is given in the methodologies and goals section. Finally, the structure section shows an outline of the remaining thesis with a brief description for the rest of the chapters.

1.1 Background

According to the 2021 ISRG Annual Report [32] 92% of web page loads in the U.S. are encrypted and 83% worldwide. While the recent focus on data privacy is to be favoured, this trend imposes new challenges in the domain of network security and traffic analysis.

Network Traffic Analysis (NTA) is the process of monitoring, storing and analysing traffic in networks. It has many applications including performance optimization, maintaining quality of service, and security and privacy [101]. Encryption makes it harder to inspect and analyse this traffic. Typical methods like port inspection and Deep Packet Inspection (DPI) can not be used with encrypted traffic [16][51]. New methods and approaches have to be developed to guarantee the safety, stability and security of networks and data.

With the recent popularity of Machine Learning (ML) and Deep Learning (DL) a lot of research focused on applying these techniques to NTA and Intrusion Detection (ID) [89]. Deep learning and especially Convolutional Neural Networks (CNNs) show outstanding performance in pattern recognition and are applied in many domains [50][102][33]. In NTA and ID one of the main problems is to create data representations from network traffic which can be used to train such systems.

First approaches used feature-vectors to train regular Neural Networks (NNs) [24][89]. Later several ways of converting these feature-vectors to Images were introduced [80][25]. The idea is to use CNNs to automatically detect important features and patterns on its own without creating a fixed size feature-vector.

1.2 Motivation

Anomaly detection is a widely covered topic and a variety of different algorithms and methods are already investigated [89]. In my opinion, many of these researches impose an unnecessary limitation on themselves by focusing on the traditional 5-tuple key (source IP, source Port, destination IP, destination Port, protocol identifier) to capture network-flows [18]. This can results in systems not being usable in certain scenarios because higher levels of encryption are used (e.g. IPSec) or systems not utilizing their full potential because techniques like combining different flows are not considered [19][60].

Another area where I see a self imposed limitation is feature representation. Many researches use the same feature-vectors for training DL models as they do for traditional ML approaches. A powerful characteristic of CNNs is the ability to detect patterns that humans and other ML algorithms can not see [50]. Using self defined feature-vectors artificially limits the models to only learn from what we think is important. Instead we should keep the representations as true to the original data as possible and let the CNNs detect the features. Different researches have shown that images created from flows dont lose information with respect to the feature vector they are created from and maintain all or most characteristics of that flow [80][25].

When it comes to NTA data for the detection of attacks, most of the time the datasets are heavily imbalanced and skewed towards the normal class [58]. Often this is compensated with data-level approaches like resampling strategies or artificial data generation [20][43]. In my opinion creating authentic synthetic attack data is very hard and algorithm-level approaches to handle the class imbalance problem might be the better solution. Therefore, we investigate the use of Siamese Neural Networks which are know to show good performance on imbalanced datasets by using distance metric learning [98][37][85][20][21].

Because we think it is important to have an IDS which is fast and easy to maintain we investigate multiple optimization strategies like online pair mining and few-shot learning to reduce training time and need of computational resources of the proposed DL models [35][84].

We want to provide insights in the areas of siamese networks, optimization, and less traditional feature representations like images, using approaches that have, to the best of our knowledge, never been tried before.

1.3 Methodology and Goals

In this thesis we aim to improve on existing image based feature representations, under consideration of modern encryption techniques, by using the strong pattern detection performance of CNNs and siamese networks. We aim for an end-to-end process covering flow extraction from network captures, creating different feature representations and evaluating different DL architectures.

The main goals of this thesis are as follows:

- propose novel 2D feature representations of network traffic flows that improve CNN accuracy over current state of the art image representations
- find which DL architecture is more suitable for handling network traffic data, CNNs or siamese networks
- investigate if approaches are consistent in terms of performance metrics, regardless if synthetic or real-world data is used. Approaches should work independently of the dataset with performance metrics following a similar trend across datasets.

In order to achieve these goals the following methodology is used:

- we create 4 novel feature representations (TUWPic-*) and compare them against a state of the art image representation called FlowPic [80]:

  1. TUWpicA
  2. TUWpicB
  3. TUWpic-mkA
  4. TUWpic-mkB
  5. FlowPic

  , where A and B refer to different methods for generating the feature representations and mk refers to *multi-key*.

- we provide a CNN architecture with adequate performance by comparing different architectures from literature and other established sources and run preliminary test to choose the best architecture for more in depth experiments
- we investigate the use of algorithm-level approaches like siamese networks, few-shot learning and online-pair mining, to handle imbalanced data
- we provide a baseline model using a traditional Random Forest ML algorithm and feature representations to compare the results and put them into a meanignful context

- we create a system that can be used to train and evaluate different combinations of architectures, datasets and other configurable parameters
- we give a comprehensive quantitative comparison of the different feature-representations and architectures proposed in this thesis

The outcome of the experiments is used to answer the following research questions:

- How does the novel feature representation TUWpic compare to state of the art 2D-flow representations with regard to standard ML performance metrics?
- Can siamese networks be used to improve performance in comparison to traditional DL models and therefore offer an algorithmic approach of dealing with imbalanced data? Do siamese networks offer any advantage over CNNs? Which architecture is best suited for NTA?
- Can few-shot learning be used to lower training time and resource demands while maintaining the same level of performance as full training?
- Can synthetic datasets be used for selecting ML/deep-learning approaches and feature representations without having to change the architecture or parameters when the same process is applied to real world data?

1.4 Structure

The remainder of the thesis is structured as follows. In Chapter 2 the foundations and background knowledge about Network Traffic Analysis, Intrusion Detection and Deep Learning are presented. Then in Chapter 3 we start by introducing the used datasets. After that the feature extraction, pre-processing and transformation methods are explained. At last we describe the used algorithms and performance metrics as well as give an overview of the experimental setup. In Chapter 4 the outcome of the previously presented experiments is shown. The results are analyzed and discussed in relation to the above defined research questions. In Chapter 5 the gained knowledge is summarized and possible improvements and future work is presented.

CHAPTER 2

Background and Related Work

In this chapter we present the needed background knowledge and cover related work relevant to this thesis. First an overview of Network Traffic Analysis (NTA) and Intrusion Detection (ID) is given. Objectives, different types of NTA, techniques, methods and challenges are introduced. Further, an overview of image based NTA is given as well as an overview of related research dealing with Machine Learning (ML) and Deep Learning (DL) for ID. Finally we introduce the fundamental concepts of DL needed to understand the applied methods and results of this thesis. Specifically Convolutional Neural Networks (CNN), Siamese Neural Networks and online pair mining is covered in detail.

2.1 Network Traffic Analysis

Network Traffic Analysis (NTA) covers a wide range of topics each with different objectives, applications and techniques [28]. The core idea is to monitor traffic at an appropriate granularity so that meaningful information and characteristics can be extracted [101]. With the growing number of networked devices, NTA techniques are essential to maintain stability and availability of communication systems. Abbasi et al. [16] outline a general framework for NTA tasks as can be seen in Figure 2.1. Each task starts with the definition of a goal or an objective, some of the most popular domains and use-cases are as follows:

- ISPs apply NTA techniques in resource planning and traffic routing (e.g. assigning higher priority to time critical packets) to guarantee a base level of Quality of Service (QoS) to an end user [67]

- NTA is used by network administrators to monitor network performance, detect bottlenecks and effectively assign resources to network entities

- in network security, NTA is used for Intrusion Detection (ID). A variety of approaches and techniques exist that focus on detecting and preventing network attacks [16].

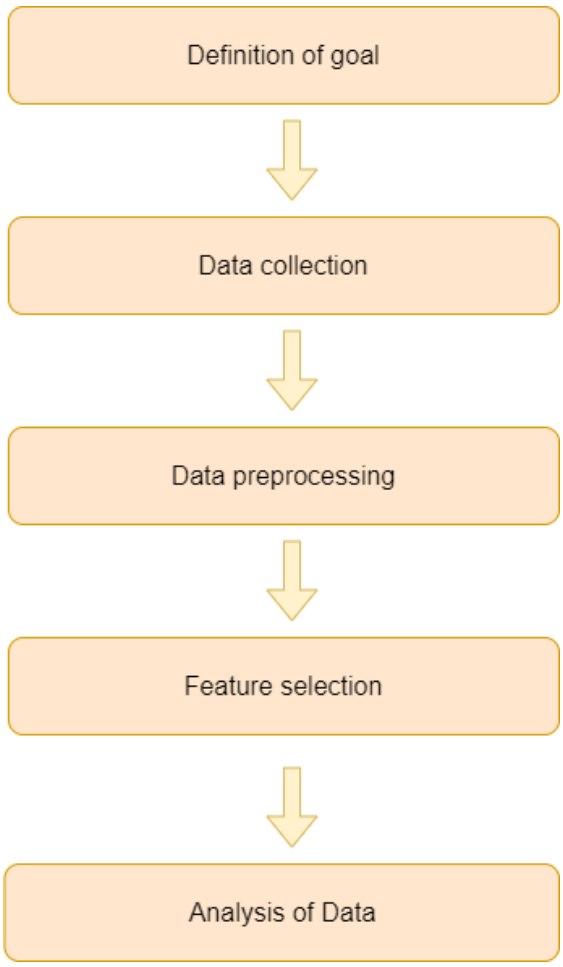


Figure 2.1: NTA framework structure based on [16]

2.1.1 Intrusion, threat and attack detection

Network intrusion detection systems continuously monitor traffic in a network or part of it for malicious traffic and take an important roll in attack prevention [93]. IDS differ from traditional firewalls by applying dynamic detection methods, while firewalls usually only apply static rule based methods [87]. IDS can be categorized into signature-based detection systems or anomaly-based detection systems. Further, they are differentiated by their deployment and operation [20]. Depending on if they are intended to protect a host or a network, we find:

Network based intrusion detection systems (NIDS) monitor all incoming and outgoing traffic of any number of hosts. NIDS can be adjusted for each host allowing for fine grained control of what to monitor [90]. NIDS must be capable of monitoring a large amount of traffic in a time efficient manner and therefore are often resource demanding.

Host based intrusion detection systems (HIDS) monitor the incoming and outgoing traffic of a single host. HIDS only have limited knowledge of the network topology in which they are deployed [46]. Some HIDS can exchange information when deployed in the same network. HIDS cannot detect attacks on other hosts in the network. Because of its local characteristics HIDS have less traffic to monitor than for example NIDS and therefore are less resources demanding.

In order for IDS to detect attacks, different approaches are used:

Signature based detection, sometimes referred to as misuse detection in literature, is a very effective method of detecting and preventing well known attacks [63]. It is a static method which uses predefined patterns of know attacks and can very quickly detect these patterns in a network [87]. Signature based systems work similar to anti-virus programs which also use signatures of malicious software for detection. Signature based systems must maintain a database of well know attacks and their corresponding patterns which results in the need for continuous updates to identify new attacks [90]. The development of these patterns needs human expertise, data and time. Thus making signature based systems vulnerable to zero-day attacks and preventing them from detecting unknown attacks[46]. Snort is an example for a signature based IDS which can be deployed on NIDS or HIDS level. Its detection engine uses rules and signatures to find suspicious packets in traffic. Its architecture and rules were later used in a more advanced IDS called Suricata which is multi-threaded and comes with many additional features not present in Snort [70].

Anomaly detection is mainly used for unknown or novel attack detection using statistics, supervised and unsupervised ML techniques. It is a dynamic method that continuously monitors the network. It works by generating a network profile of "normal use" [52]. If there is a significant difference between the generated normal profile and the current behavior of the network the system will raise an alarm [57]. Anomaly detection based systems are prone to have high false positive rates since they are unable to see the difference between a new user and an attack. However, they are needed to detect novel and zero-day attacks. An ongoing topic of research is to define and extract good features

that can be used to define such a baseline model. Supervised ML and DL models are the most recent development in IDS. Supervised ML and DL models are trained to detect attacks or create a model of normality [87]. This is the approach we focus on in this thesis. While statistics and unsupervised approaches can also rely on well-know behaviors and patterns, supervised techniques need labeled ground truth data which imposes a challenge on its own. Labeled IDS datasets are often hard to get, outdated, incomplete and heavily imbalanced [58]. While DL shows promising results in many domains, an evaluation of different ML and DL models by Thapa et al. [87] showed that in the domain of ID, DL models don't outperform ML models.

Different approaches use different information, we can categorize them into 4 main groups:

Port-based approaches use well know ports of applications to classify traffic. They extract the port numbers from TCP and UDP packet header and compare them against a registered list of ports from the Internet Assigned Number Authority (IANA)[34][3]. This was a very effective and simple solution in the early days of ID. Methods like dynamic port assignment and the reuse of well know ports for multiple applications and use-cases make this method obsolete nowadays [91].

Payload-based approaches make use of DPI [34]. Information specific to attacks or malicious traffic is extracted from TCP and UDP packet payloads and headers. The system has a database of attacks and checks each packet for attack patterns. Often the terms signature-based and payload-based are used interchangeably however not all signature based IDS make use of DPI. Payload-based methods tend to be very precise but lack the possibility to inspect encrypted traffic[91]. With an ever increasing focus on privacy and encryption this method becomes more and more obsolete. Most modern approaches use statistics or flow based methods.

Flow-based approaches extract statistical features from network flows [91]. A network flow is a series of packets that share common features in a given time-frame. The features which are used to define a flow are called flow keys. Some commonly used flow keys are source and destination IP, source and destination port and the protocol identifier[18]. Depending on the number of flow keys a network flow captures different behaviors. A 5-tuple key captures flows on application level, a 2-tuple key ("source IP","destination IP") captures flows on host level (host to host conversation) and a 1-tuple key (source IP or destination IP) captures information of a specific endpoint [60]. These flow-based approaches mostly use some form of supervised or unsupervised ML or DL for anomaly detection [34][91].

Aggregated statistics are often used in time-series based approaches. Statistics of packets are aggregated over time resulting in a time-series. These time-series can then be used in a model or rule based system to check if its pattern corresponds to normal behavior or differs at certain points [47].

2.1.2 Challenges

Regardless of the approach or type of information that is used in an IDS, there are several common challenges that are fundamental to NTA and need to be addressed.

Often the problem source is the data or the lack of it. As addressed by Sharafaldin et al. data in the realm of NTA often is outdated, is missing explanations, has many duplicate entries, is generated in unrealistic scenarios and suffers from information loss because of privacy regualations [82][81]. To overcome these shortages, they created the IDS2017 dataset [2], which we also use in this thesis.

Related to the problem of data generation and collection, there is the problem of missing labels or ground truth [82]. Some dataset come with inconsistent or no labeling at all. In these cases the exact details of the attack like time, date, attacking hosts and ports, victim hosts and ports and type of attack have to be reconstructed. That is why ID dataset are often created in a simulated environment where a regular network is emulated and the attack parameters can be controlled precisely [91]. Both datasets used in this research come with labels and we do not have to use additional tools or methods to create them ourselves.

A problem that is especially relevant in ML based approaches is feature selection. Not all features have the same importance and some might even degrade performance of models by introducing noise. Other features might be highly correlated to each other and yield no additional information. Feature selection can help to mitigate these problems and improve detection rate and lower computational difficulty. Choosing the right features often requires human expertise and a lot of time and can be a research topic on its own as discussed by Khraisat et al. [47]. In this thesis we rely on already established and tested feature-vectors [60].

While all challenges are equally important, in this thesis we mainly focus on the problem of imbalanced datasets and feature representations since they are inherently connected to the research topic. Additionally we focus on encrypted network traffic, which impacts the available features and flows.

Feature representations are heavily linked to the methodological approach used in an IDS. In one way or another the recorded traffic has to be transformed into a representation that can be used by an IDS [76]. ML and DL based approaches often need data in numerical format, some models might be able to deal with time-series data while others need to aggregate that kind of data into a single value. The goal is to maintain as much of the original information of the network data as possible when transforming it into a representation suitable for and IDS. In this thesis we use two different feature representations, namely feature-vectors and images. Both are created from network flows. A more detailed overview of different feature representations is given in Chapter 2.1.3.

Due to the nature of network traffic, ID datasets are often imbalanced in a way that they are skewed towards benign (non-malicious) traffic [58] . Benign traffic often makes up over 90% of such datasets. Depending on the size of the network there can be

thousands of web-page loads, VoIP calls, video streams and many other network based actions. Malicious traffic only accounts for a small fraction in this network. To overcome the problem of imbalanced data there are several methods to mitigate the impact of it. Data-level techniques manipulate the dataset by increasing or decreasing samples from specific classes. Algorithm-level techniques aim to implement algorithms that can handle imbalanced data without previous manipulation of the dataset [20]. Dealing with imbalanced data is a common problem in many domains and therefore a well studied subject. Following are some common methods and approaches that can be used to handle imbalanced data:

- A common data-level approach is resampling. Several resampling strategies exist. [85]. The most common methods are oversampling, undersampling and Synthetic Minority Oversampling Technique (SMOTE). Oversampling increase the number of samples in minority classes by copying existing samples. Undersampling reduces the number of samples in the majority classes by randomly eliminating samples. SMOTE uses characteristic of the minority classes to create new artificial samples of it. The main drawbacks of these re-sampling methods are as follows [20]:
  - Oversampling can lead to over-fitting of minority classes.
  - Undersampling may lead to loss of information in the majority classes.
  - SMOTE may introduce noise, cover invalid information and generate overlapping samples between classes.
- Recently DL based methods are gaining popularity for handling imbalanced data. Iliyasu et al. [43] use a Deep Convolutional Generative Adversarial Network (DCGAN) to create new samples from random noise. GANs consist of a generator NN which creates samples from random noise. A second NN called discriminator then takes samples from both the generator and the real dataset and determines if its real or fake. The loss functions maximizes the probability for the generator to create a sample which is classified as real by the discriminator. Thus the generator is approaching the underlying distribution of the training data.
- Lee et al. [55] use a GAN to create new samples from the CICIDS-2017 dataset [2] as depicted in Fig. 2.2. The same RF model trained with GAN outperforms the non GAN RF model. It also outperforms a RF model trained with a SMOTE manipulated dataset. They later applied the same method to other datasets and achieved similar results [56].
- A Supervised Variational Auto-Encoder with Regularization (SAVAER-DNN) is proposed by Yang et al. [100] to create new samples from the underlying distribution. The system is tested on various datasets and shows improvement in detecting lower frequency classes.
- Another DL based approach is using Siamese Neural Networks (S-NN). Siamese Networks can handle datasets with only a few samples per class since it creates

similarity based distance embeddings for each sample rather than predicting class probabilities [98][48]. The approach we use in this thesis is also based on Siamese Neural Networks. Several studies have tested the effectiveness of Siamese networks to address the problem of imbalanced data and have shown optimistic results [85][20]. Siamese networks are explained in detail in Chapter 2.2.3.

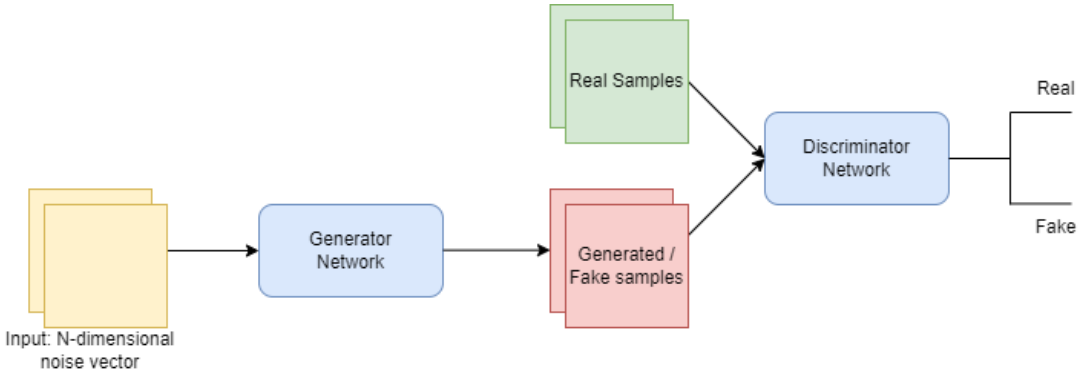


Figure 2.2: GAN used to generate new samples based on [56].

Encryption plays a vital role in modern day network communication. Most modern browser warn the user if they visit a website without encryption. In web, the defacto standard for encryption is SSL (Secure Sockets Layer) and its successor TLS (Transport Layer Security). But also other means of encryption become more and more popular. VPNs are used in a wide range of scenarios like:

- accesing work computers and networks from away
- circumventing geoblocking for specific contents on the internet
- reducing ones digital fingerprint on the internet

Most VPNs either use TLS or IPSec as encryption protocol. TLS is implemented on transport layer level while IPSec offers network layer encryption. This allows IPSec to operate either in transport mode, where it offers similar level of encryption as TLS, or in tunneling mode, where also the original IP header is protected [49]. In this thesis we consider a high level of encryption either using TLS or IPSec (transport mode) which limits the features that can be extracted from captured network traffic to: IP addresses and packet statistics like inter-arrival time and lengths.

2.1.3 Feature Representations

Feature selection, feature engineering and transformation is key to successfully train ML and DL models for anomlay detection like we do in this thesis. Network traffic is usually captured in PCAP files. A PCAP file contains all incoming and outgoing packets of the host where it is deployed [76]. Every entry contains meta information of a packet such as timestamp, protocol, protocol identifier,ip addresses and ports. Using these features, network flows can be extracted [60]. Depending on the granularity, different types of behaviors can be captured in network flows as described in Chapter 2.1.1.

Salman et al. [76] groups feature representations in the following categories:

- **behavioral-based** representations are used to built rule-based profiles and policies as shown by Hu et al. [38]. It works similar to vector-based approaches with the main difference being that feature-vectors are used as input to other processes and behavior-based approaches simply work on the extracted flow and packet statistics.
- **vectors** are on of the most common forms of feature representations in NTA. They are typically generated from network flows and provide statistical information about incoming packages such as min, max, standard deviation, mean, etc. of flow packet attributes. A 249 long feature vector was proposed by Moore et al. which is still used today and is also one of the feature representations used in this thesis [62].
- **time series** are created by continuously monitoring and updating statistics of incoming packets. One such approach was proposed by Conti et al. [27] where 4 time series have been created (1. outgoing bytes, 2. incoming bytes, 3. incoming and outgoing bytes, 4. inter-arrival times) from network flows.
- **word embeddings** are most commonly used in web traffic classification. Strings can be extracted from HTTP requests, URLs, DNS queries and packet payloads (DPI). They can be further processed by techniques from Natural Language Processing (NLP).[76]
- **images** are the latest addition to typical feature representations for network traffic and also the main feature representation used in this thesis. With CNNs achieving very good results in many computer vision applications multiple image representations have been proposed [34][43][97]. We give a detailed overview of image based feature representations in Chapter 2.1.4 and Chapter 3.

As mentioned before, in this thesis we focus on the extraction of network flows from encrypted traffic and the creation of feature-vectors and images. The applied methods and approaches are covered in detail in Chapter 3.

2.1.4 Anomaly Detection Methods

Methods for anomaly detection cover a broad spectrum of algorithms, techniques and approaches [45][71]. At the highest level they can be separated into supervised and

unsupervised approaches. Unsupervised approaches like clustering and outlier-detection are often used because of missing labeled data which is a key challenge in NTA as we have discussed before. Unsupervised methods have drawbacks in terms of validation and evaluation and supervised methods are preffered if the data allows it.

Key focus of most suggested approaches and methods is improvement of detection rate or reduction of complexity and the resulting gain in processing speed and ease of use. Kwon et al. [52] have comprised a detailed overview of different methods used throughout the time as can be seen in Fig. 2.3. Iglesias et al. [40] analyzed various papers and created a list of proposed methods with most promising performance:

- **Rule induction, decision trees, and random forests** are robust methods but the performance is affected by class imbalance
- **Neural networks and support vector machines** often yield very good results but come with some drawbacks like high preprocessing complexity, computational costs and lack of explainability (black boxes)
- **Probabilistic and Bayesian methods** are fast and simple and can be applied to a variety of data, however these methods assume that features are independent which is often not the case in network traffic data
- **Clustering** often is used as part of an analysis framework or as transformation (dimensionality reduction) method

In the following we present examples of different approaches for anomaly and attack detection. This should help the reader to be aware of existing approaches and help him/her put this thesis into context. While most of these approaches are also suitable for attack detection, there can be different goals in NTA. Some approaches try to classify specific applications based on their traffic like described by Taylor et al. [86], while others focus on specific functions within an app [27]. Others only differentiate between malicious and non malicious traffic [69].

Taylor et al. apply [86] random forest to identify Android apps based on app fingerprints generated from statistical flow features.

A rather exotic method is presented by Ali et al. [17], where Particle Swarm Optimization (PSO) is combined with Fast Learning Network (FLN). However, PSO needs human expertise to define an objective function for the model which might lead to undetected zero-day attacks.

Support Vector machines (SVM) and Principal Component Analysis (PCA) are used in [30] to for faster classification with reduced input dimensions. Using the NSL-KDD dataset [8], they showed that their approach was faster and achieved better accuracy compared to using SVM with the original 41 features.

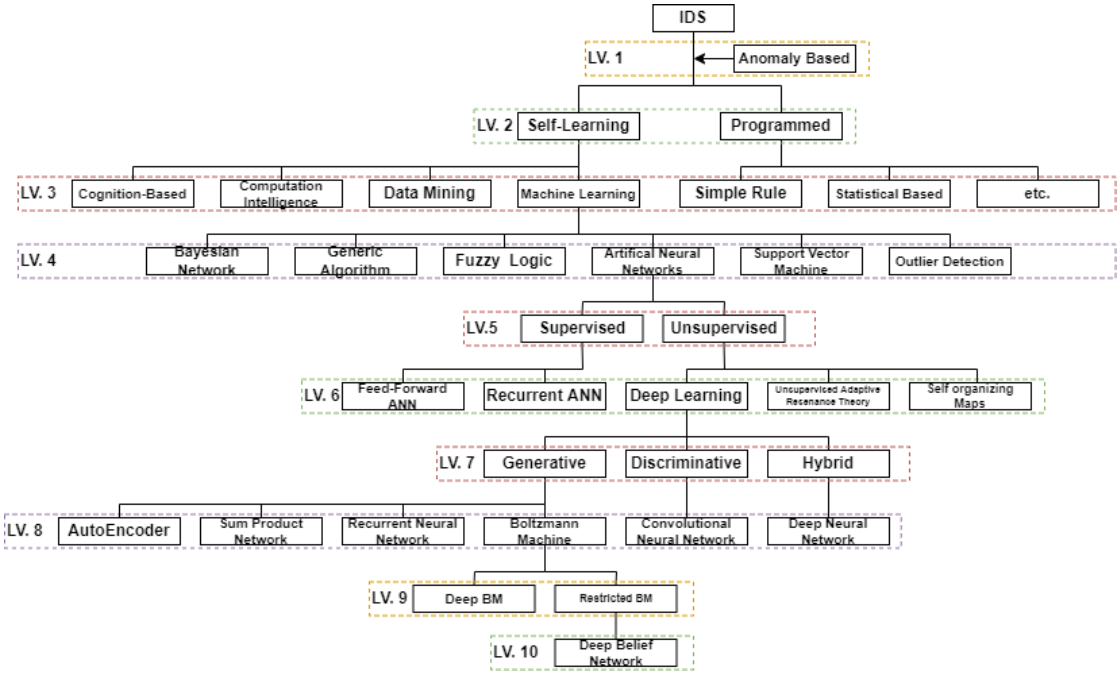


Figure 2.3: Intrusion detection techniques based on [52].

A semi-supervised NT classification method is proposed in Rezaei et al. [72]. First a CNN is trained with unlabeled data, then the learned weights are transferred to a larger CNN which is re-trained with a smaller labeled dataset. They showed that the unsupervised pre-training step significantly increased the performance of the CNN without increasing the number of labeled samples.

In the next section we present image based intrusion detection approaches using CNNs.

Image Based Intrusion Detection

In 2011 Nataraj et al. [65] created so called Malware Images from executables. They converted malware infected binary files into feature vectors of 8 bit unsigned integers. Each integer represented a greyscale pixel in the resulting image. These images had fixed width and variable height depending on the binary file size. They extracted information from the images using image processing methods and applied a k-nearest neighbor classifier for the classification task. Even if this method does not find use in the realm of ID, it shows that the idea of creating images from malicious data has been around for some time. With the growing popularity of DL, image based methods for ID were developed.

Chen et al. [25] were one of the first to use image based feature representations for ID. Their proposed feature representation Seq2Img is an image representation of network traffic that uses Reproducing Kernel Hilbert Space (RKHS) embeddings. Packet length,

packet inter arrial times and packet directions are mapped from their original space into the RKHS space. The resulting 6 channel image approximately covers the probability distribution of the underlying flow. A simple CNN model was used to classify the images. For performance comparison, they extracted statistical features from the flows and traind other ML models with them.

At the same time Wang et al. [94] proposed a different image based feature representation. Flows are extracted from network captures and stored as single files (either PCAP or BIN depending on which type of flow was extracted). Each flow is limited to a maximum size of 784 bytes (zero padding is used if a flow is to short). They apply the same method as described in Nataraj et al. [65] to convert these files into images. Because they use a fixed file size they can generate same size (28x28) images which are needed in order to train their CNN model. They used a LeNet-5 bsed CNN architecture for their model. An almost identical method is later investigated by He et al. [34].

In Zhou et al. [103] the feature vector proposed by Moore et al. [62] is extracted from network flows and then converted into a 16×16 pixel matrix. Each of the 249 features contained in the feature vector is normalized. The normalized faetures are then filled into the pixel matrix. The remaining pixels are zero padded. They then feed these images into various LeNet-5 based CNN models for classification.

Shapira et al. [80] proposed a representation called FlowPic. It is a 2D payload-size distribution histogram. Images of 1500x1500 pixels are created by plotting packet size against scaled arrival time. They also use a LeNet-5 related CNN model for classification. We use FlowPic as a comparison for the proposed feature representation in this thesis and describe the algorithm to create FlowPics in detail in Chapter 3.2.3.

While above discussed methods all directly transform flow data to image data, Xiao et al. [97] propose a different method. They first apply dimensionality reduction to reduce noise and correlation from the flows and create images based on the resulting data. An auto-encoder is used to generate the reduced feature set. An auto-encoder consist of an input layer I, an output layer O and a hidden layer H. The goal of the auto-encoder is to achieve $L \approx O$ going through the hidden layer H. The number of neurons n in the hidden layer is choosen to satisfy $n = m * m$ where $m * m$ is target dimension of the images to be generated. They then apply a CNN to do the classification task.

To overcome the problem of skewed datasets Iliyasu et al. [43] proposed a sampling technique based on Generative Adversarial Networks (GAN). GANs train a NN model to trick a classifier (discriminator) into thinking that the original sample and the sample created by a generator network are the same. Thus the generator network learns the underlying distribution of the original dataset. New samples are then generated by showing original samples to the GAN and taking the ouput from the generator model. They also propose a new feature representation based on random sampling of flow packets. They define a feature vector for each packet (packet length, packet inter-arrival-time, packet direction) and randomly take 20 packets from each flow. These packets are then transformed into images of size 20x3 with the feature values being scaled to the range of

-1 to 1.

A different appraoch is taken by Khan et al. [46]. They extract 78 features per flow, normalize and one-hot encode them and then apply Short-Time Fourier Transform (STFT) to it. Based on the resulting STFT data they create histograms using a custom algorithm. The result are images of size 28x28x3 that are then used to train a CNN model.

The literature discussed above shows the most used approaches and feature representations. Other research mostly tries to optimize named approaches or tries to combine feature representations of one research with the classification approach of another research [33][59]. They might also vary in their goal, some are trying to classify types of traffic while others focus on security. But hardly any research introduces new feature representations or classification approaches.

In this thesis we use a traditional ML algorithm, namely random forest, as a baseline model since it shows consistet results throughout the time and it is used across the field. The performance of this baseline model is used as reference for evaluating DL based approaches. The next section introduces the concepts of deep learning and CNNs which is the base for the approach proposed in this thesis.

2.2 Deep learning

Deep Learning powers many modern applications and systems. It is applied in search result optimization, e-commerce recommendation and speech recognition to just name a few examples [53]. In our case, we want to use the pattern detection power of CNNs to help identify malicious network traffic. The benefit of DL in contrast to regular ML algorithms is the possibility to do representation learning. Representation learning allows an algorithm to detect and extract important features on its own, without providing additional information or context. Traditional ML algorithms are usually trained with data that was extracted with complex rules and pre-processing steps [53] while Neural Networks can learn complex non-linear function mappings without any prior feature engineering steps [79]. ML and DL is categorized in supervised and unsupervised learning:

- Supervised learning uses labeled data to train models. A model can output one or multiple scores depending on the task. An objective or loss function is used to compute the error or distance between the models output and the labeled data. The model then adjusts itself in a way so that the loss is minimized [53].
- Unsupervised learning algorithms use data without assigned labels. The labels or classes of the data is discovered by the unsupervised learning algorithm. Clustering is an example for an unsupervised algorithm [68].

DL has to be used with some considerations. When compared to other ML algorithms, DL often needs more data, time and computational resources. One way to increase

performance and reduce training time is to use GPUs which implement fast and efficient matrix calculations which favors NN architectures [96]. Another problem is explainability. While ML algorithms like decision trees and random forests are more or less interpretable, DL models are so called black-boxes. We do not know how and why outputs are generated the way they are [77]. This is especially a problem in fields like medicine where the outcome might decide about a certain drug to use or treatment to apply [88]. If we can not explain why the medication is suggested, is it reasonable to use it?

Deep Learning is a special case of Machine Learning. Deep refers to the stacking of many small binary classifiers, called perceptrons, on top of each other to create a multi-layer interconnected network of machine learning units. Artificial Neural Network (ANN) are models based on this layered design. By stacking many of these layers on top of each other, a deep ANN can be created. These are the foundation for more complex models like Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs).

2.2.1 Artificial Neural Network

An Artificial Neural Network (ANN) is made up of simple functional units that are connected to each other [53]. These units are called neurons and can be grouped into layers. Each of these layers can consist of hundreds of neurons. ANNs consist of stacks of layers where neighboring layers are connected to each other. Each neuron is connected to a subset (or all) neurons of the previous layer. This allows for data to be passed forth and back in the network [79]. The output of a neuron is calculated by the weighted sum of its inputs plus a bias and passed through an activation function [53]. Using the backpropagation algorithm, the weights are adjusted to minimize the error of the objective function [79]. This way the ANN will approach a state where the network approximately learns the black box function that can map the inputs to its corresponding outputs.

The perceptron learning algorithm in Fig. 2.4 shows how the output of a neuron is calculated inside an ANN. The function is defined in Equation 2.1 where X_{i+1} is the output of the neuron, f_i it the non-linear activation function, W_i and X_i are the weights and inputs (vectors) from the previous layer and b_i is the bias.

$$X_{i+1} = f_i(W_i X_i + b_i) \tag{2.1}$$

Using backpropagation and gradient decent [39][22] the parameters of each neuron can be updated as defined in Equations 2.2 and 2.3 where W_{new} and b_{new} are the updated parameters, λ is the learning rate and E_i is the loss function.

$$W_{new} = W - \lambda \times \frac{\delta E}{\delta W} \tag{2.2}$$

$$b_{new} = b - \lambda \times \frac{\delta E}{\delta b} \tag{2.3}$$

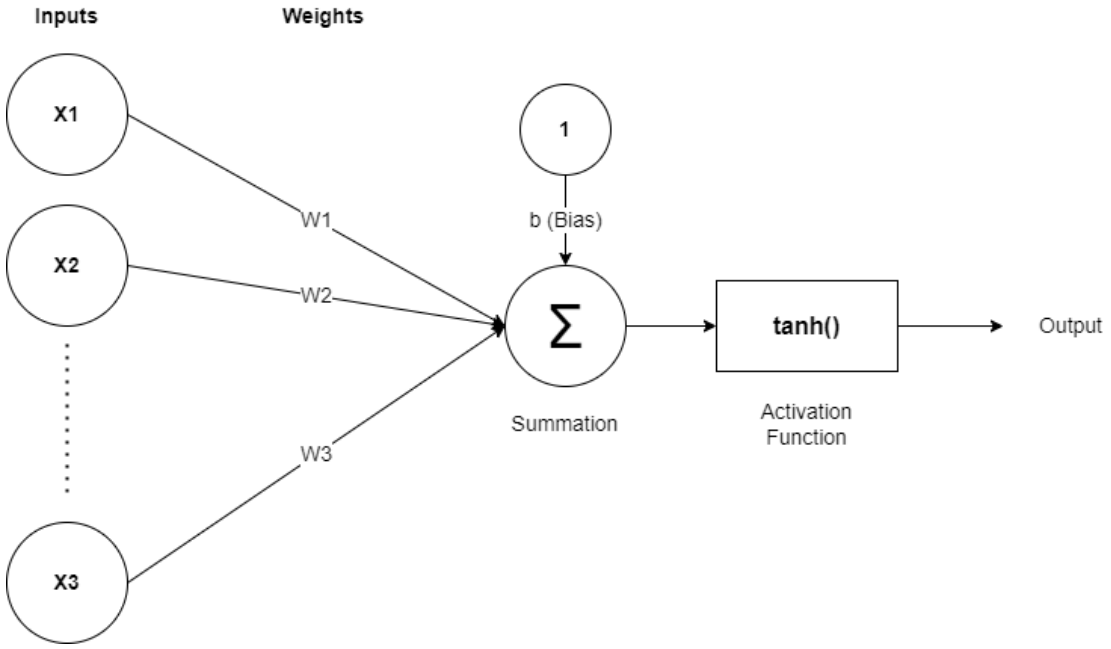


Figure 2.4: Perceptron algorithm based on [79].

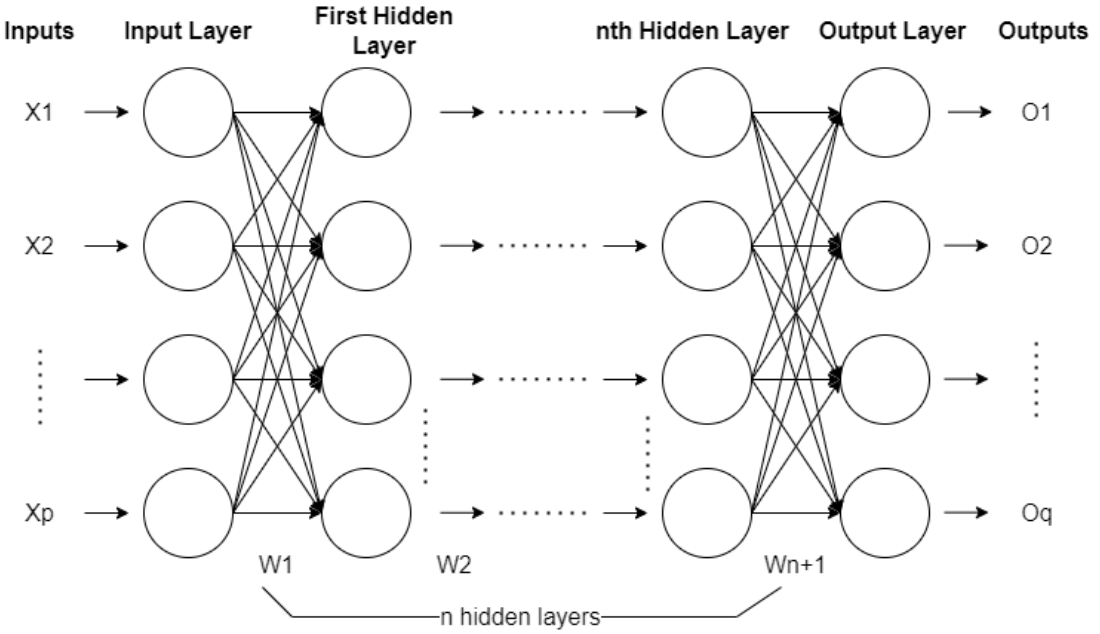


Figure 2.5: Feed forward deep neural network based on [52].

Gradient descent is an optimization algorithm for finding the minimum of a function. Since the goal of an ANN is to minimize the loss function, gradient decent can be applied

to find optimal parameters to minimize that function [68]. A gradient is defined as the vector of partial derivatives of a function with respect to its input variables. It can be seen as a vector pointing in the steepest direction from the current position. By iteratively moving along this direction the gradient will approximate 0 and we will have reached a minimum. This minimum does not have to be the global minimum of the function but can also be a local minimum. A parameter named learning rate is introduced to control the step size with which we move along the gradient. In 2.2 and 2.3 λ is the learning rate which is multiplied by the gradient in respect to a variable. A generic definition can be formulated as:

$$\theta_{new} = \theta - \lambda \times \frac{\delta E}{\delta \theta} \tag{2.4}$$

This gradient is applied to each adjustable weight and bias in the network. To efficiently calculate the gradient for each weight the backpropagation algorithm is used. Starting at the output layer of the network the error of the loss function is calculated. This error is then send back through the network so that each neuron can adjust its weights [53]. The backpropagation algorithm can be recursively defined by the following equations. The gradient in respect to weight $w_{j,k}$ is:

$$\frac{\delta E}{\delta w_{j,k}^{l}} = \theta_j^l \times y_k^{l-1} \tag{2.5}$$

with

$$\theta_j^l = \frac{\delta E}{\delta y_j^l} \times f_l'(z_j^l) \tag{2.6}$$

if l is the output layer, else

$$(\sum_{i=1}^{q} \theta_j^{l+1} \times w_{i,j}^{l+1}) \times f_l'(z_j^l) \tag{2.7}$$

where f_l' is the activation function and z_j^l being defined as the activation value of neuron j in layer l:

$$\sum_{i=0}^{k} w_{j,i}^{l} \times y_i^{l-1} \tag{2.8}$$

Where y_j^l is the output of neuron j in layer l and q is the number of units in layer $l+1$. Although backpropagation is an efficient algorithm, too much computing power would be needed to do it for every sample in a dataset [74]. Stochastic Gradient Decent (SGD), and derivations of it, is commonly used to train networks. The dataset is shuffled and grouped into batches of samples. The loss is calculated as an average of all the samples in one batch and then backpropagation is used to adjust the weights with an average error. While this method will not always find the perfect solution, it was shown that it helps to prevent from getting stuck in local minima and decreases convergence time.

Activation functions serve multiple purposes in neural networks. They aid the learning of high order polynomials and keep output values in a certain range [46]. The most

important aspect of activation functions is that they are differentiable and make sure that backpropagation can be used for training [66]. Some common activation functions are :

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.9}$$

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.10}$$

$$ReLU(x) = max(0, x) \tag{2.11}$$

$$softmax(x) = \frac{e^{x_k}}{\sum_{n=1}^{n} e^{x_i}} \tag{2.12}$$

Depending on the type of network, the classification task, and the input data, different activation functions might lead to better results.

While ANNs work really well for numerical and statistical data, they fall short when we want to work with visual data. If an ANN is trained with images it will learn the pixel values at each input neuron. This does not make much sense since objects in images can have different colors, be at different locations and can be recorded at different angels. To overcome this problem Convolutional Neural Networks (CNNs) were developed. They use a series of self-learned filters like edge detection, blur or sharpen to extract features from images before feeding them into an ANN.

2.2.2 Convolutional Neural Networks

In recent years CNNs have shown excellent performance in the field of visual computing and image processing [50][33]. CNNs are often applied to tasks where spatial or time related data is used. However, Zhang et al. [102] showed that CNNs can also be applied to traditional data which they define as data without spatial or time aware components but with statistical correlation. CNNs work especially well for data in the form of multi-dimensional arrays [53][46].

CNNs use convolutional layers and pooling layers to extract features from the input array. A fully connected layer is used to flatten the generated feature maps to a fixed number of neurons [64]. A visual representation of convolutional layers and pooling layers can be seen in Fig. 2.6

Convolutional layers build the foundation of CNNs. A convolutional layer applies a series of filters (convolutional kernels) to its input and creates a feature map for each filter. The convolution is defined as the dot product between the input and the filter resulting in a feature map as output. The output of the convolutional layer is calculated as follows. Assume $n^l 3 \times 3$ filters $W^l \in \mathbb{R}^{3\times3}$ in convolutional layer l with n^{l-1} being the number of filters in the previous layer. The feature map Y^l is defined as:

$$Y^l = f(\sum_{n=1}^{n^l} W^{l,n} \times Y^{l-1} + b^l) \tag{2.13}$$

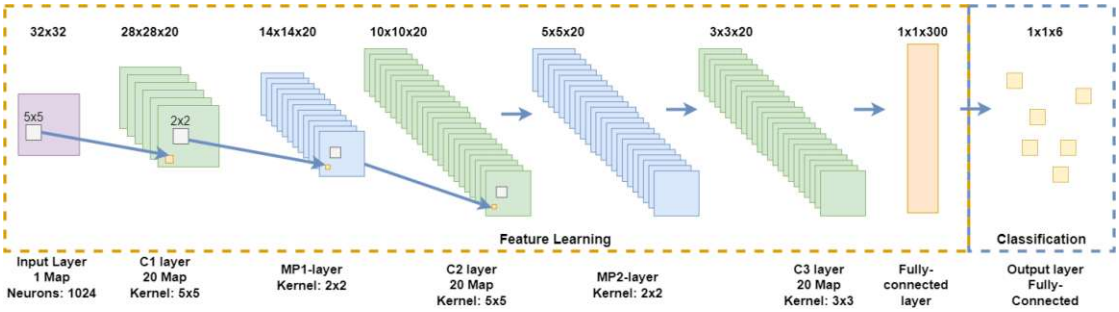


Figure 2.6: CNN architecture based on [64].

where $f()$ is a non-linear activation function [102][53][46].

Pooling layers are used to reduce the amount of features after convolutional layers. Similar to the convolutional layer, the pooling layer is given a size. Assuming a pooling layer of dimension 2×2, each feature map is divided into patches of 4 (2×2) values. The resulting feature maps are down-scaled versions of the original feature maps. Two methods are commonly used as pooling functions. Max-pooling uses the maximum value in each patch. Average-pooling uses the average value of the patch [102][53][46].

The fully connected layer is used as a bridge between the multidimensional feature maps and the one dimensional output layer. After the last pooling layer, the feature maps are flattened to a 1-dimensional vector. This vector is the input of the fully connected layer. [46]

The idea of an CNN is to extract features from the input in a stack of convolutional layers and pooling layers. Then use these features as input for an ANN by flattening them.

Traditional ANNs and CNNs work by mapping similar inputs to the same output class. To achieve this, a lot of different training samples for each class have to be available. Often this is not the case. Also other data related problems like heavily imbalanced datasets can significantly impact the performance of NNs. With the help of Siamese Neural Networks (SNNs) small and imbalanced dataset can also be used to train NNs and use their benefits. Instead of learning class probabilities, vector embeddings are created. A distance function tries to separate embeddings in the vector space by pushing them apart if the they belong to different classes. In this thesis we use SNNs as a way to overcome the data imbalance in traffic data.

2.2.3 Siamese Networks

Siamese Neural Networks were originally proposed as way of signature verification by Bromley et al. [23]. A siamese neural network consist of twin networks that are joined by an energy function (loss function) [48]. Both networks take separate inputs, called pairs, but share the same weights which are adjusted together during training. By sharing the weights, the output of of each network should be very similar [48].

In comparison to regular neural network architectures, siamese networks have no target class in the usual sense. Rather than mapping input vectors to output classes, siamese networks create vector embeddings for each sample. The loss function is utilized to train the network so that similar input pairs are close together in the embedding space and dissimilar pairs are pushed apart [61]. Sometimes this method is called deep metric learning [95].

The first loss function used in siamese networks is called contrastive loss and is defined as:

$$L = \frac{1}{2}lD^2 + \frac{1}{2}(1-l)max(0, m-D)^2 \tag{2.14}$$

where $l = 0$ if input pairs are similar and $l = 1$ if the input pairs are dissimilar, $m > 0$ is a margin term that prevents pairs with very large distances to impact the loss and D is the distance between the emebeddings of the siamese networks.

Typical distance functions include euclidean distance, manhatten distance [61] and cosine similarity [44]. The concept of pairs was extended by Schroff et al. [78], where they proposed a loss function based on triplets that was quickly adopted by many researchers [29][104]. The idea of triplet loss is to add an additional twin network creating a triplet network. Instead of pairs the network is trained with triplets. Each triplet consists of an anchor sample, a positive sample and a negative sample. The goal of triplet loss is to maximize the distance between the anchor and the negative sample and minimize the distance between the anchor and the positive sample. The triplet loss is defined as:

$$L(A, P, N) = max(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha, 0) \tag{2.15}$$

where A is the anchor input, P is the positive input, N is the negative input and α is a margin term to control the influence of the triplet. A visual representation of the triplet loss concept can be seen in 2.7.

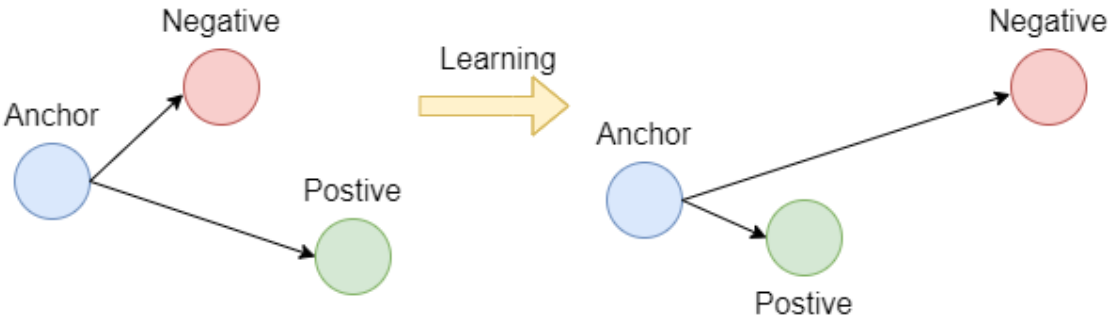


Figure 2.7: Triplet Loss embeddings based on[78].

Because siamese networks learn embeddings and not class mappings, they cannot be directly used for classification. In order to classify samples with siamese networks the embeddings of the training set must be used as input for a classification algorithm like

k-nearest neighbors (KNN) [92]. Then the network can be used to create embeddings for the test data and classify these embeddings with the classification algorithm.

Siamese networks come with their own disadvantages and advantages. One of the benefits introduced by SNNs is few-shot and one-shot learning. SNNs need significantly less training data to create embeddings than regular NNs would need to create class mappings [33][37]. Depending on the scenario, often only a couple of samples per class are needed to successfully train a SNN. This is called few-shot learning. In rare cases this can be even improved to one-shot learning, where only one sample per class is used. Low-end and mobile devices often utilize this technique to energy-efficiently train models.

A problem of siamese networks is sampling of pairs and the volume of training data. According to Wang et al. [95] deep metric learning has to consider all possible combinations of training samples and therefore has to deal with much more training samples in comparison to regular classification tasks. This results in quadratically larger samples for contrastive loss and cubically larger samples for triplet loss. This increase the problem of models having slow convergence and getting stuck in local optima because many samples don't contribute to the loss function.

Multiple sampling strategies have been proposed but generating pairs beforehand (offline) always results in possible loss of information and is a long and tedious data pre-processing step [73]. Pairs or triplets are categorized in hard, moderate and easy triplets. Hermans et al. [35] give a very intuitive description of the problem: "Intuitively, being told over and over again that people with differently colored clothes are different persons does not teach one anything, whereas seeing similarly-looking but different people (hard negatives), or pictures of the same person in wildly different poses (hard positives) dramatically helps understanding the concept of "same person". On the other hand, being shown only the hardest triplets would select outliers in the data unproportionally often and make $f\theta$ unable to learn "normal" associations, [...]"

With the proposed online learning method by Wang et. Al [95] two problems are solved at once. Using this method, samples of pairs and triplets do not have to be generated beforehand but rather are generated as part of the network. Additionally, since the samples are generated inside the network the need for twin or triplet networks is removed because there are no longer pairs and triplets that are used as input. Since then multiple online mining algorithms were proposed [84].

The combination of online learning [95] and a good sampling strategy leads to Online (Semi-)Hard Mining [35][84]. Given a batch of training data consisting of c classes and w images. Both hard and semi-hard mining calculate the pairwise distances between the samples resulting in an cw pairwise distance matrix. For each sample a we can then select the hardest positive and the hardest negative sample within the batch. Because they are only the hardest samples in the batch they can be seen as moderate sample in the complete dataset. If both hardest positive and hardest negative samples are used for training we call it hard loss. If only the hardest negative sample is used we call it semi-hard loss [35][84].

The mathematical definition of these losses is as shown in 2.16 and 2.17 as defined by Hermans et al. [35] and Sikaroudi et al. [84].

$$\mathcal{L}_{\text{BSH}} := \sum_{i=1}^{c} \sum_{a=1}^{w} \sum_{\substack{p=1 \\ p \neq a}}^{w} \Big[m + \mathcal{D}\left(y_a^i, y_p^i\right) - \min_{\substack{j \in \{1,\dots,c\} \setminus \{i\} \\ n \in \{1,\dots,w\}}} \left\{ \mathcal{D}\left(y_a^i, y_n^j\right) \mid \mathcal{D}\left(y_a^i, y_n^j\right) > \mathcal{D}\left(y_a^i, y_p^i\right) \right\} \Big]_+ . \tag{2.16}$$

$$\mathcal{L}_{\text{BH}} := \sum_{i=1}^{c} \sum_{a=1}^{w} \left[m + \max_{p \in \{1,\dots,w\} \setminus \{a\}} \mathcal{D}\left(y_a^i, y_p^i\right) - \min_{\substack{j \in \{1,\dots,c\} \setminus \{i\} \\ n \in \{1,\dots,w\}}} \mathcal{D}\left(y_a^i, y_n^j\right) \right]_+ \tag{2.17}$$

CHAPTER 3

Methodology and Experiments

In this chapter we first introduce the reader to the datasets used in the experiments, then we go over the different feature representations that are used as input for multiple Machine Learning and Deep Learning models, finally we show the experimental setup that was used to conduct the experiments.

3.1 Datasets

In this thesis, we use two datasets that cover synthetic and real-world data. The Intrusion Detection Evaluation Dataset (CIC-IDS2017) [2] is a synthetic dataset created by the Canadian Institute for Cybersecurity [81][83]. The second dataset is collected from real traffic by the Measurement and Analysis on the WIDE Internet (MAWI) Working Group. Both datasets come as raw *PCAP* files which allows for full control of flow definition and feature extraction. The *PCAP* [9] file format contains a timestamp which marks the capture time, the size of the packet and the packet itself. For the rest of the thesis, the first dataset is referred to as IDS2017 and the second dataset is referred to as MAWI. Following, the origin, collection and characteristics of the datasets are described. Statistical information like class distribution are presented in Chapter 3.5.

3.1.1 IDS2017

The IDS2017 dataset is designed in accordance to a paper by Sharafalding et al. [81]. The dataset was created with an emphasis on the following eleven characteristics which to them are critical for a comprehensive IDS dataset namely, *Attack Diversity, Anonymity, Available Protocols, Complete Capture, Complete Interaction, Complete Network Configuration, Complete Traffic, Feature Set, Heterogeneity, Labeling, and Metadata*[83].

The dataset was collected in a lab environment over the course of a week. The following protocols are captured in the dataset HTTP, HTTPS, FTP, SSH and email. The research

group simulated attacks of the following categories *Web based, Brute force, DoS, DDoS, Infiltration, Heart-bleed, Bot and Scan*. The dataset is made available in the *PCAP* file format, with one file for each day [2].

Labels for this dataset are provided as time-ranges. Each time-range specifies the type of attack, the attacker machine and the victim machines.

3.1.2 MAWI

WIDE is a Japan based research consortium that runs their own internet testbed and is in charge of multiple internet operations including backbone networks and name servers. It collects anonymized traffic data from its backbone at multiple sample points [26]. Data is collected every day for 15 minutes. With the data being sampled from a backbone network, these 15 minutes cover a large amount of traffic. A *PCAP* file containing the data worth of one day is around 10 gigabytes. In this thesis the data from samplepoint-f on May 1st 2021 is used [6].

MAWILab is an ensemble of network traffic anomaly detectors developed by the MAWI working group [31]. They developed a classification strategy based on similarity estimation and association rule mining. The traffic is categorized into benign, anomalous, and suspicious, with 10 subcategories of the last two [7]. The subcategories and benign are used as labels in this thesis. These 10 labels are *other, http, alpha, tunneling, port scan, icmp scan, udp scan, dos*. When using this dataset, some flows need to be removed because of ICMP probing caused by the USC ANT project.

Labels for this dataset are provided as tuples of (source IP, source port, destination IP, destination port).

Before we can use the datasets as input for our models, we need to bring them into a format which they understand. This is done by creating network traffic representations.

3.2 Network Traffic Representations

We have introduced different types of feature representations in Chapter 2.1.3 and discussed how they are linked to specific algorithms and models. Following we describe how flows are defined and extracted and how the different feature representations are created. It is important to note that we specifically look at encrypted network traffic. Encryption heavily limits the number of possible features. The encryption protocols TLS and IPSec, as introduced in Chapter 2.1.2, both offer a high level of encryption. IP address, packet direction, packet-(inter)-arrival time and packet length are the only features that are guaranteed to be observed. This directly impacts the definition of the extracted network flows.

3.2.1 Flow Key

Features which are used to define a flow are called flow keys. We discussed the different types of flow keys and what type of traffic they capture in Chapter 2.1.1. The traditional flow key is a 5-tuple key (srcIp, dstIp, srcPort, dstPort, protocolId) and captures traffic on application level. When IPSec encryption is used, we have no access to ports or the protocol identifier and are limited to 2-tuple and 1-tuple flow keys. In this thesis we describe feature representations that should work regardless of the level of encryption and therefore we onlylimit ourselfs to 2-tuple and 1-tuple flow keys.

We extract 2-tuple and 1-tuple flows from network traffic captures (*PCAP* files). With these flow keys we can capture communication on host and endpoint level. Flows are extracted with a maximum duration of 60 seconds by setting both, idle timeout and active timeout to 60 seconds. In total, 3 different types of flows are extracted from each dataset. The features of each vector are based on established feature vectors used in other research [60].

- The 2-tuple key extracts the TA feature vector. Main information obtained from the TA vector are patterns in backbone Internet traffic according to unidirectional, straightforward time series footprints [60] [41].
- The 1-tuple key extracts the AGM feature vector: The AGM vector was designed for discovering profiles in the Internet Background Radiation. It is based on observing network devices rather than flows [60] [42].

Note that the TA feature vector covers bidirectional traffic and is only extracted once. The AGM feature vector covers unidirectional traffic and is extracted twice. Once with *sourceIPAddress* as flow key and once with *destinationIPAddress* as flow key. Table 3.1 shows a comparison of the different flow keys for each feature vector. Using these 3 types of flows we can create all the feature representations we need.

Key Feature	TA	AGM $_s$	AGM $_d$
sourceIPAddress	•	•	.
destinationIPAddress	•	.	•

Table 3.1: Flow keys of TA, AGM_S and AGM_D vectors

As proposed in [60], combining different feature vectors leads to deeper insights in the analyzed traffic. By combining the 2-tuple TA vector and the two 1-tuple AGM vectors we create a new vector which covers communication statistics, attacker statistics, and victim statistics depending on the AGM vector direction. The new combined feature vector is called *multi-key* feature vector. The *multi-key* feature vector dataset is created similar to a left-join and is done in two steps:

1. An intermediary dataset is created by joining the TA dataset with the AGM_S dataset based on matching *sourceIPAddress*. For each feature vector f_{TA} in the TA dataset the closest feature vector f_{AGMS} from the AGM_S dataset, where *flowStartMilliseconds* is less than or equal to *flowStartMilliseconds* of f_{TA}, is determined the matching pair.

2. The final *multi-key* dataset is created by repeating the previous step but instead of the TA dataset the intermediary dataset is used and instead of the AGM_S dataset the AGM_D dataset is used.

This is only one way of creating the *multi-key* feature vector and it is not guaranteed to be the most optimal. Matching flows by closest timestamp may result in overlapping combinations, where AGM_S and AGM_D flows can occur mutliple times in the *multi-key* feature vector. However, the resulting feature vector shows robust performance and this method of rigorously joining has been applied before [60].

Next we will discuss how we can use the extracted feature-vectors to create the representations that will serve as inputs to our models.

3.2.2 Baseline Representations

We use a random forest based approach to establish an evaluation baseline for our deep learning based approaches. The performance of the baseline model is used as reference against which we can compare our DL based approaches. Random forests need numeric or categorical data as input. Using the TA, AGM and *multi-key* flows defined in the previous section, we can create a fixed size feature-vector as input for our random forest model. Because we want to investigate the impact of using a combined flow in comparison to a single flow, two feature-vectors are created. One is based on the 2-tuple TA vector and one is based on the *multi-key* vector. Table 3.2 shows the resulting 2-tuple and *multi-key* feature vectors based on the extracted flows.

Table 3.2: 2-tuple and *multi-key* feature vector with features used for training the baseline classifier. Features with the **FK** prefix are the flow keys that define the specific flow.

Vector	Feature	2-tuple	*multi-key*
TA	flowStartMilliseconds	.	.
	flowDurationMilliseconds	●	●
	FK sourceIPAddress	.	.
	FK destinationIPAddress	.	.
	packetTotalCount [forward]	●	●
	min(ipTotalLength) [forward]	●	●
	max(ipTotalLength) [forward]	●	●
	median(ipTotalLength) [forward]	●	●
	mean(ipTotalLength) [forward]	●	●

Continued on next page

Table 3.2 – continued from previous page

Vector	Feature	2-tuple	*multi-key*
	mode(ipTotalLength) [forward]	•	•
	stdev(ipTotalLength) [forward]	•	•
	min(_interPacketTimeSeconds) [forward]	•	•
	max(_interPacketTimeSeconds) [forward]	•	•
	median(_interPacketTimeSeconds) [forward]	•	•
	mean(_interPacketTimeSeconds) [forward]	•	•
	variance(_interPacketTimeSeconds) [forward]	•	•
	min(_interPacketTimeSeconds) [forward]	•	•
	packetTotalCount [backward]	•	•
	min(ipTotalLength) [backward]	•	•
	max(ipTotalLength) [backward]	•	•
	median(ipTotalLength) [backward]	•	•
	mean(ipTotalLength) [backward]	•	•
	mode(ipTotalLength) [backward]	•	•
	stdev(ipTotalLength) [backward]	•	•
	min(_interPacketTimeSeconds) [backward]	•	•
	max(_interPacketTimeSeconds) [backward]	•	•
	median(_interPacketTimeSeconds) [backward]	•	•
	mean(_interPacketTimeSeconds) [backward]	•	•
	variance(_interPacketTimeSeconds) [backward]	•	•
	min(_interPacketTimeSeconds) [backward]	•	•
	min(_interPacketTimeSeconds)	•	•
	max(_interPacketTimeSeconds)	•	•
	median(_interPacketTimeSeconds)	•	•
	mean(_interPacketTimeSeconds)	•	•
	variance(_interPacketTimeSeconds)	•	•
	packetTotalCount	.	.
	accumulate(_ipTotalLength)	.	.
	accumulate(_interPacketTimeMicroseconds)	.	.
	accumulate(flowDirection)	.	.
	accumulate(protocolIdentifier)	.	.
AGM_S	flowStartMilliseconds	.	.
	flowDurationMilliseconds	.	•
	FK sourceIPAddress	.	.
	packetTotalCount	.	.
	distinct(destinationIPAddress)	.	•
	mode(destinationIPAddress)	.	.
	modeCount(destinationIPAddress)	.	•
	min(_interPacketTimeSeconds)	.	•
	max(_interPacketTimeSeconds)	.	•

Continued on next page

Table 3.2 – continued from previous page

Vector	Feature	2-tuple	*multi-key*
	median(_interPacketTimeSeconds)	.	•
	mean(_interPacketTimeSeconds)	.	•
	variance(_interPacketTimeSeconds)	.	•
	mode(_interPacketTimeSeconds)	.	•
	min(ipTotalLength)	.	•
	max(ipTotalLength)	.	•
	median(ipTotalLength)	.	•
	mean(ipTotalLength)	.	•
	variance(ipTotalLength)	.	•
	distinct(ipTotalLength)	.	•
	mode(ipTotalLength)	.	•
	accumulate(ipTotalLength)	.	.
	accumulate(_interPacketTimeMicroseconds)	.	.
AGM_D	flowStartMilliseconds	.	.
	flowDurationMilliseconds	.	•
	FK destinationIPAddress	.	.
	packetTotalCount	.	.
	distinct(sourceIPAddress)	.	•
	mode(sourceIPAddress)	.	.
	modeCount(sourceIPAddress)	.	•
	min(_interPacketTimeSeconds)	.	•
	max(_interPacketTimeSeconds)	.	•
	median(_interPacketTimeSeconds)	.	•
	mean(_interPacketTimeSeconds)	.	•
	variance(_interPacketTimeSeconds)	.	•
	mode(_interPacketTimeSeconds)	.	•
	min(ipTotalLength)	.	•
	max(ipTotalLength)	.	•
	median(ipTotalLength)	.	•
	mean(ipTotalLength)	.	•
	variance(ipTotalLength)	.	•
	distinct(ipTotalLength)	.	•
	mode(ipTotalLength)	.	•
	accumulate(ipTotalLength)	.	.
	accumulate(_interPacketTimeMicroseconds)	.	.

3.2.3 Deep Learning Representations

CNNs use filters to extract features and information from images. By providing 2D representations or images constructed from flow features, the CNN can extract information, patterns and statistical correlations in the training data. Thus, we eliminate the need for creating large and complex feature vectors. We expect the CNN models to only learn features that are relevant for the classification task, eliminating variance from features that yield little to no information and find features that were not considered (or could not be considered) in the baseline feature vectors. Additionally, a visual representation can also help humans to better understand the data and flow behaviors.

Following we first present a state of the art image representation called FlowPic. Then we introduce our own image representation called TUWpic. Both can be generated from the flows extracted in Chapter 3.2.1.

FlowPic

FlowPic is a state of the art 2d-feature representation proposed by Shapira et al. [80]. It is based on only two features, which makes it a perfect candidate for comparison because the features are available in encrypted traffic.

FlowPics can be seen as arrays of Payload Size Distributions (PSD) with one PSD for each interval between packets [80]. The final image is a histogram with packet inter-arrival-time (iat) on the X-axis and packet length on the Y-axis. To create these images a list of tuples *(IP packet length, time of arrival)* has to be constructed for each flow. The maximum length of an IP packet is 1500 bytes as defined by the Ethernet Maximum Transmission Size (MTU) value. Packets larger than 1500 bytes are not considered during image construction. In order to create square images, the packet inter-arrival-time also has to be scaled to the same dimension. In the FlowPic paper, flows are limited to 60 seconds. Using a 60 to 1500 mapping, the inter-arrival-times are scaled to the range of 0 to 1500. Now the normalized list of pairs of a flow can be mapped to a 1500×1500 histogram. The value in each cell is equal to the number of pairs assigned to that cell. Each of these histograms is then converted to a pixel matrix, exported as an image and called a FlowPic. The algorithm for creating a FlowPic is defined in Algorithm 3.1. The classification problem in [80] includes many traffic intense categories such as video calls or VoIP. After some initial tests with the IDS2017 dataset, it showed that most flows have sparse packets which resulted in many images being almost black. To resolve this issue we modified the FlowPic method so that the value in the histogram is not based on the count of matching tuples but rather is set to 255 (white) if there is any matching tuple or 0 (black) if there is no matching tuple. This idea was already used in [80] to visualize the images but was not used for classification. Additionally, instead of scaling only the inter-arrival-times to a range of 0 to 1500, we scale both packet inter-arrival-time and packet size to the range of 0-127 to create smaller 128x128 size images. This is done to significantly increase training time and reduce the image size.

Examples of generated **adapted FlowPics** can be seen in Figure 3.1.

(a) Adapted FlowPic (Benign) (b) Adapted FlowPic (Botnet)

Figure 3.1: Adapted FlowPic with x-Axis showing packet inter-arrival-times and y-axis showing packet length. White pixels indicate existing packets.

Algorithm 3.1: FlowPic

Input: A feature-vector $flow$
Output: A FlowPic F

$F \leftarrow$ init zero matrix [128,128];
foreach *tuple t (iplen, iat)* $\in flow$ **do**
  if $t_{iplen} \leq 1500$ **then**
    $iat_s \leftarrow$ map t_{iat} from range(1,60000000) to range(0,127);
    $iplen_s \leftarrow$ map t_{iplen} from range(1,1500) to range(0,127);
    $F[iat_s, iplen_s] \leftarrow 255$;
  end
end
return F;

TUWpic

TUWpic is a dynamic size 2D feature representation designed for use with bidirectional 2-tuple flows using packet length, packet inter-arrival-time, and packet direction for construction. TUWpic tries to keep as much information of the original flow as possible using available features (packet length, packet inter-arrival-time, and packet direction) that have shown meaningful results in NTA classification problems. By keeping the sequentiality of the flow, we also encode temporal information into the resulting representation. In FlowPic, this information is lost. In TUWpic, we draw variable packet features in a consistent way resulting in repeating patterns. Using convolutional layers in CNNs we can train kernels to detect these generic patterns across the resulting images.

The dimension of TUWpic is defined by the maximum number of packets $pktsMx$ that should be considered from each flow. The packets are sorted by arrival time and if a flow has more packets than $pktsMx$, the additional packets are discarded. The parameter $pktsMx$ can be choosen freely and is set to 128 in this thesis. Choosing the number of packets to include is not trivial and limitations impossed by the target feature representation have to be considered. Most of the image based feature representations presented in Chapter 2.1.4 use all available packets in a flow or as many as possible. Chen et al. [25] state that they achieved better performance using more packets but only used 10 in the final experiments to guarantee on-the-fly detection without performance issues. We have choosen 128 packets because $pktsMx$ also defines the resulting image size and we wanted to generate images having the same size as the FlowPic images. In this thesis, all TUWpic based generated images have a size of 128x128. Starting from top-left, each row in the matrix represents a packet. The contents of each row are split into two sections $vbins$, where each section is $vbin = \frac{pktsMx}{2}$ (image half size). Each half represents one direction of the flow resulting in a matrix where the left half of the matrix covers all incoming packets and the right side covers all outgoing packets. This matrix is converted into a gray-scale image and called TUWpic.

From each packet within a flow (row in the resulting image) the row-value f and the row-value-span l are calculated, where f is a value between 0 and 1 determining the resulting greyscale pixel color and l is the number of columns (pixels) it spans. We propose two methods to calculate the row-value and row-value-span for each packet. We refer to them as method A and method B and the resulting image representations are called **TUWpicA** and **TUWpicB** respectively.

TUWpicA

Using method A, the row-value f_A is based on the packets inter-arrival-times, and the row-value-span l_A is based on the packet length. The value f_A is close to 1 when the packet inter-arrival-time is low and close to 0 when the inter-arrival-time increases. With f_A being at least 0.1 indicating that a packet arrived. We do this to avoid black or near black rows that could appear in the middle of the flow if the inter-arrival-time is very high. By introducing a minimum pixel value for existing packets we make a clear

separation between packet and no packet. The mathematical expression for f_A is given in Equation 3.1. The row-value-span l_A is based on the ratio of packet length to maximum packet length. The equation for l_A is given in Equation 3.2.

The row-value f_A is calculated as:

$$f_A = \max(1 - log(iat)/log(iatMx), 0.1) \tag{3.1}$$

where iat is packet inter-arrival-time of the current packet and $iatMx$ is the inter-arrival-time in microseconds at which f_A becomes 0. In this thesis $iatMx$ is set to $iatMx = 5000000$. The log function is used to better cover the scale of the inter-arrival-times. We are interested in algorithmic patterns in the flow and inter-arrival-times can contain such information. Using log the resulting value will be mapped to a certain range inside 0 to 1 depending on the magnitude (microseconds, milliseconds, seconds) of the inter-arrival-time. If the inter-arrival-time is small and therefore the packet was received shortly (microseconds) after the previous packet f_A will be high. If the difference is in the millisecond range, f_A will be in the vicinty of 0.5. If there are seconds between two packets, meaning a high inter-arrival-time, f_A will be close to 0.

The row-value-span or column span l_A is calculated as:

$$l_A = int(vbins * len/lenMx) \tag{3.2}$$

where $vbins = \frac{pktsMx}{2}$, len is the current packet length and $lenMx$ is the packet length at which $l_A = vbins$.

Examples of generated **TUWpicA** can be seen in Figure 3.2.

(a) TUWpicA (Benign) (b) TUWpicA (Botnet)

Figure 3.2: TUWpicA

TUWpicB

Method B swaps the features on which f_B and l_B are based. The row-value f_B is now defined by the packet size. l_B is 1 when the packet size is the same as the maximum packet size $pktsMx$ and decreases with smaller packet size. As before, the value is capped at 0.1 to indicate the existence of a packet. The row-value-span l_B increases if the inter-arrival-time of a packet is low and decreases if the inter-arrival-time-is-high. The row-value-span is at least 1. The equations for above calculations are given in Equation 3.3 and Equation 3.4.

The row-value f_B is calculated as:

$$f_B = 0.9 * (len/lenMx) + 0.1 \tag{3.3}$$

where len is the current packet length and $lenMx$ is the packet length at which $f_B = 1$.

The row-value-span or column span l_B is calculated as:

$$l_B = int(\max(1 - log(iat)/log(iatMx), 1/vbins) * vbins) \tag{3.4}$$

where iat is packet inter-arrival-time of the current packet and $iatMx$ is the packet inter-arrival-time in microseconds at which $l_B = 1$. In this thesis $iatMx$ is set to $iatMx = 5000000$.

Examples of generated **TUWpicB** can be seen in Figure 3.3.

(a) TUWpicB (Benign) (b) TUWpicB (Botnet)

Figure 3.3: TUWpicB

TUWpic-mk

TUWpic-mk is based on TUWpic and tries to improve on it by not only considering a single 2-tuple flow but utilizing the multi-key flow format introduced in Chapter 3.2.1.

This gives us more information to encoded than what similar approaches in the literature usually have to disposal. Since the multi-key feature vector consists of 3 different flows (TA, AGM_S, AGM_D), an intuitive way of transforming it into an image is to assign each feature vector to a different color channel. Each channel then represents a different type of communication. We can reuse the definitions of TUWpic from above and assign it to the red color channel. Following we present two ways for transforming a 1-tuple AGM flow into a 2D matrix. We then use these transformations to build **TUWpic-mkA** and **TUWpic-mkB** by assigning the transformed AGM_S and AGM_D matrices to the green and blue color channels.

Calculating row-value and row-value-span for 1-tuple flows use the same equations as described in the 2-tuple flow scenario. The resulting images differ in two ways. First, the starting index is not defined by the flow direction. Instead, each row is filled from the start, up to the calculated row-value-span. Second, a new variable related to the number of source and destination IP addresses (depending on the AGM vector) is introduced. The value of this variable $f2$ is always 1, and the starting index within each row is $\frac{pktsMx}{2}$. The length of the row-value-span $l2$ is calculated as follows.

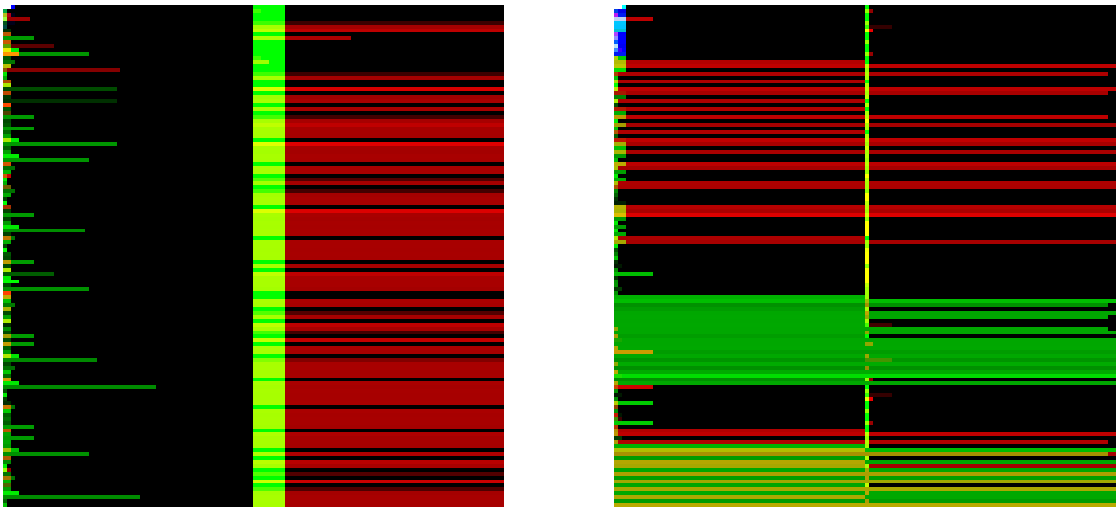
Given the number of distinct destination or source IP addresses of a flow $distIPs$ and a user defined variable $hostMX$ representing the maximum number of IP addresses, the length $l2$ of row-value $f2$ can be calculated as shown in Equation 3.5

$$l2 = int(vbins * \min(1, \frac{distIPs}{hostMX})) \tag{3.5}$$

where $vbins = \frac{pktsMx}{2}$. The row-value-span $l2$ becomes larger as $distIPs$ is getting closer to $hostMX$. In this thesis $hostMX$ is set to $hostMX = 500$.

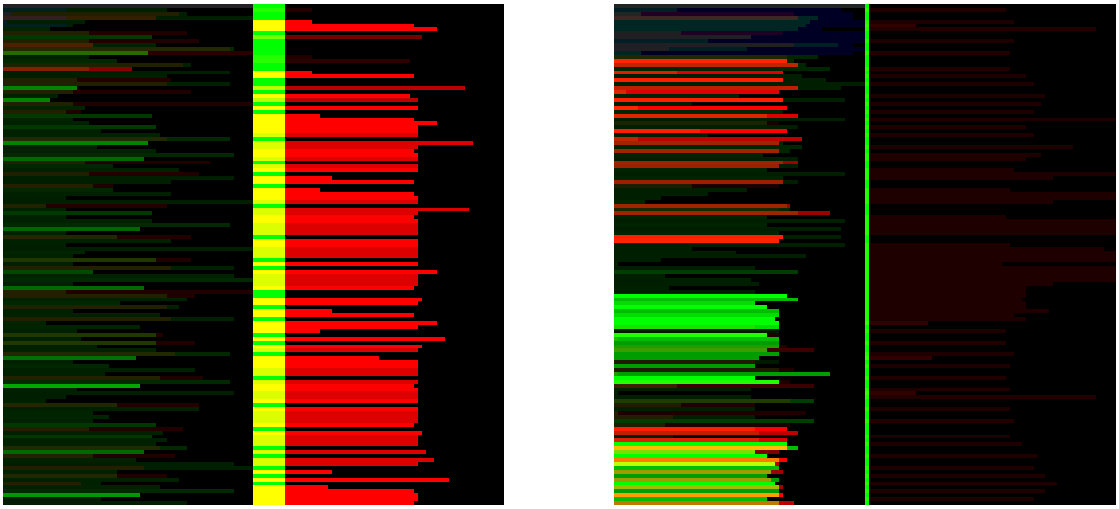
Examples of generated **TUWpic-mkA** and **TUWpic-mkB** can be seen in Figure 3.4 and Figure 3.5.

The algorithms for **TUWpicA**, **TUWpicB**, **TUWpic-mkA** and **TUWpic-mkB** are shown in Algorithms 3.2, 3.3,3.4 and 3.5. Table 3.3 shows the main differences of the image representations. Since we are dealing with large datasets, we have to efficiently handle the creation and storage of image representations. In the next section, we discuss how some of the problems imposed by large image datasets can be mitigated.



(a) TUWpic-mkA (Benign) (b) TUWpic-mkA (Botnet)

Figure 3.4: TUWpic-mkA. The first channel (red) is the most prominent as it covers bidirectional flow data. The green channel is the second most prominent as it shows data sent from the source to the destination, this is expected as more data is typically downloaded than uploaded. The blue channel covers outgoing traffic from the victim to the attacker and has the least amount of packets. We can see that the amount of blue lines increases in the botnet image, probably due to communication between the remote botnet master.



(a) TUWpic-mkB (Benign) (b) TUWpic-mkB (Botnet)

Figure 3.5: TUWpic-mkB

Algorithm 3.2: TUWpicA

Input: A feature-vector X; a scalar $pktsMx$, a scalar $vbins$, a scalar $lenMx$, a scalar $iatMx$

Output: A TUWpicA TA

$TA \leftarrow$ init zero matrix $[pktsMx, 2 * vbins]$;
foreach *index* i, *tuple* t *(iat, iplen, flowdir)* $\in X$ **do**
  $v \leftarrow$ zero vector of size $2 * vbins$;
  if $i \geq pktsMx$ **then**
    break for;
  end
  $f \leftarrow \max(1 - log(t_{iat})/log(iatMx), 0.1)$;
  $l \leftarrow int(vbins * t_{iplen}/lenMx)$;
  $i \leftarrow t_{flowdir} * vbins$;
  $v[i : i + l] \leftarrow f$;
  $TA[j, :] \leftarrow v$;
end
return TA;

Algorithm 3.3: TUWpicB

Input: A feature-vector X; a scalar $pktsMx$, a scalar $vbins$, a scalar $lenMx$, a scalar $iatMx$

Output: A TUWpicB TB

$TB \leftarrow$ init zero matrix $[pktsMx, 2 * vbins]$;
foreach *index* i, *tuple* t *(iat, iplen, flowdir)* $\in X$ **do**
  $v \leftarrow$ zero vector of size $2 * vbins$;
  if $i \geq pktsMx$ **then**
    break for;
  end
  $f \leftarrow 0.9 * (t_{len}/lenMx) + 0.1$;
  $l \leftarrow int(\max(1 - log(t_{iat})/log(iatMx), 1/vbins) * vbins)$;
  $i \leftarrow t_{flowdir} * vbins$;
  $v[i : i + l] \leftarrow f$;
  $TB[j, :] \leftarrow v$;
end
return TB;

Algorithm 3.4: 1-tuple image method A

Input: A feature-vector X; a scalar $pktsMx$, a scalar $vbins$, a scalar $lenMx$, a scalar $iatMx$, a scalar $hostMx$

Output: An image I

$I \leftarrow$ init zero matrix $[pktsMx, 2 * vbins]$;
foreach *index* i, *tuple* t *(iat, iplen, distIPs)* $\in X$ **do**
  $v \leftarrow$ zero vector of size $2 * vbins$;
  if $i \geq pktsMx$ **then**
    break for;
  end
  $f \leftarrow \max(1 - log(t_{iat})/log(iatMx), 0.1)$;
  $l \leftarrow int(vbins * t_{iplen}/lenMx)$;
  $l2 \leftarrow int(vbins * min(1, t_{distIPs}/hostMx))$;
  $v[vbins : vbins + lp] \leftarrow 1$;
  $v[: l] \leftarrow f$;
  $I[j, :] \leftarrow v$;
end
return I;

Algorithm 3.5: 1-tuple image method B

Input: A feature-vector X; a scalar $pktsMx$, a scalar $vbins$, a scalar $lenMx$, a scalar $iatMx$, a scalar $hostMx$

Output: An image I

$I \leftarrow$ init zero matrix $[pktsMx, 2 * vbins]$;
foreach *index* i, *tuple* t *(iat, iplen, distIPs)* $\in X$ **do**
  $v \leftarrow$ zero vector of size $2 * vbins$;
  if $i \geq pktsMx$ **then**
    break for;
  end
  $f \leftarrow 0.9 * (t_{len}/lenMx) + 0.1$;
  $l \leftarrow int(\max(1 - log(t_{iat})/log(iatMx), 1/vbins) * vbins)$;
  $l2 \leftarrow int(vbins * min(1, t_{distIPs}/hostMx))$;
  $v[vbins : vbins + lp] \leftarrow 1$;
  $v[: l] \leftarrow f$;
  $I[j, :] \leftarrow v$;
end
return I;

Name	x-axis	y-axis	W	R	G	B
FlowPic	iat	size	packet			
TUWpicA	dir + size	packet	iat			
TUWpicB	dir + iat	packet	size			
TUWpic-mkA	dir + size	packet		iat	iat + IP addresses	iat + IP addresses
TUWpic-mkB	dir + iat	packet		size	size + IP addresses	size + IP addresses

Table 3.3: Comparison of image based feature represenations. IATs are packet inter-arrival-times and size refers to packet length and dir refers to the direction of the packet (incoming or outgoing).

3.2.4 Handling image data

Using large image datasets imposes some challenges that need to be addressed. Generating images for all samples beforehand leads to a lot of single files. Dealing with hundreds of thousands single image files quickly becomes very messy and hard to manage. Also, reading and writing single files can lead to performance bottlenecks during training. Depending on the training environment the data could be stored in the cloud or on slow hard drives.

One way to address the storage problem is to generate the images during training and testing in memory, rather then permanently storing them. This addresses the problem of handling a lot of files but depending on the image generation method, it could be even slower than reading and writing them.

Both problems can be addressed by using a data format specifically developed for storing deep learning data. All deep learning experiments in this thesis are conducted using *Python* and *Keras* [5]. *Keras* is a high level deep learning framework for *Tensorflow*. Using *Tensorflow* we can utilize their custom file format *TFRecord*[14]. *TFRecord* uses Googles *Protocol Buffers*[10] serialization technology to efficiently store sequences of binary data. The *TFRecord* file format allows us to store the generated images in a single compressed file. A *TFRecord* dataset not only is an efficient data storage format, but since it is specifically developed for deep learning, it supports things like splitting a dataset in shards and storing them on different servers, shuffling data, caching data, creating batches, and a lot of other useful functions.

We now have showed a way to create and store feature representations that can be used as inputs to our ML and DL models. Before we go over these models, we need to define which experiments we want to conduct and how we are going to evaluate them.

3.3 Classification Tasks and Goals

In order to evaluate the performance of the proposed representations, we evaluate our models using binary and multiclass classification. Both datasets provide ground truth data that allows for easy labeling of flows with different anomalous categories or as benign traffic. Anomalous traffic can be a multitude of categories depending on the dataset. When doing binary classification, all non-benign labels are replaced by a single *Attack* label.

Typical problems of IDS and in anomaly detection are high *False-Positive (FPR)* rates and low *True-Positive-Rates (TPR)* [63]. The goal is to maximize *TPR* and minimize *FPR* [18]. This is especially a challenge in IDS, because most samples belong to the benign class. This is one of the main reasons *accuracy* is not a significant evaluation metric for imbalanced data or data that is skewed towards one class [43]. It will always result in a high score because benign traffic is classified as such, but the metric ignores the skewness of the data. We therefore use the following metrics that are also common in the field and in imbalanced classification, e.g., confusion matrix, fale-positive-rate,

precision, recall, f1, auc-roc and average precision [87][18][91][69]. The most promising and interesting metrics to look at are F1 and average-precision. They tend to deliver the most reliable results in imbalanced binary classification tasks [75].

Following we describe the individual metrics that ared used for evaluation, before going over the individual models and architectures used for evaluation in the next section.

Confusion Matrix

A confusion matrix is a $n \times m$ matrix with n and m being the number of classes. For a binary classification task a confusion matrix would be of size 2×2, for a multiclass classification task with 7 different classes the confusion matrix would be of size 7×7. The rows of a confusion matrix indicate the true label, the columns the predicted labels. Based on the value in each element the following statistics can be retrieved for each class: **True-Positive (TP)**, **True-Negative (TN)**, **False-Positive (FP)**, **False-Negative (FN)**. In a 2×2 confusion matrix this would look like depicted in Figure 3.6.

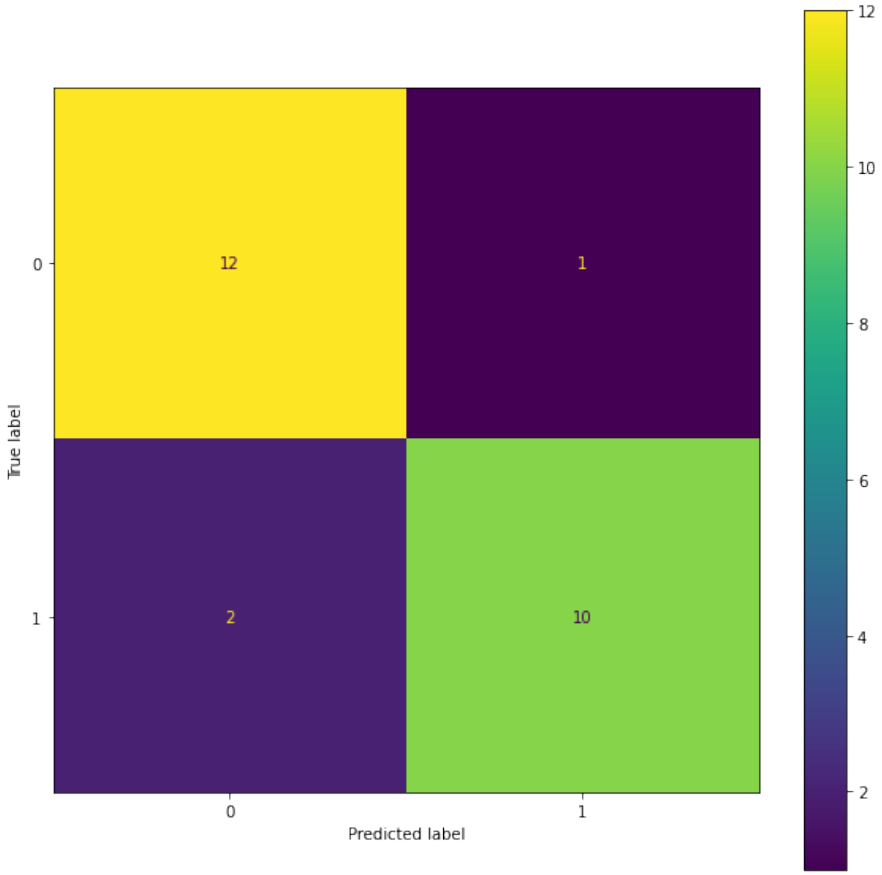


Figure 3.6: Example of a binary confusion matrix C with $TN = C_{0,0}$, $FN = C_{1,0}$, $TP = C_{1,1}$ and $FP = C_{0,1}$. [12].

Accuracy

Accuracy is an indicator that shows how many samples are correctly classified. It is not very expressive for skewed datasets for the reasons we talked about above. Accuracy is defined as:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.6}$$

False-Positive Rate

The False-Positive Rate (FPR) is an indicator for how many negative samples the model predicted positive. It is complementary to recall and defined as:

$$FPR = \frac{FP}{TN + FP} \tag{3.7}$$

Precision

Precision is an indicator that measures how often a model is correct about its positive predictions. Precision is defined as:

$$Precision = \frac{TP}{TP + FP} \tag{3.8}$$

Recall (True-Positive Rate)

Recall or True-Positive Rate (TPR) is a measure of how well a model recognizes positive samples. The metric needs to be evaluated together with other metrics since it can be easily manipulated by just classifying everything as positive:

$$Recall = \frac{TP}{TP + FN} \tag{3.9}$$

F1-Score

The F1-Score (F1) is the harmonic mean of precision and recall. It is a combined metric that gives precision and recall the same importance. It is defined as:

$$F1Score = 2 \times \frac{(Precision \times Recall)}{Precision + Recall} \tag{3.10}$$

ROC-AUC

Receiver Operator Characteristic (ROC) curve is a plot of FPR against recall at multiple thresholds. Computing the AUC (Area Under the Curve) is an indicator for how well a model performs. An AUC of 1 is a perfect classifier, an AUC of 0.5 is a purely random classifier. Figure 3.7 show an example of an ROC curve.

Average Precision

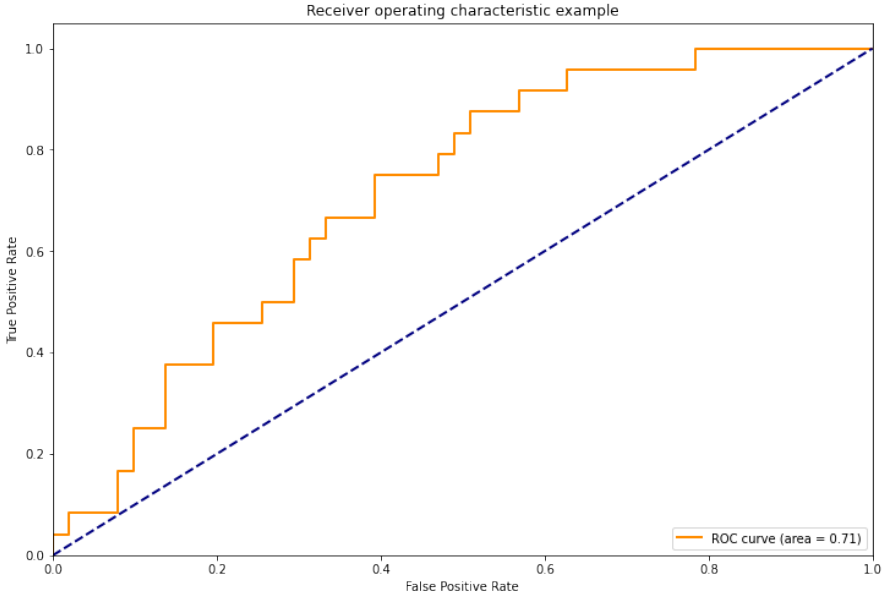


Figure 3.7: ROC curve with AUC [11]

Average Precision (AP) can only be used to evaluate binary scores (or per class scores). "AP summarizes a precision-recall curve as the weighted mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight." [11] The Precision-Recall Curve (PRC) is not a metric on its own but a way to show the relation (trade-of) between Precision and Recall. To do this, precision and recall values are plotted against each other at different thresholds. A good classifier has both, high precision and high recall.

$$AP = \sum_{i=0}^{n} (R_i - R_{i-1}) \times P_i \tag{3.11}$$

where n is the number of thresholds.

3.4 Classification Algorithms and Models

The main focus of this thesis is to investigate DL models and associated feature representations. Since we are using only 2-tuple flows and a relatively new dataset there is little to none comparable work that can be used for performance comparison. We therefore propose a random forest ensemble as a baseline evaluation model, so that we can put the DL results into context.

3.4.1 Baseline Model

Random Forests (RF) are used in IDS and traffic classification across the literature and have shown to be the best option many times [40]. We use an ensemble of 100 estimators and apply halving grid search [13] with 5-fold cross validation.

The halving grid search iteratively removes parameters that do not perform well. This is done by testing every parameter pair on a model with reduced resources. When used with RF, the resource is the number of estimators. This way, all possible parameter combinations are tested with a fast model. With every iteration the resource is increased while the number of parameters decreases.

Cross Validation (CV) is used to validate a model's performance without evaluating it on the test data. This is needed when searching for the best hyperparameters in a grid search. Without CV an additional validation dataset is needed in order to evaluate the models performance since it is not good practice to tune a model on the test set. The train set also cannot be used to estimate the model's performance, because the model has already seen all the samples in the dataset. With CV, we can overcome this problem by splitting the training data equally into n-folds. Each training must now be done n times. During each cycle, 1 of the n folds is left out and used as a validation set. The average of the performance across all folds is then the final performance metric and can be used for hyperparameter selection.

In summary the baseline model can be described as follows. The baseline model is a RF ensemble of 100 estimators. The hyperparameters *max_depth* and *min_samples_split* are tuned using halving grid search and $5-fold$ CV. The halving grid search uses a factor of 2 as halving parameter. The halving parameter defines the proportion of hyperparameters to keep for the next iteration. With a halving parameter of 2, half of the hyperparameters are selected, a parameter of 3 would keep a third of the hyperparameters. The number of resources per hyperparameter pair is limited to a maximum of 20. Meaning during the grid search, each combination uses an ensemble of 20 estimators.

The RF model is needed in order to have a valid baseline to which we can compare the DL models which are presented next.

3.4.2 Neural Network Models

Deep Learning (DL) and Convolutional Neural Networks (CNNs) are used in different studies for network traffic classification and intrusion detection [25][98][80]. Choosing

the right CNN architecture is a key factor in optimizing model performance. In some preliminary tests we compare 5 CNN architectures against each other using the IDS2017 dataset. Because all of these models are rather "shallow", we also include an additional, efficient, state-of-the-art model - **MobileNetV3**. The used architectures are described below. We tried to keep the models as close to the literature as possible and only adjusted the input layers to work with the proposed feature representations. If more adjustments where made, it will be described in the corresponding models section.

Feature images are stored with color values in the range of 0-255 to save space, each input layer is at the same time a re-scaling layer that normalizes the color values. This layer is ignored in the following descriptions of the CNN architectures. Also, the activation function is always assumed to be ReLu.

FlowPic

In Shapira et al. [80], not only a novel image based feature representation is proposed but also a CNN architecture. The proposed architecture is based on the famous LeNet-5 CNN architecture [54].

The model has a 1500x1500x1 size input layer. The first convolutional layer consists of 10 10x10 filters with a stride of 5. Following, a max-pooling layer with the size of 2x2 is applied. Then another convolutional layer with 20 filters of size 10x10 and stride of 5, followed by another 2x2 max-pooling layer are used. The max-pooling layer is followed by a flatten layer and then a fully-connected layer with 64 units. The 64 dense layer is connected to the output layer.

Because we are working with smaller images of size 128x128 the input layer was adjusted and also the stride of the second convolutional layer cannot be applied and a stride of 2 is used instead. The architecture can be seen in Figure 3.8

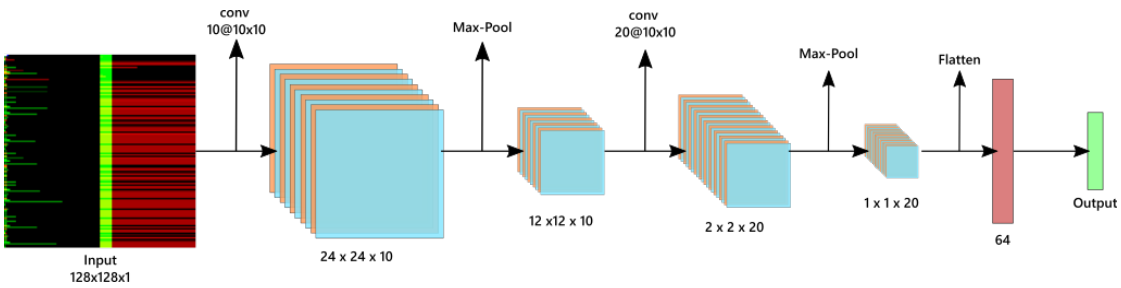


Figure 3.8: FlowPic CNN architecture

Seq2Img

Similar to FlowPic, Seq2Img is an image based feature representation and a CNN architecture at the same time. The architecture is based on 6 channel images and adapted accordingly to be used with images of size 128x128x1 and 128x128x3. The first

convolutional layer consists of 32 5x5 filters with size 5, followed by a max-pooling layer. The same two layers are repeated except that the number of filters is increased to 64. Then, a flatten layer and two fully-connected layers with 1024 and 128 units follow before ending at the output layer.

The architecture can be seen in Figure 3.9

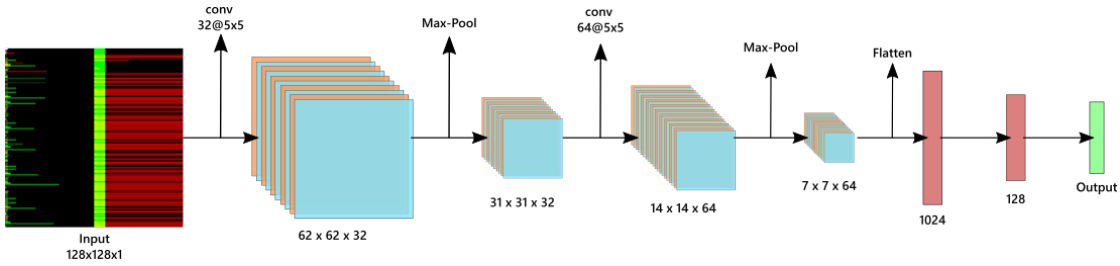


Figure 3.9: Seq2Img CNN architecture

Vec2Img

In Moustakidis et al. [63], an image based feature representation and an accompanying CNN architecture is proposed. The CNN starts with a single convolutional layer with 16 2x2 filters followed by a flatten layer and 3 fully-connected layers with size 1028 before reaching the output layer. The documentation of the architecture in the cited work is not very precise, and it is not clear if there was a pooling layer used after the convolutional layer. Following best-practice and other architectures a max-pooling layer was used. Additionally, the original input layer of 7x7x1 was changed to 128x128x1 and 128x128x3.

The architecture can be seen in Figure 3.10

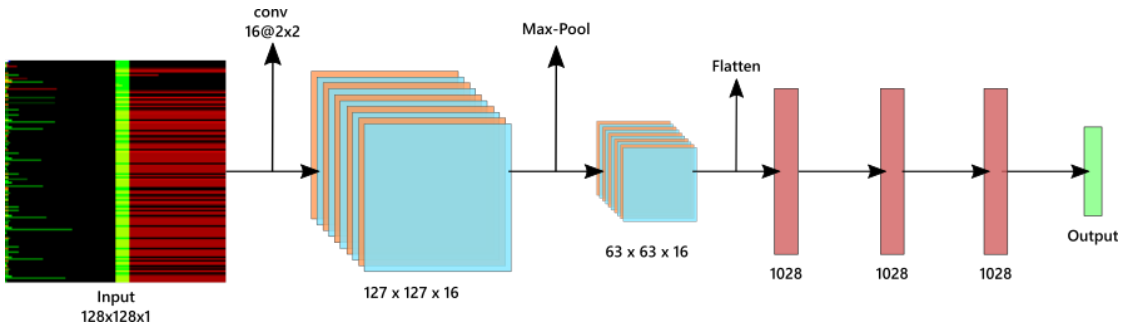


Figure 3.10: Vec2Img CNN architecture

LeNet5

A CNN using the original LeNet5 architecture with some modern features is proposed here. Instead of average-pooling, max-pooling is used. Also, the amount of filters in the

convolutional layers is increased as well as the number of neurons in the fully connected layers.

The adapted LeNet5 architecture starts with a convolutional layer with 32 5x5 filters followed by a max-pooling layer, another convolutional layer with 48 filters of size 5x5 and a final max-pooling layer. This is continued with a flatten layer and two fully-connected layers with 256 and 84 neurons each. This architecture is similar to Seq2Img but uses different combinations.

The architecture can be seen in Figure 3.11

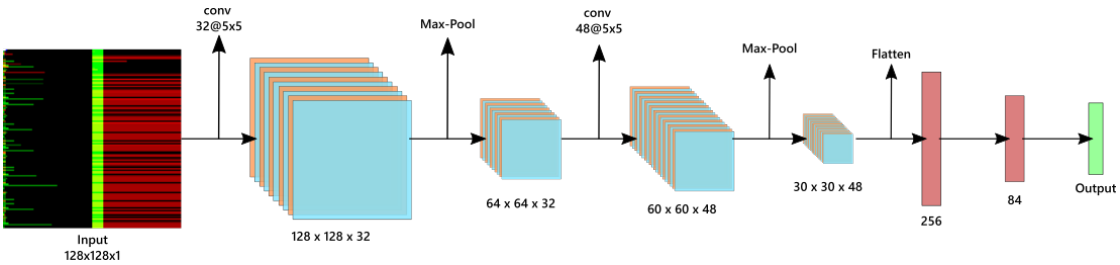


Figure 3.11: Adapted LeNet5 CNN architecture

MobileNetV3

Nowadays, the above architectures would be considered small. The original LeNet5 architecture only has 60000 trainable parameters. With more computing power we can use deeper networks with more parameters to retrieve better classification results. ID is a time and resource demanding area therefore we propose a CNN optimized for mobile devices which has similar requirements. **MobileNetV3Small** [36] with 2.4 million trainable parameters, has significantly more parameters than the original LeNet5 architecture. To use that many parameters in a mobile device, special mobile CNN building blocks were created (bneck). These building blocks are considered layers on their own when describing the architecture. **MobileNetV3Small** consists of 16 layers in total. Figure 3.12 shows the structure of such a bneck block, the overall structure can be seen in Table 3.4.

Siamese Neural Networks and Online Mining

We now have defined various feature represenations and models but are yet to address one of the main challenges of IDS dataset, imbalanced data. As we stated in Chapter 2.1.2, one way of handling imbalanced datasets is using distance metric learning [99]. In this thesis, this is implemented using siamese neural networks. We also discussed how online mining can be used to reduce the computational costs by creating pairs and triplets of samples during training (online) inside the loss functions. Applying online mining removes the need for twin or triplet networks, since they are only needed when pairs and triplets are generated beforehand and need to be processed at the same time.

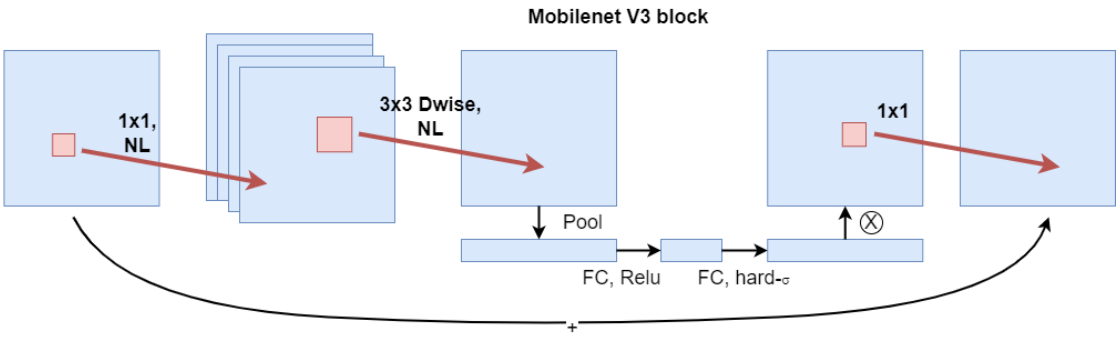


Figure 3.12: MobileNetV3 building block (bneck)
based on [36]

Input	Operator	exp size	#out	SE	NL	s
$224^2 \times 3$	conv2d,3x3	–	16	–	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	✓	RE	2
$56^2 \times 16$	bneck, 3x3	72	24	–	RE	2
$28^2 \times 24$	bneck, 3x3	88	24	–	RE	1
$28^2 \times 24$	bneck, 5x5	96	40	✓	HS	2
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	120	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	144	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	288	96	✓	HS	2
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	conv2d, 1x1	–	576	✓	HS	1
$7^2 \times 576$	pool, 7x7	–	–	–	–	1
$1^2 \times 576$	conv2d 1x1, NBN	–	1024	–	HS	1
$1^2 \times 1024$	conv2d 1x1, NBN	–	k	–	–	1

Table 3.4: MobileNetV3Small CNN architecture

The goal of this learning method is to create embeddings that minimize the distance between the same class and maximize the distance to other classes. These embeddings are then used to categorize new samples based on their embeddings. This method of learning is referred to as distance metric learning. In order for a network to learn such embeddings, special loss functions are needed. These loss functions expect the class labels and the embeddings of the feature as inputs whereas traditional loss functions use class labels and predicted labels. The last hidden layer is usually a fully-connected layer returning a vector also called embeddings of the size of neurons in that layer. In order to effectively measure the distance between two embeddings, they need to be normalized. In this thesis the *L2* norm is used for normalization. This can be achieved by replacing the

output layers of the above described CNNs with a normalization layer. Inside the loss function we can create a pairwise distance matrix of size $n \times n$ where n is the number of samples in the batch. From this distance matrix we can calculate the loss functions as described in Chapter 2.2.

Two loss functions and two online mining techniques are used in this thesis. Using matrix operations, the losses and online mining techniques can be implemented very efficiently.

Contrastive loss is implemented using the L2 norm (euclidean distance) and batch-all online mining. Batch-all mining calculates the loss for all available pairs in a batch. Pseudocode for the loss function is given in Algorithm 3.6 and works as follows. First a pairwise distance matrix d is created from the embeddings. Then a pairwise binary adjacency matrix *adjacency* is created using the labels. This matrix contains 1 for elements where the labels are equal and 0 for elements where the labels are different. These same size matrices that can be used to calculate the contrastive loss as defined in the following equation:

$$l = adjacency * d^2 + (1 - adjacency) \max(margin - d, 0)^2 \tag{3.12}$$

where *margin* is a regularization term indicating a minimum distance between negative pairs.

Algorithm 3.6: Contrastive loss with batch-all online mining

Input: A list of class labels *labels*, a list of embeddings *embeddings*, a scalar margin m

Output: Contrastive loss l

$d \leftarrow pairwise_distance_matrix(embeddings)$;
$adjacency \leftarrow pairwise_binary_adjacency_matrix(embeddings)$;
$l \leftarrow adjacency * d^2 + (1 - adjacency) * max(m - d, 0)^2$;
$l \leftarrow \frac{sum(l)}{len(labels)}$;
return l;

Triplet loss also uses the euclidean distance as a metric but uses online hard mining instead of batch all mining. Pseudocode for the triplet loss is given in Algorithm 3.7. The loss for a triplet (a, p, n) is defined as:

$$l = \max(d(a, p) - d(a, n) + m, 0) \tag{3.13}$$

where $d(a, p)$ is the distance between the anchor and the positive pair, $d(a, n)$ is the distance between the anchor and the negative pair and m is a margin term.

As for the contrastive loss d and *adjacency* are created. Additionally, the inverted version of *adjacency* is also computed $adjacency_not$. To compute hard negative samples we take the row-wise maximums from *dist* and subtract them from d. This results in a matrix

where 0 indicates elements that are far apart. Pairs are closer to each other the larger the negative number is, with the largest negative in each row being the negative row-wise maximum. By multiplying with $adjacency_not$ we eliminate positive pairs. Taking the row-wise minimum from this new matrix gives us the hardest negative pair for each sample. We then need to add the row-wise maximums back to get the original values. In order to calculate hard positives we set the diagonal of *adjacency* to 0 to mask out pairs of the same samples. Then we multiply the adjusted mask *adjacency* with d and take the row-wise maximums. The triplet loss can now be calculated using Equation 3.13.

Algorithm 3.7: Triplet loss with batch-hard online mining

Input: A list of class labels *labels*, a list of embeddings *embeddings*, a scalar margin m

Output: Triplet loss l

$d \leftarrow pairwise_distance_matrix(embeddings)$;
$adjacency \leftarrow pairwise_binary_adjacency_matrix(embeddings)$;
$adjacency_not \leftarrow invert(adjacency)$;
$row_maximums \leftarrow row_max(pairwise_distance)$;
$hard_negatives \leftarrow (pdist_matrix - axis_maximums) * adjacency_not$;
$hard_negatives \leftarrow row_min(hard_negatives) + row_maximums$;
$adjacency_mask \leftarrow adjacency - diagonal_matrix_ones(len(labels))$;
$hard_positives \leftarrow pdist_matrix * adjacency_mask$;
$hard_positives \leftarrow row_min(hard_positives)$;
$l \leftarrow \max(hard_positives - hard_negatives + m, 0.0)$;
$l \leftarrow mean(l)$;
return l;

3.5 Experimental Setup

The previous chapters and sections introduced the needed background knowledge, related work and descriptions of specific techniques and methods used in this thesis. In this chapter we want to focus on the actual implementation and execution of the experiments.

We designed and end-to-end process starting with network traffic captures and ending with the evaluation of trained models, this process can be split into the following logical steps:

1. flow extraction
2. preprocessing and labeling
3. creating train and test datasets
4. creating feature representations
5. model training
6. model evaluation

All data and scripts used during the experiment are stored in an accompanying repository. For reproducibility, all methods that involve random number generators are used with the same seed value. When not stated otherwise, all programming was done in *Python* [15] on Windows 10. The repository contains a step-by-step instructions file to reproduce the experiments.

3.5.1 Flow Extraction

We start the process by extracting the flows defined in Chapter 3.2.1 from the respective dataset *PCAP* files. *Go-flows* [1] is a flow extraction tool developed by the Institute of Telecommunications, TU Vienna which we can use for that job. *Go-flows* allows us to extract, compute statistics and aggregate features based on the IANA IPFIX (IP Flow Information Export Entities) [4] specification. The resulting flows are then stored in *CSV* files. Three different config files were created to extract the flows. One for 2-tuple bidirectional flows (TA feature vector), one for 1-tuple unidirectional source flows (AGM_S feature vector) and one for 1-tuple unidirectional destination flows (AGM_D feature vector).

For each dataset we end up with 3 *CSVs* files containing TA, AGM_S and AGM_D feature-vectors.

3.5.2 Preprocessing and Labeling

In the next step, unwanted flows are removed and labels are assigned. This only has to be done for the 2-tuple TA dataset. The AGM datasets are only used as part of the *multi-key* feature-vector and never as a stand alone dataset.

The following preprocessing is applied to the datasets:

- IDS2017: The dataset does not need any further preprocessing.
- MAWI: As described on the MAWI website, the data suffers from *ICMP ANT probing* [31]. Thus, flows where data is sent/received by the top 2 IP addresses that send/receive the most ICMP traffic are removed.

After preprocessing, the labels are assigned using custom labeling scripts.

3.5.3 Train and Test Datasets

After the flows are labeled, the 2-tuple dataset is split into a train and a test set where 70% is used as train data and 30% is used as test data. The data is split in a stratified manner. The MAWI dataset contains significantly more flows than the IDS2017 dataset. Using the whole dataset would be infeasible for the scope of this thesis, training time and computational complexity would be too high. Therefore, MAWI dataset is downsampled to 500.000 flows before splitting. This is also done in a stratified manner so that the original class distribution stays the same.

Some statistical information of the datasets are given in Tables 3.5 and 3.6. As can be seen, the datasets are heavily imbalanced and skewed towards the *Normal* (benign) class. The MAWI dataset consists of 75% *Normal* traffic. In the IDS2017 dataset as much as 99.7% is non-malicious traffic.

	Datasets		
Class	Original	Train	Test
Normal	407813 (99.7%)	285468 (99.7%)	122345 (99.7%)
Infiltration	1098 (0.3%)	769 (0.3%)	329 (0.3%)
Botnet	163 (<0.1%)	114 (<0.1%)	49 (<0.1%)
DDoS	57 (<0.1%)	40 (<0.1%)	17 (<0.1%)
Portscan	26 (<0.1%)	18 (<0.1%)	8 (<0.1%)
Web Attack	23 (<0.1%)	16 (<0.1%)	7 (<0.1%)
Brute Force	11 (<0.1%)	8 (<0.1%)	3 (<0.1%)
Total	409191	286433	122758

Table 3.5: IDS2017 dataset class distributions

	Datasets		
Class	Original	Train	Test
Normal	372687 (74.5%)	260881 (74.5%)	111806 (74.5%)
Multipoint	78180 (15.6%)	54726 (15.6%)	23454 (15.6%)
TCP Scan	26077 (5.2%)	18254 (5.2%)	7823 (5.2%)
UDP Scan	21234 (4.2%)	14864 (4.2%)	6370 (4.2%)
ICMP Scan	1217 (0.2%)	852 (0.2%)	365 (0.2%)
HTTP	414 (0.1%)	290 (0.1%)	124 (0.1%)
Alpha	164 (<0.1%)	115 (<0.1%)	49 (<0.1%)
Other	12 (<0.1%)	8 (<0.1%)	4 (<0.1%)
DoS	10 (<0.1%)	7 (<0.1%)	3 (<0.1%)
Tunneling	5 (<0.1%)	3 (<0.1%)	2 (<0.1%)
Total	500000	350000	150000

Table 3.6: MAWI dataset class distributions

3.5.4 Creating Feature Representations

After the data is split into train and test sets, we can create the needed feature representations.

A random forest ensemble with halving grid search and cross-validation is used as a baseline model for evaluation. We want to evaluate our models using 2-tuple flows and *multi-key* flows. Using the merging strategy described in Chapter 3.2 we can combine the 2-tuple key train and test set with the 1-tuple source and 1-tuple destination datasets resulting in two new *multi-key* train and test sets. These datasets contain the full *TA* and *AGM* features. In Chapter 3.2.2 we showed the relevant features for training the random forest model. These features are directly selected in the training script, no additional feature vector datasets have to be generated. These approaches are not directly comparable since the random forest model is trained with many aggregated flow features and the image representations are created from 3 individual packet features. However, the features used to train the RF model are all based on the same 3 packet features that are also used for the image based methods. Thus, both methods use the same information. The following Table **??** shows the main differences of the undelying data of the different feature represenations.

In Chapter 3.2.4 we discussed the problems that occur when dealing with large image datasets and how they can be addressed. We use the algorithms described in Chapter 3.2.3 and create *TFRecord* datasets of each feature representation. Similar to the feature vector representations, the TUWpic-mkA and TUWpic-mkB representations first need to create a *multi-key* dataset. This is done in-memory during the image dataset generation.

Ealier, we described how few-shot learning can be used to deal with imbalanced datasets.

Feat.	Type	Features	Sequ. Pres.	Compl. Flow
TA FV	aggregated flow	32	no	yes
Multi-Key FV	aggregated flow	64	no	yes
FlowPic	per packet	2	no	yes
TUWpicA	per packet	3	yes	no
TUWpicB	per packet	3	yes	no
TUWpic-mkA	per packet	3	yes	no
TUWpic-mkB	per packet	3	yes	no

Table 3.7: Packet/Flow information preserved in the different feature representations. Type indicates if data is represented per packet or as aggregated flow statistics. Sequ. Pres. refers to preservation of sequentiality or time characteristics. Compl. Flow indicates if the complete flow (all packets) were used to derive the features or create the feature representations.

In order to evaluate the performance of few-shot learning, we create additional image datasets containing reduced number of samples. The assumption is, that when using only a small amount of samples from each class, we can create embeddings that capture the distribution of the original dataset well enough to successfully classify the test data. New samples don't have to match the training embeddings exactly, they just have to be nearby. Undersampling is used to limit the number of samples in each class to at most 40 samples. Then image datasets for these new datasets are created. Few-shot learning is only investigated in multiclass classification tasks.

In total, we create 29 sub-datasets for training and testing for each of IDS2017 and MAWI. Where 4 datasets contain feature vectors and 25 datasets are used to store image representations. An overview of the generated datasets is given in Table 3.8. In the next step we show how these datasets are used to train the models.

3.5.5 Model Training

After the different training and test dataset are created, the models can be trained. The baseline RF model and a series of different CNN architectures were presented in Chapter 3.4. Training all proposed architectures would be out of scope for this work and we decided to run preliminary tests to find the CNN architecture most suitable for this task.

In order to select the best CNN architecture for our problem, preliminary tests are conducted. For each CNN architecture a model is trained for 50 epochs on the **FlowPic**, **TUWpicA** and **TUWpicB** IDS2017 datasets in a multiclass classification task. To select the best architecture among them, True-Positive (TP), False-Negative (FN) and False-Positive (FP) metrics are used. In Table 3.9 the results of each model and dataset is shown. By calculating the average performance, we found that the best performing model was the adapted LeNet-5 architecture. The best performing feature representation

Name	Task	F. Rep.	Res.	Train/Test
2tuple-train.csv	both	feature vector	-	test
2tuple-test.csv	both	feature vector	-	train
multikey-train.csv	both	feature vector	-	train
multikey-test.csv	both	feature vector	-	test
binary-FlowPic-train.tfrecord	binary	FlowPic	-	train
binary-FlowPic-test.tfrecord	binary	FlowPic	-	test
binary-TUWpicA-train.tfrecord	binary	TUWpicA	-	train
binary-TUWpicA-test.tfrecord	binary	TUWpicA	-	test
binary-TUWpicB-train.tfrecord	binary	TUWpicB	-	train
binary-TUWpicB-test.tfrecord	binary	TUWpicB	-	test
binary-TUWpic-mkA-train.tfrecord	binary	TUWpic-mkA	-	train
binary-TUWpic-mkA-test.tfrecord	binary	TUWpic-mkA	-	test
binary-TUWpic-mkB-train.tfrecord	binary	TUWpic-mkB	-	train
binary-TUWpic-mkB-test.tfrecord	binary	TUWpic-mkB	-	test
multiclass-FlowPic-train.tfrecord	multiclass	FlowPic	-	train
multiclass-FlowPic-test.tfrecord	multiclass	FlowPic	-	test
multiclass-TUWpicA-train.tfrecord	multiclass	TUWpicA	-	train
multiclass-TUWpicA-test.tfrecord	multiclass	TUWpicA	-	test
multiclass-TUWpicB-train.tfrecord	multiclass	TUWpicB	-	train
multiclass-TUWpicB-test.tfrecord	multiclass	TUWpicB	-	test
multiclass-TUWpic-mkA-train.tfrecord	multiclass	TUWpic-mkA	-	train
multiclass-TUWpic-mkA-test.tfrecord	multiclass	TUWpic-mkA	-	test
multiclass-TUWpic-mkB-train.tfrecord	multiclass	TUWpic-mkB	-	train
multiclass-TUWpic-mkB-test.tfrecord	multiclass	TUWpic-mkB	-	test
multiclass-FlowPic-train-few.tfrecord	multiclass	FlowPic	u	train
multiclass-TUWpicA-train-few.tfrecord	multiclass	TUWpicA	u	train
multiclass-TUWpicB-train-few.tfrecord	multiclass	TUWpicB	u	train
multiclass-TUWpic-mkA-train-few.tfrecord	multiclass	TUWpic-mkA	u	train
multiclass-TUWpic-mkB-train-few.tfrecord	multiclass	TUWpic-mkB	u	train

Table 3.8: Training and test sets for each of IDS2017 and MAWI. *Name* is the filename of the dataset. *Task* refers to the classification task, either binary, multiclass or both. *R. Rep.* is the name of the feature representation. *Res.* indicates if a resampling technique is applied (*u = undersampling*) and *Train/Test* refers to the type of dataset, either training or testing.

was **TUWpicA**. The results of the RF classifier are also included for comparison. Based on these preliminary tests, the adapted LeNet-5 architecture is used in all further experiments.

Different training tasks need different model configurations. We store the configurations in *JSON* files, which increases reusability and reproducibility instead of submitting them as command line arguments or hardcoding them into the training script. The following Table 3.10 shows the most relevant parameters and their values in different training configurations. All models are trained using *Adam* optimizer and implement an early stopping mechanism. When using few-shot learning, the batch size is reduced and the number of epochs is increased. Additionally, the early stopping patience is adjusted for the larger number of epochs. The RF based experiments are listed in Table 3.11.

Models that do not use contrastive loss or triplet loss use binary-crossentropy and categorical-crossentropy as loss functions. The trained models are stored in the *H5* file format for later evaluation.

Random Forest training was done on an AMD Ryzen 7 1700 CPU. The CNN models

Architecture	FlowPic TP	FlowPic FN	FlowPic FP	TUWpicA TP	TUWpicA FN	TUWpicA FP	TUWpicB TP	TUWpicB FN	TUWpicB FP	Average TP	Average FN	Average FP
FlowPic	80	332	35	69	343	47	53	359	66	67.3	344.6	49.3
Seq2Img	90	322	34	68	344	28	75	337	70	77.6	334.3	44.0
LeNet5	82	330	27	76	336	39	77	335	34	78.3	333.6	33.3
Vec2Im	84	328	34	76	336	32	62	350	62	74.0	338.0	42.6
MobileNetV3	0	412	0	74	338	50	76	336	75.0	50.0	362.0	41.6
Average	67.2	344.8	26	70.8	339.4	39.2	68.6	343.4	61.4	199	213	9399
										Random Forest		

Table 3.9: Preliminary test results. TP samples of the *Normal* class were removed in order to make the results easier comparable. They would result in very high TP values which makes it less intuitive to compare. Because the TP values of the other classes are kept and FN and FP values of all classes are shown, the TP values of the normal class can be left out with no information loss. The most important information (how many attacks are classified correctly and how many benign sample are classified as attacks) is directly visible.

were trained on Nvidia GTX 1060 6GB and/or Nvidia V100 GPUs. Each datasets yields 17 trained models, 15 of which are CNN models and 2 of which are RF models. With the trained models we can proceed to the last part of the experiment, evaluation.

3.5.6 Evaluation and Metrics

For each model, we compute the performance metrics defined in Chapter 3.3. Each model produces two files, a *JSON* file containing the calculated performance metrics, and a *CSV* file containing ground-truth labels, predicted labels and predicted probabilities.

Evaluation of RF models and crossentropy based CNN models is done by using the train datasets as inputs. The models predict classes which can be directly used to calculate performance metrics. Evaluation of embedding based models is a bit more complex.

In order to evaluate models that use distance metric learning, the following approach is used.

1. the trained model is loaded
2. embeddings for both, training and test datasets are generated (predicted)
3. a K-Nearest-Neighbor (KNN) classifier is trained with the training embeddings
4. the trained KNN is used to predict the class labels of the test embeddings
5. performance metrics are generated based on the KNN results

It intuitively shows that KNN aligns very well with this method. A KNN with $K = 3$ is used for nearest neighbors classification. Using $K = 1$ could lead to false classifications because of outliers, using too many neighbors on the other hand could include classes

		Parameters					
Model	Dataset	Epochs	Batch	LR	ESP	ES-Delta	Img. Channels
Cross.	bin/cat-FlowPic	200	512	0.001	20	0.0005	1
Contr.	bin/cat-FlowPic	200	512	0.001	20	0.0005	1
Triplet	bin/cat-FlowPic	200	512	0.001	20	0.0005	1
Cross.	bin/cat-TUWpicA	200	512	0.001	20	0.0005	1
Contr.	bin/cat-TUWpicA	200	512	0.001	20	0.0005	1
Triplet	bin/cat-TUWpicA	200	512	0.001	20	0.0005	1
Cross.	bin/cat-TUWpicB	200	512	0.001	20	0.0005	1
Contr.	bin/cat-TUWpicB	200	512	0.001	20	0.0005	1
Triplet	bin/cat-TUWpicB	200	512	0.001	20	0.0005	1
Cross.	bin/cat-TUWpic-mkA	200	512	0.001	20	0.0005	3
Contr.	bin/cat-TUWpic-mkA	200	512	0.001	20	0.0005	3
Triplet	bin/cat-TUWpic-mkA	200	512	0.001	20	0.0005	3
Cross.	bin/cat-TUWpic-mkB	200	512	0.001	20	0.0005	3
Contr.	bin/cat-TUWpic-mkB	200	512	0.001	20	0.0005	3
Triplet	bin/cat-TUWpic-mkB	200	512	0.001	20	0.0005	3
Contr.	cat-FlowPic-few-shot	1000	64	0.001	200	0.0005	1
Triplet	cat-FlowPic-few-shot	1000	64	0.001	200	0.0005	1
Contr.	cat-TUWpicA-few-shot	1000	64	0.001	200	0.0005	1
Triplet	cat-TUWpicA-few-shot	1000	64	0.001	200	0.0005	1
Contr.	cat-TUWpicB-few-shot	1000	64	0.001	200	0.0005	1
Triplet	cat-TUWpicB-few-shot	1000	64	0.001	200	0.0005	1
Contr.	cat-TUWpic-mkA-few-shot	1000	64	0.001	200	0.0005	3
Triplet	cat-TUWpic-mkA-few-shot	1000	64	0.001	200	0.0005	3
Contr.	cat-TUWpic-mkB-few-shot	1000	64	0.001	200	0.0005	3
Triplet	cat-TUWpic-mkB-few-shot	1000	64	0.001	200	0.0005	3

Table 3.10: Training and test sets for the different classification approaches. The CNN based approaches are applied to each of IDS2017 and MAWI. Model indicates the type of DL model that was used. ESP (Early Stopping Patience) is the number of epochs with no improvement over ES-Delta (Early Stopping Min-Delta) after which the training is stopped.

			Parameters		
Model	Dataset	Task	CV	Ensemble	GridSearch
RF	ta	bin/cat	5-fold	100	Halving
RF	mk	bin/cat	5-fold	100	Halving

Table 3.11: List of experiments based on RF models. Dataset refers to the type of feature-vector that was uesd to train. CV indicates if cross-validation was applied. Ensemble is the number of decision trees used in the RF model, and GridSearch indicates if and what type of GridSearch is applied.

from far apart samples. The euclidean distance was used as distance metric in the KNN classifier.

CHAPTER 4

Results and Discussion

In this chapter we present the data obtained from the experiments, interpret them and discuss their meaning in regards to the research questions formulated in Chapter 1

4.1 Results

Several Neural Network models with different combinations of feature representations and network architectures were trained. In addition, a Random Forest model used as a baseline for performance evaluation is also trained. We use the obtained performance data and compare it to help us answer the following questions "How does the novel feature representation TUWpic compare to state of the art 2D-flow representations?", "Are siamese networks a viable solution for dealing with imbalanced data?", "Can few-shot learning be used to lower training time and resource demands while maintaining the same level of performance as full training?" and "What are the limitations and implications of using real or synthetic data for training and evaluation?".

As described in Chapter 3, each model was trained on a train dataset and evaluated on a test dataset. We group the results logically by dataset (MAWI or IDS2017) and classification task (binary or multiclass). Tables 4.1,4.3,4.2 and 4.4 show the performance metrics obtained from test dataset evaluation. The first two columns indicate the combination of network architecture and feature representation. The "MODEL" column contains the used ML and DL models:

- RF: Random Forest baseline model
- CROSS: NN model using binary or categorical crossentropy
- CONTR: Siamese model using contrastive loss
- TRIPLE: Siamese model using triplet loss

- CONTR (fs): Siamese model using contrastive loss and few-shot learning
- TRIPLE (fs): Siamese model using triplet loss and few-shot learning

whereas the "FEAT." column contains the used feature representation:

- MK: multi-key feature vector (vector based)
- TA: 2-tuple key feature vector (vector based)
- FP: FlowPic (image based)
- A: TUWpicA (image based)
- B: TUWpicB (image based)
- mkA: TUWpic-mkA (image based)
- mkB: TUWpic-mkB (image based)

The remainder of the columns contain the observed performance metrics as defined in Chapter 3.3. We sort the tables by F1 score since the metric is available across all results and robust in terms of imbalance. Macro averaging was used in the multiclass cases to obtain a single value. Macro averaging is more resistant to imbalanced data than other forms of averaging since it treats the score of each class with the same importance.

Figures 4.14.24.3 and 4.4 visualize the performance for each model-representation pair per performance metrics as bar charts. The sorting of the bars is kept the same across all subplots and is based on the F1 score.

Figures 4.5,4.6,4.7 and 4.8 show the performance of feature representations as swarm plots for each performance metric. Each figure represents one performance metric with the corresponding values on the y-axis. The x-axis is split into sections based on the different feature representations. Within each section, dots show the performance of models using this feature representation. Using these figures we can easily compare different feature representations against each other. The same approach is used to compare models against each other. Figures 4.9,4.10,4.11 and 4.12 show the performance of models as swarm plots. Instead of feature representations, the x-axis is split into sections based on different loss functions. Within each section, dots show the performance of models using this loss function.

Looking at these plots some interesting insights can be gained. The bar plots show a clear difference between *multi-key* based approaches and approaches where datasets without *multi-key* data was used. As expected, the FPR scores behave inverse to the other performance metrics. Showing low values for *multi-key* based approaches. The bar plots also demonstate well why accuracy is not a good metric when applied to imbalanced data. Especially in the binary classification tasks, there is very little difference between

the outcomes. The swarm plots also give some interesting per-model insights. For both types of swarm plots, feature-representation and loss representation based, we can see clustering happen in binary classification tasks. The performance difference between *multi-key* and non-*multi-key* based approaches seems to be especially strong in these cases. Swarm plots of multiclass classification tasks show that the individual models are scattered across the metric range.

Based on the reported values we can now find objective answers to the stated research questions. The metrics show no indication that the proposed image based feature representations TUWpicA and TUWpicB significantly improve performance over the state-of-the-art representation flowpic. Only in combination with multi-key based data, the proposed representations TUWpic-mkA and TUWpic-mkB consistently perform better than flowpic. When we only look at results that are not using a *multi-key* based approach, the data is not so clear. In binary classification tasks, most image representations outperform the baseline approach. However, in multiclass classification tasks, the RF model achieves better scores than most image based representations. Using Siamese networks does not significantly improve performance over traditional CNNs using crossentropy loss. The top perfoming models vary greatly in each evaluation group. The best perfoming model overall was the baseline random forest model, leading the evaluation groups in 3 out of 4 cases. Few-shot learning based approaches consistently performed worse than most of the other approaches. Only when used in combination with multi-key based feature representations and only using the MAWI dataset, few-shot learning managed to outperform some other approaches. It showed that approaches tend to behave the same regardless of if they are using synthetic data (IDS2017) or real world data (MAWI). Approaches that perform well in synthetic data show similar performance in real world data. In the binary classification task, the top perfoming model of the MAWI dataset achieves a higher F1 score than the top perfoming model of the IDS2017 dataset. In the multiclass scenario this behaviour swaps and the top perfoming MAWI model falls behind the top performing IDS2017 model.

In this section we have shown the experimental results and presented objective answers to the research questions. In the next section we interpret our findings and try to find reasons for the outcome.

model	feat.	F1	Prec.	Rec.	AP	FPR	AUC	Acc.
contr	mkB	0.978	0.997	0.96	0.916	0.04	0.96	1.0
triplet	mkB	0.977	0.999	0.956	0.911	0.044	0.956	1.0
cross	mkA	0.973	0.992	0.955	0.896	0.045	0.955	1.0
rf	mk	0.973	0.986	0.96	0.894	0.04	0.96	1.0
triplet	mkA	0.966	0.992	0.943	0.872	0.057	0.943	1.0
contr	mkA	0.966	0.996	0.939	0.872	0.061	0.939	1.0
cross	mkB	0.964	0.998	0.935	0.867	0.065	0.935	1.0
cross	fp	0.684	0.84	0.627	0.176	0.373	0.627	0.997
contr	B	0.681	0.841	0.623	0.172	0.377	0.623	0.997
contr	fp	0.68	0.853	0.621	0.174	0.379	0.621	0.997
cross	A	0.677	0.866	0.617	0.175	0.383	0.617	0.997
triplet	A	0.674	0.864	0.615	0.171	0.385	0.615	0.997
cross	B	0.674	0.861	0.615	0.169	0.385	0.615	0.997
contr	A	0.67	0.861	0.611	0.164	0.389	0.611	0.997
triplet	fp	0.654	0.883	0.597	0.152	0.403	0.597	0.997
triplet	B	0.647	0.868	0.592	0.139	0.408	0.592	0.997
rf	ta	0.505	0.512	0.731	0.015	0.269	0.731	0.928

Table 4.1: IDS2017 binary test dataset evaluation (sorted by F1)

model	feat.	F1	Prec.	Rec.	AP	FPR	AUC	Acc.
rf	mk	0.996	0.997	0.995	0.99	0.005	0.995	0.997
cross	mkB	0.991	0.992	0.989	0.977	0.011	0.989	0.993
contr	mkB	0.99	0.991	0.988	0.974	0.012	0.988	0.992
cross	mkA	0.989	0.99	0.987	0.972	0.013	0.987	0.991
contr	mkA	0.988	0.989	0.987	0.97	0.013	0.987	0.991
triplet	mkB	0.979	0.982	0.976	0.948	0.024	0.976	0.984
triplet	mkA	0.964	0.966	0.963	0.911	0.037	0.963	0.973
cross	B	0.651	0.793	0.632	0.409	0.368	0.632	0.799
cross	fp	0.646	0.771	0.628	0.396	0.372	0.628	0.794
contr	B	0.636	0.756	0.62	0.383	0.38	0.62	0.788
contr	fp	0.63	0.775	0.616	0.385	0.384	0.616	0.79
triplet	fp	0.627	0.767	0.613	0.379	0.387	0.613	0.788
cross	A	0.609	0.728	0.599	0.354	0.401	0.599	0.776
triplet	B	0.607	0.799	0.599	0.374	0.401	0.599	0.787
rf	ta	0.582	0.633	0.671	0.349	0.329	0.671	0.594
contr	A	0.566	0.733	0.571	0.328	0.429	0.571	0.769
triplet	A	0.561	0.727	0.567	0.323	0.433	0.567	0.767

Table 4.2: MAWI binary test dataset evaluation (sorted by F1)

model	feat.	F1	Prec.	Rec.	FPR	AUC	Acc.
rf	mk	0.863	0.988	0.792	0.013	0.998	1.0
triplet	mkA	0.752	0.938	0.676	0.016	0.814	1.0
cross	mkA	0.736	0.921	0.67	0.016	0.944	1.0
rf	ta	0.686	0.708	0.769	0.077	0.968	0.906
contr	mkB	0.641	0.856	0.592	0.014	0.76	1.0
triplet	mkB	0.603	0.738	0.538	0.013	0.745	1.0
contr	mkA	0.603	0.671	0.565	0.013	0.78	1.0
contr	fp	0.574	0.758	0.526	0.108	0.746	0.997
contr	B	0.571	0.78	0.512	0.109	0.724	0.997
cross	fp	0.566	0.698	0.517	0.11	0.904	0.997
cross	A	0.559	0.677	0.508	0.109	0.853	0.997
triplet	A	0.553	0.68	0.51	0.111	0.723	0.997
triplet	fp	0.513	0.633	0.5	0.115	0.74	0.997
contr	A	0.51	0.749	0.451	0.11	0.68	0.997
cross	mkB	0.493	0.713	0.433	0.029	0.865	0.999
triplet	B	0.476	0.625	0.448	0.115	0.744	0.997
cross	B	0.449	0.547	0.404	0.112	0.742	0.997
contr (fs)	mkB	0.152	0.162	0.683	0.071	0.815	0.686
contr (fs)	fp	0.147	0.163	0.88	0.104	0.948	0.61
triplet (fs)	fp	0.132	0.15	0.851	0.104	0.929	0.693
contr (fs)	mkA	0.125	0.156	0.747	0.093	0.852	0.54
triplet (fs)	mkA	0.121	0.148	0.76	0.074	0.863	0.632
contr (fs)	A	0.113	0.154	0.689	0.108	0.82	0.477
triplet (fs)	mkB	0.109	0.148	0.7	0.082	0.826	0.521
contr (fs)	B	0.109	0.153	0.64	0.111	0.79	0.449
triplet (fs)	B	0.108	0.15	0.66	0.108	0.806	0.491
triplet (fs)	A	0.105	0.147	0.746	0.106	0.853	0.513

Table 4.3: IDS2017 multiclass test dataset evaluation (sorted by F1)

model	feat.	F1	Prec.	Rec.	FPR	AUC	Acc.
rf	mk	0.72	0.836	0.685	0.002	0.927	0.989
cross	mkB	0.577	0.62	0.55	0.002	0.873	0.992
contr	mkB	0.538	0.581	0.517	0.002	0.758	0.992
cross	mkA	0.533	0.558	0.516	0.002	0.82	0.992
contr	mkA	0.516	0.529	0.507	0.003	0.739	0.99
triplet	mkB	0.509	0.573	0.485	0.005	0.761	0.983
triplet	mkA	0.493	0.557	0.475	0.007	0.744	0.973
cross	B	0.311	0.412	0.28	0.075	0.743	0.786
rf	ta	0.301	0.284	0.485	0.046	0.841	0.479
contr	B	0.295	0.467	0.267	0.082	0.629	0.778
triplet	B	0.291	0.441	0.267	0.082	0.654	0.78
triplet (fs)	mkB	0.264	0.279	0.544	0.054	0.746	0.562
contr (fs)	mkA	0.26	0.267	0.522	0.055	0.741	0.534
contr (fs)	mkB	0.258	0.26	0.561	0.053	0.756	0.576
cross	A	0.249	0.423	0.217	0.086	0.739	0.767
triplet (fs)	mkA	0.247	0.257	0.559	0.062	0.755	0.492
cross	fp	0.23	0.37	0.201	0.076	0.663	0.781
triplet	A	0.226	0.435	0.198	0.092	0.595	0.761
contr	A	0.223	0.446	0.195	0.092	0.585	0.761
contr	fp	0.216	0.375	0.183	0.08	0.583	0.778
triplet	fp	0.201	0.372	0.174	0.08	0.588	0.776
triplet (fs)	fp	0.177	0.212	0.306	0.081	0.674	0.671
contr (fs)	fp	0.173	0.212	0.354	0.082	0.686	0.669
contr (fs)	A	0.151	0.151	0.419	0.093	0.756	0.397
triplet (fs)	A	0.15	0.15	0.383	0.093	0.758	0.397
triplet (fs)	B	0.13	0.148	0.337	0.088	0.733	0.273
contr (fs)	B	0.127	0.148	0.428	0.09	0.741	0.252

Table 4.4: MAWI multiclass test dataset evaluation (sorted by F1)

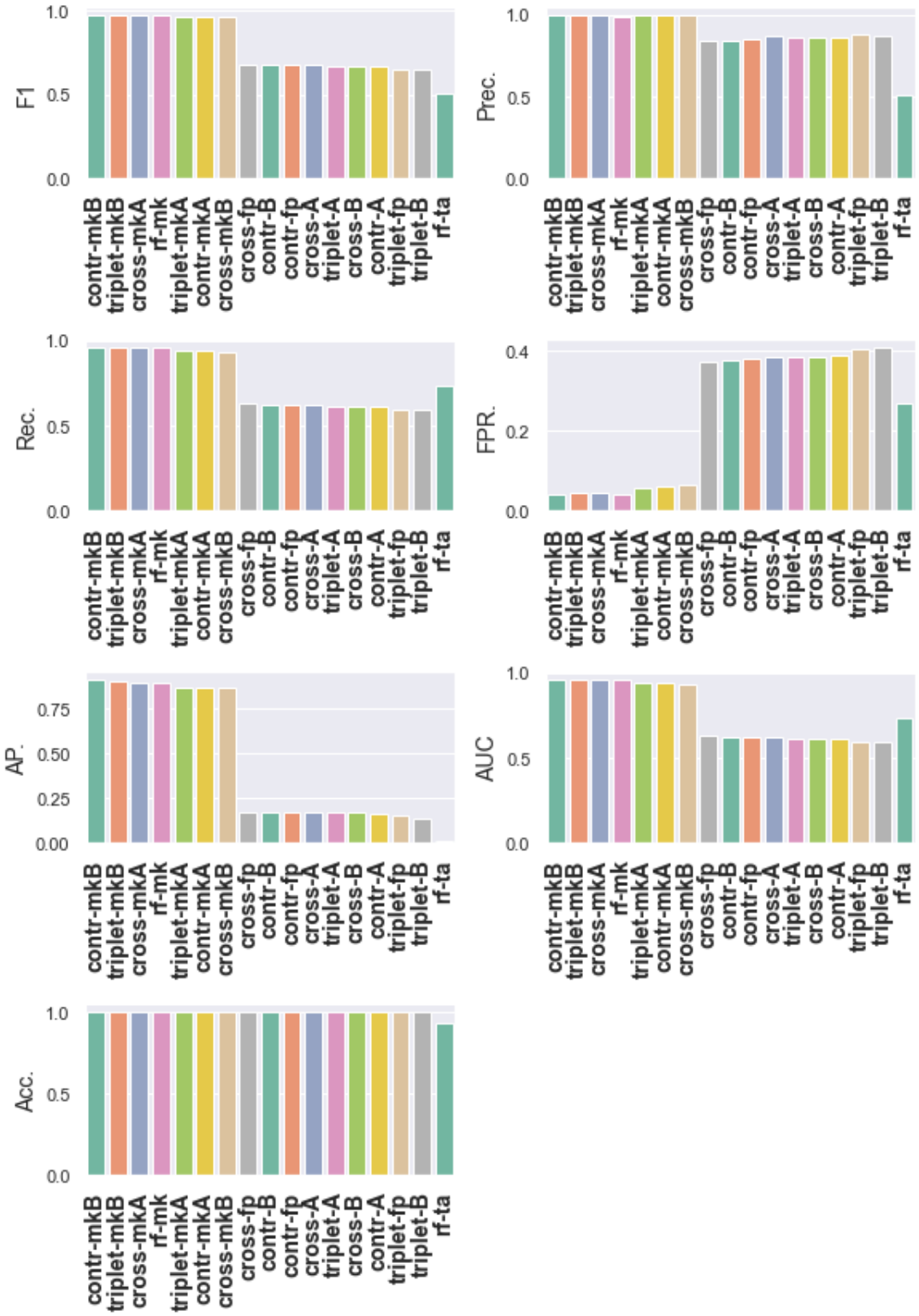


Figure 4.1: IDS2017 performance metrics per model/feature-representation (Binary)

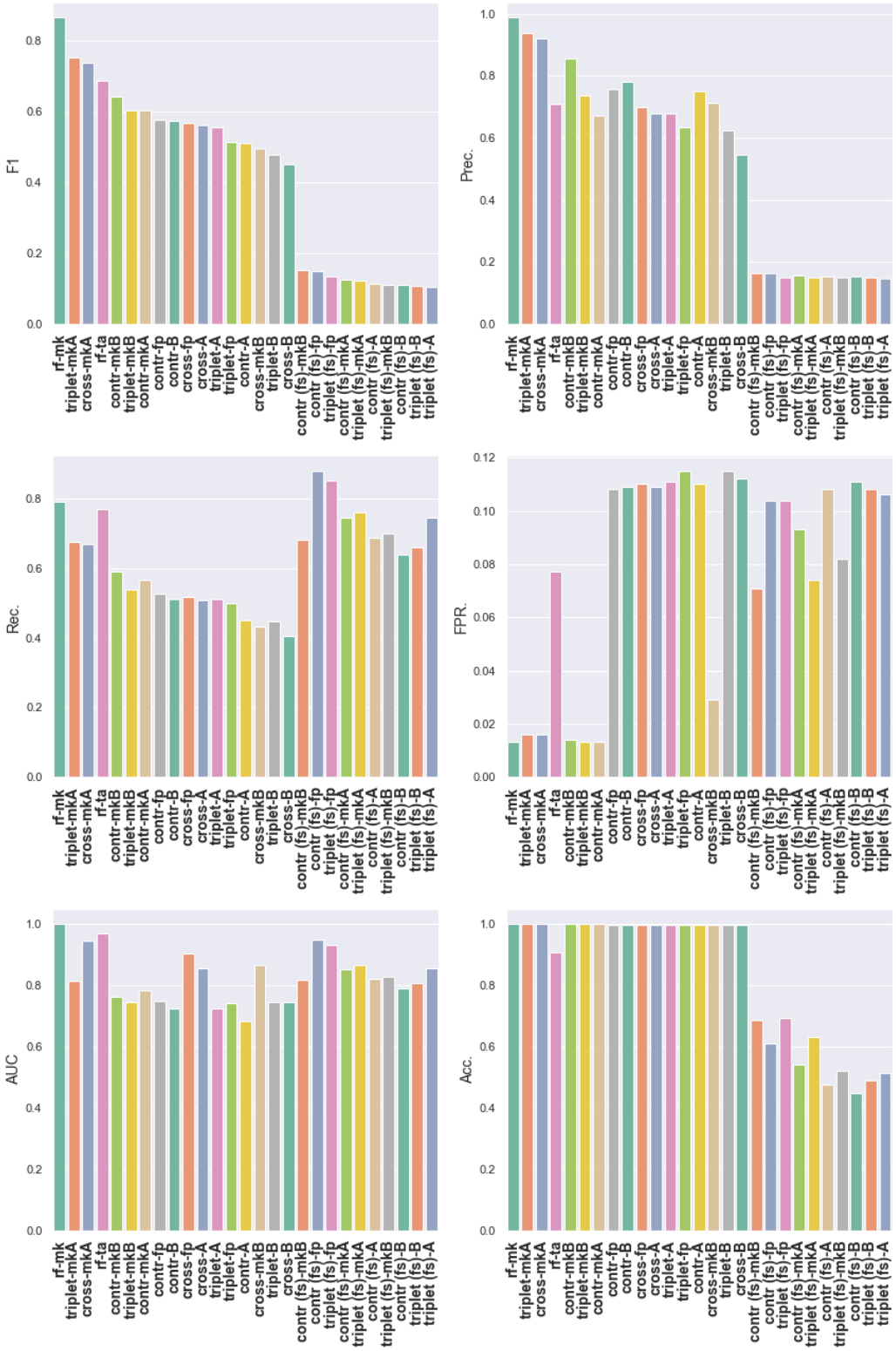


Figure 4.2: IDS2017 performance metrics per model/feature-representation (Multiclass)

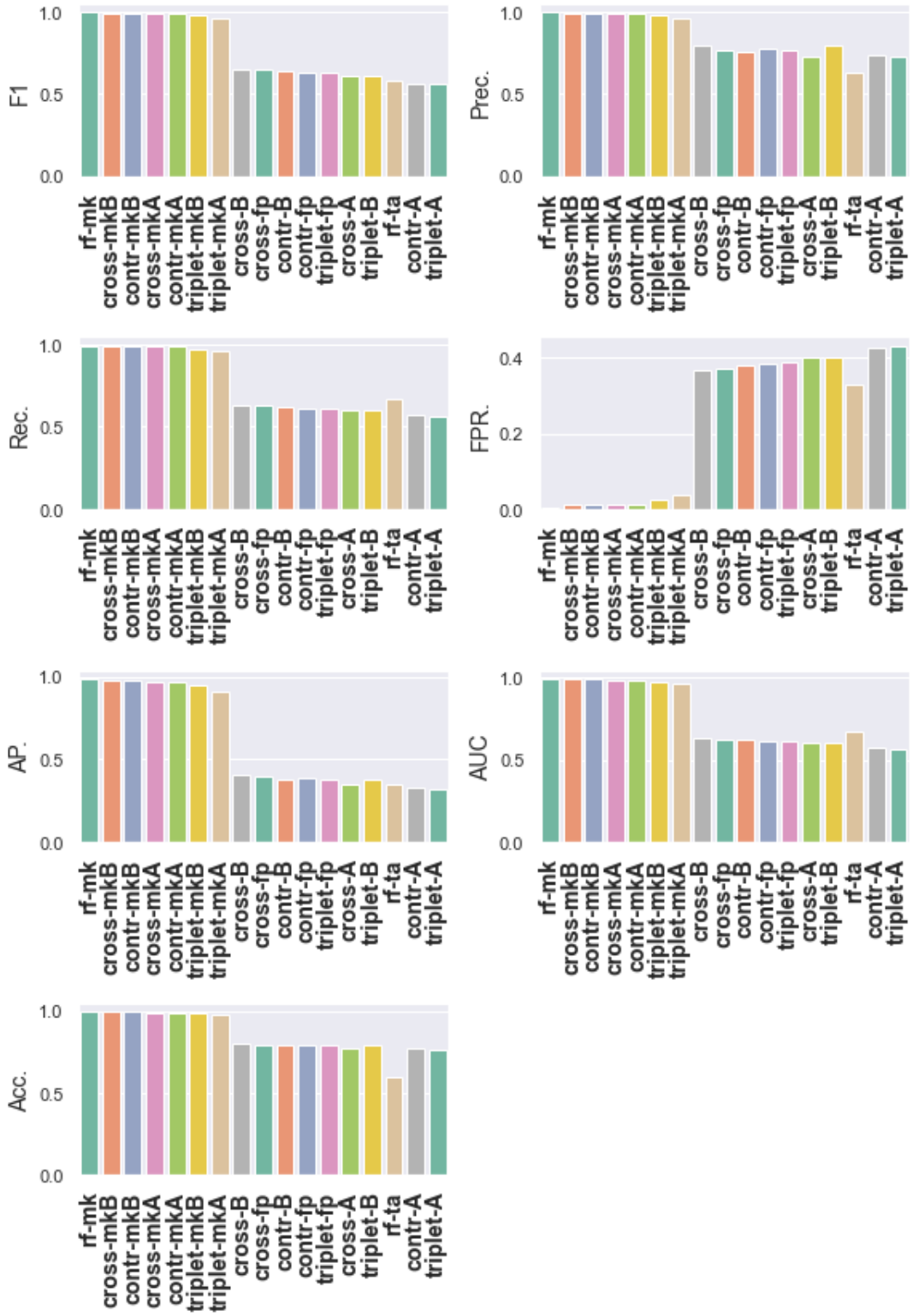


Figure 4.3: MAWI performance metrics per model/feature-representation (Binary)

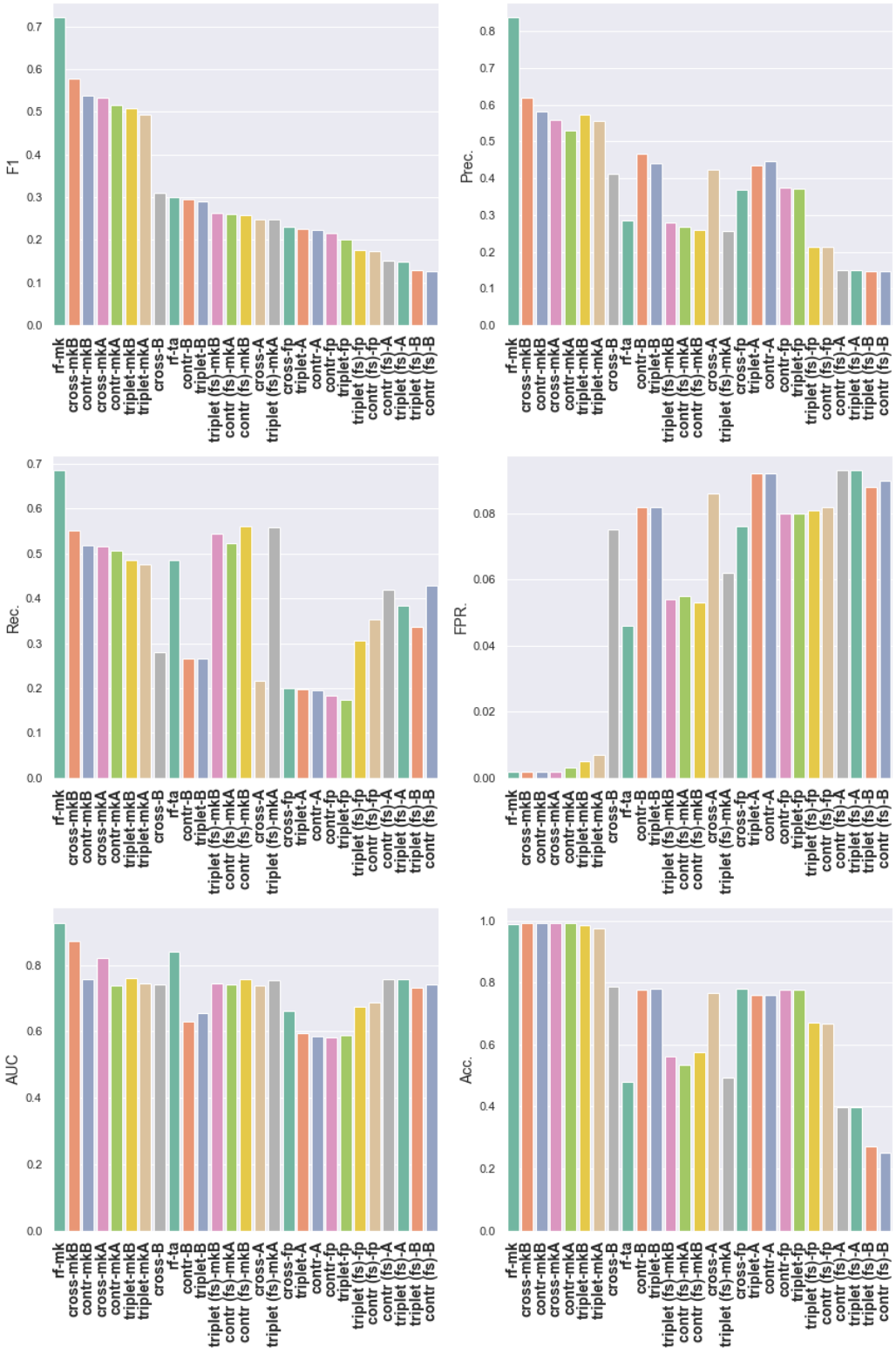


Figure 4.4: MAWI performance metrics per model/feature-representation (Multiclass)

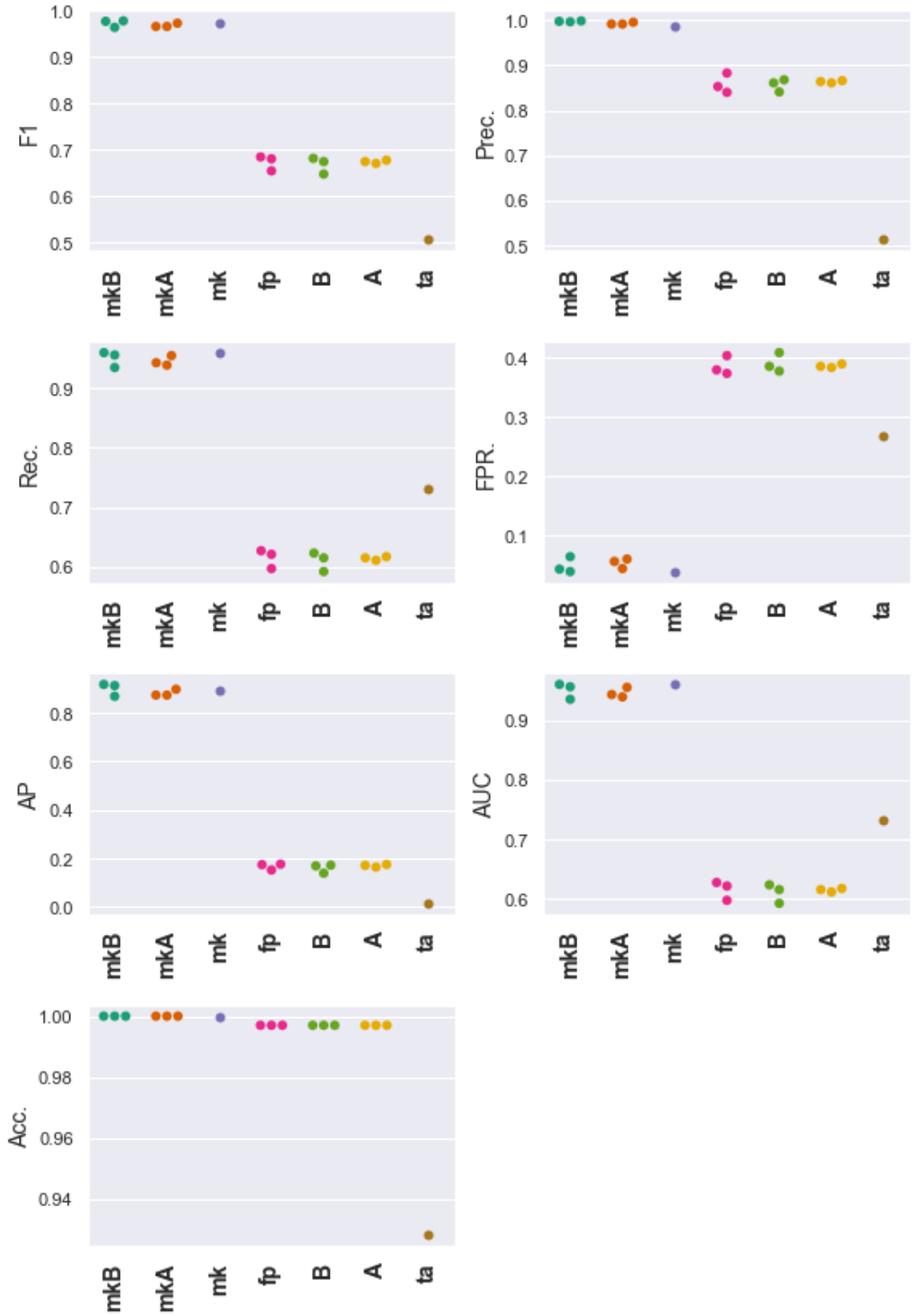


Figure 4.5: IDS2017 performance metrics per feature-representation (Binary). Each dot represents a model trained with the corresponding feature representation listed on the x-axis. The y-axis shows the performance metric for each model.

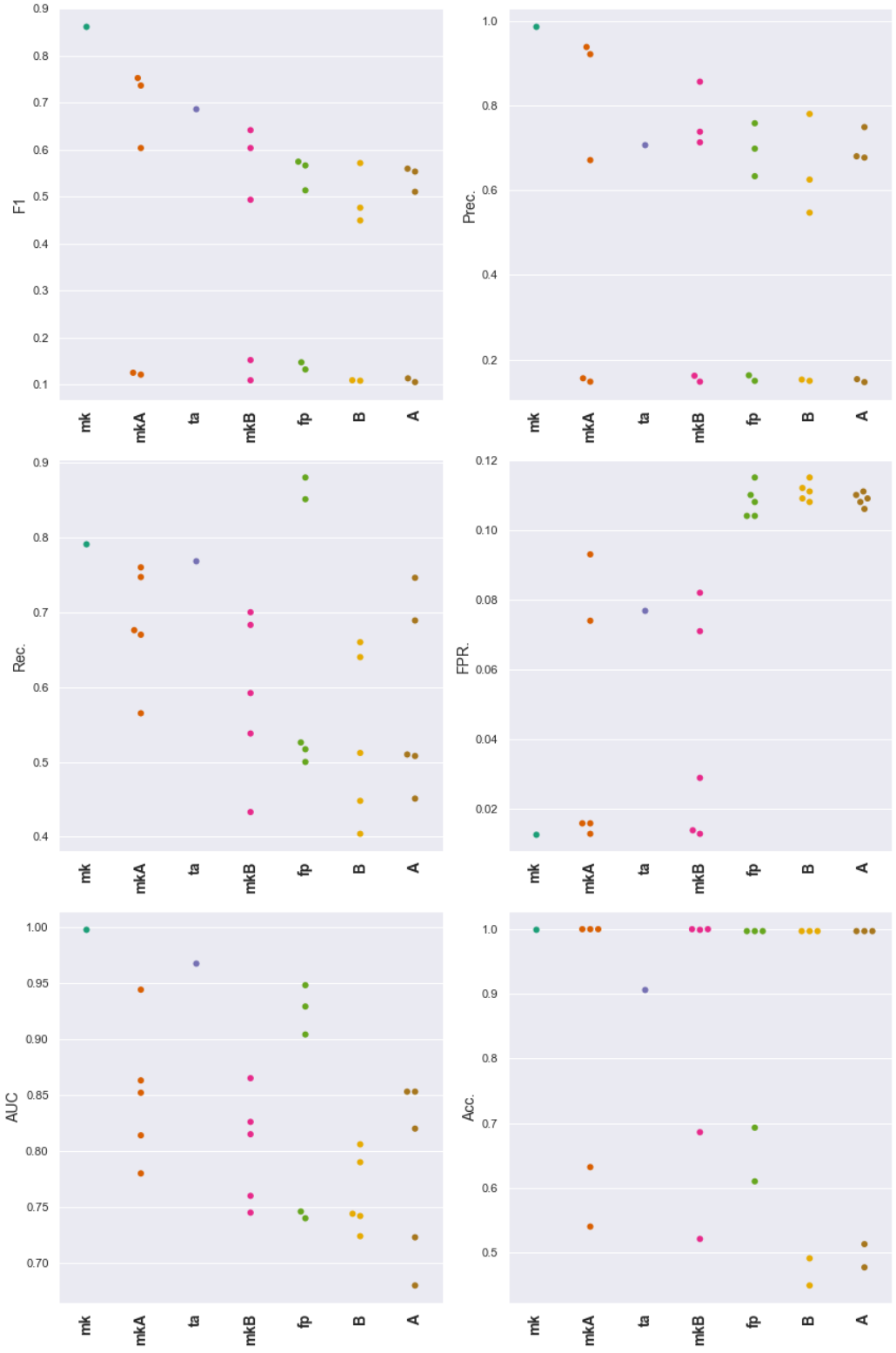


Figure 4.6: IDS2017 performance metrics per feature-representation (Multiclass). Each dot represents a model trained with the corresponding feature representation listed on the x-axis. The y-axis shows the performance metric for each model.

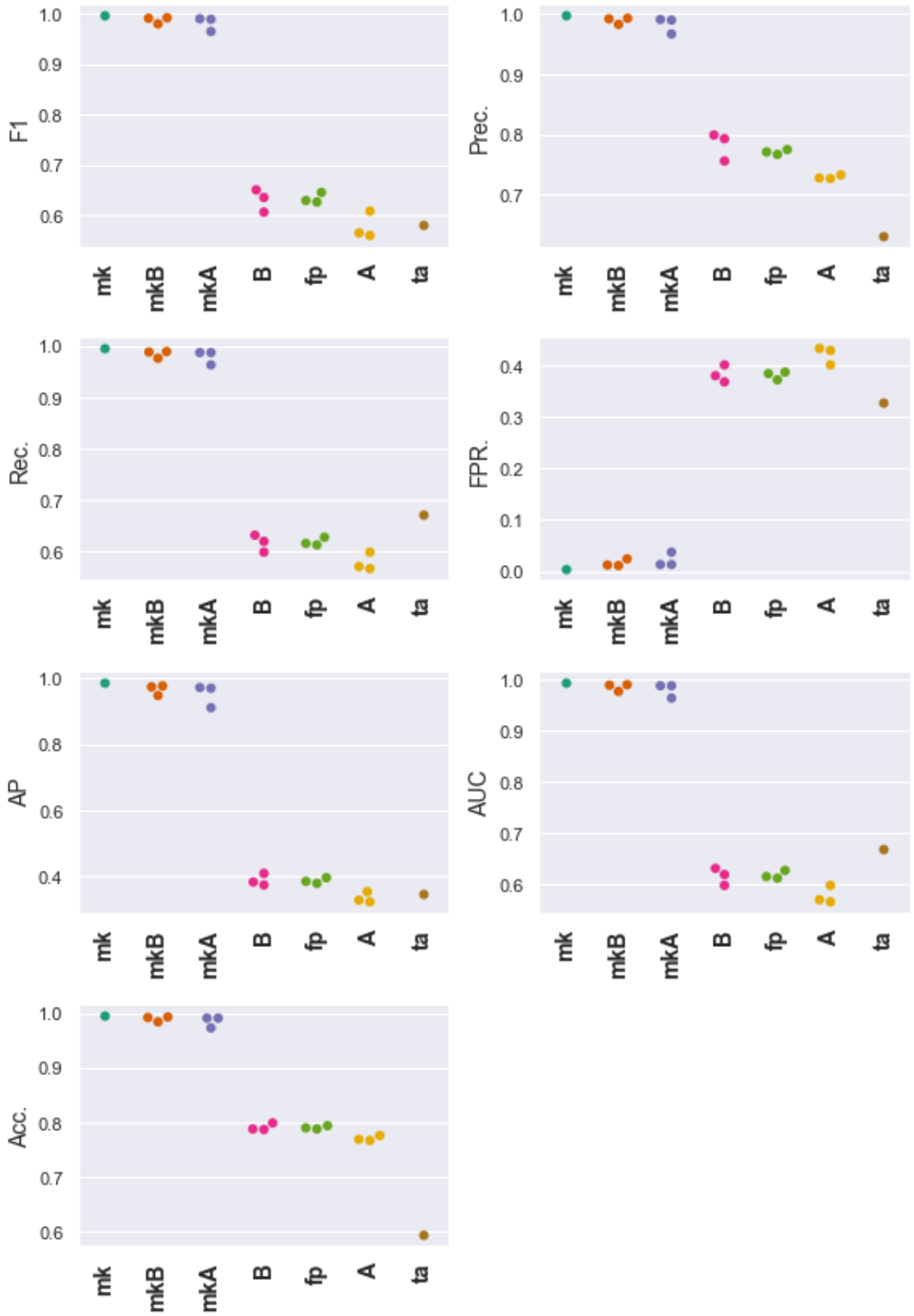


Figure 4.7: MAWI performance metrics per feature-representation (Binary).

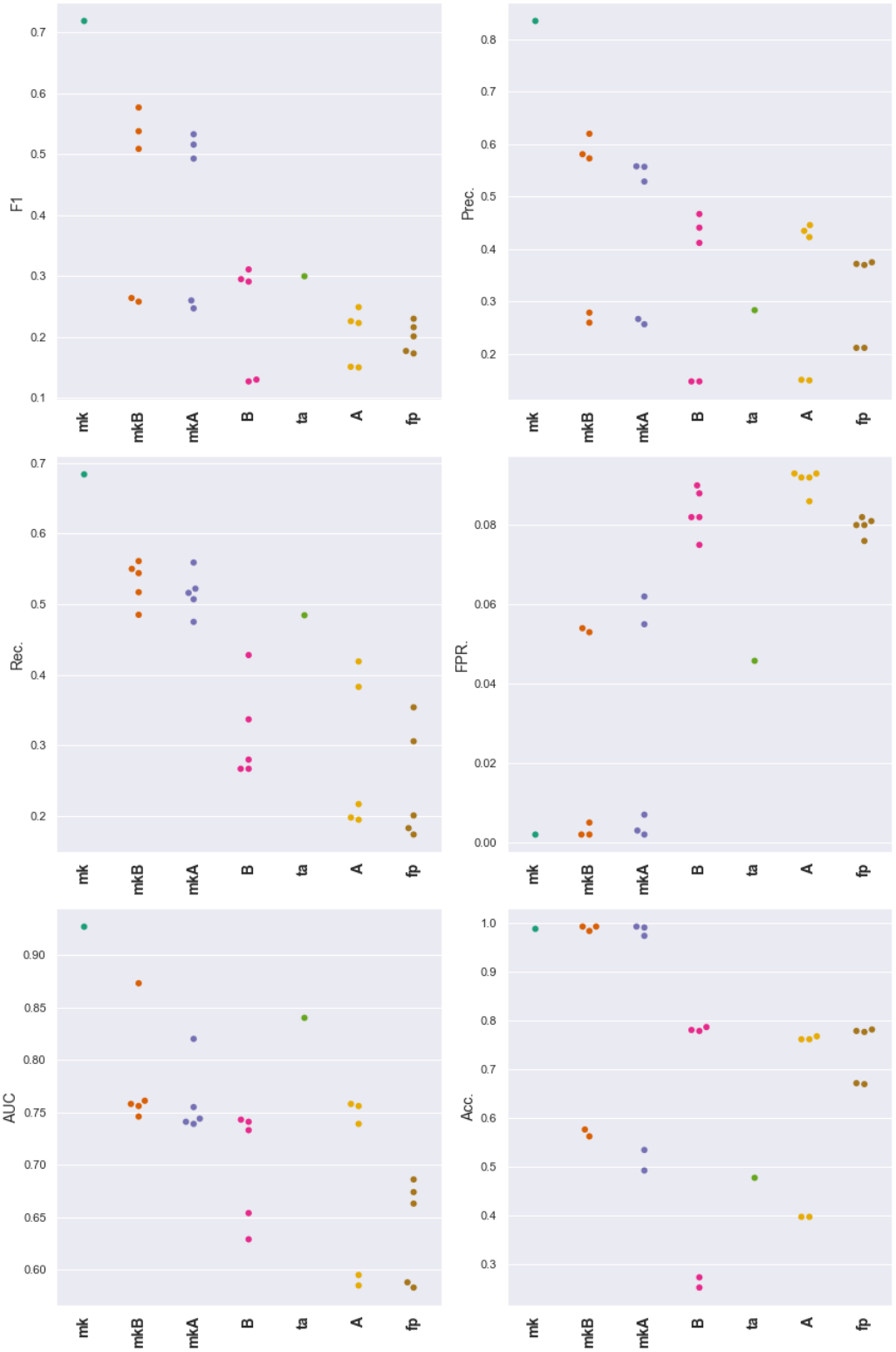


Figure 4.8: MAWI performance metrics per feature-representation (Multiclass)

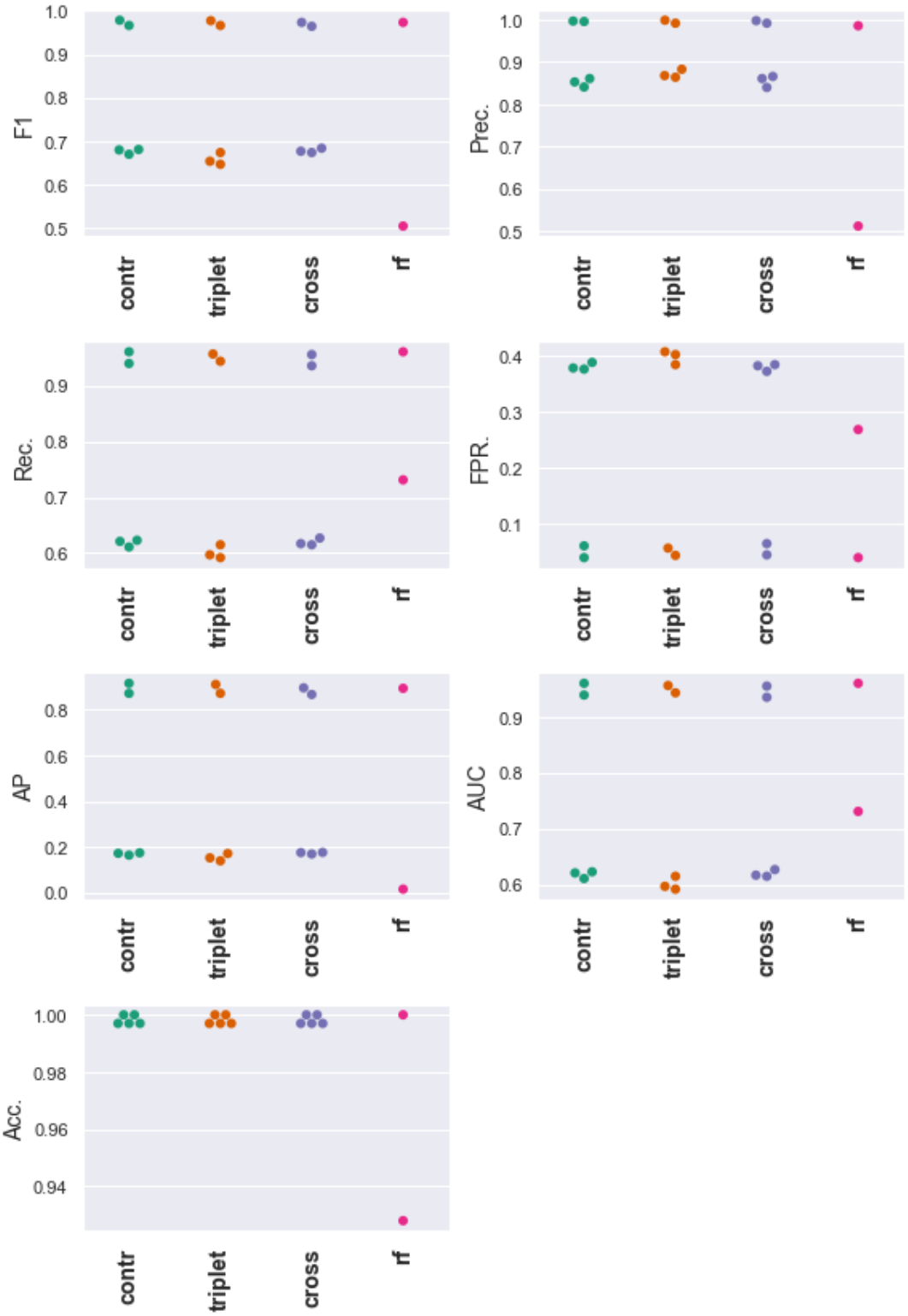


Figure 4.9: IDS2017 performance metrics per model (Binary). Each dot represents a model trained with the corresponding loss function (crossentropy, contrastive, triplet) listed on the x-axis. The y-axis shows the performance metric for each model.

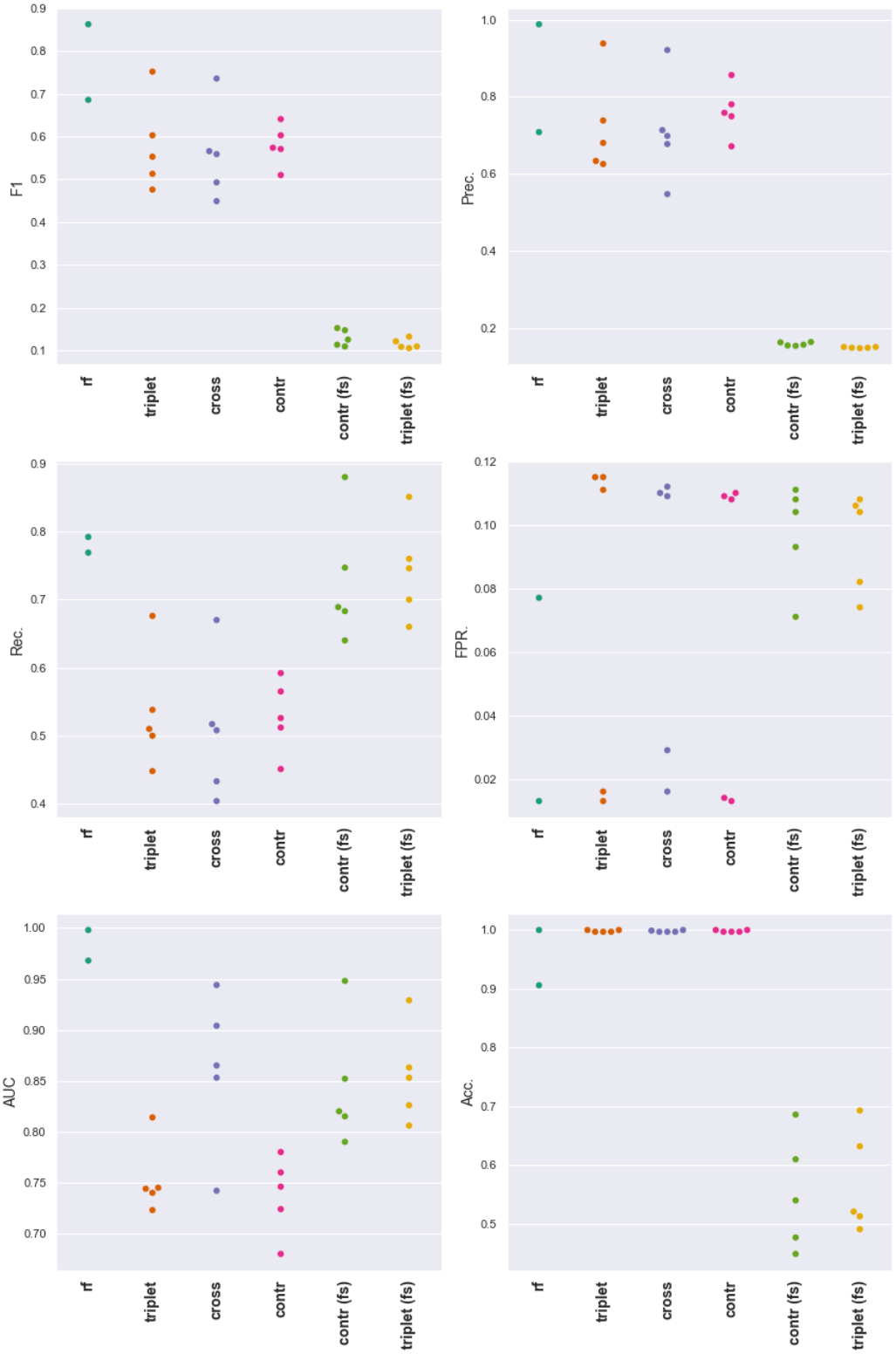


Figure 4.10: IDS2017 performance metrics per model (Multiclass). Each dot represents a model trained with the corresponding loss function (crossentropy, contrastive, triplet) listed on the x-axis. The y-axis shows the performance metric for each model.

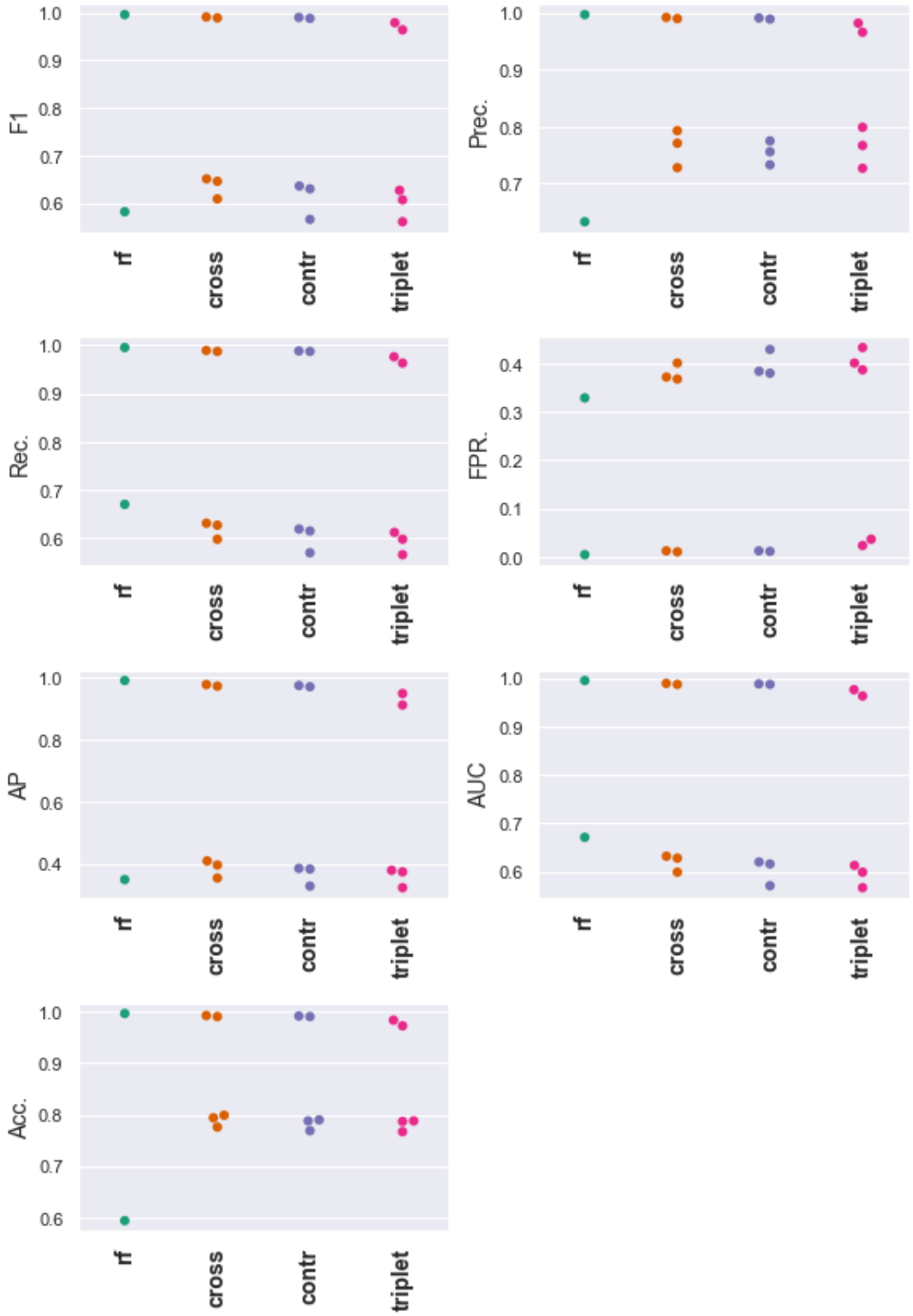


Figure 4.11: MAWI performance metrics per model (Binary)

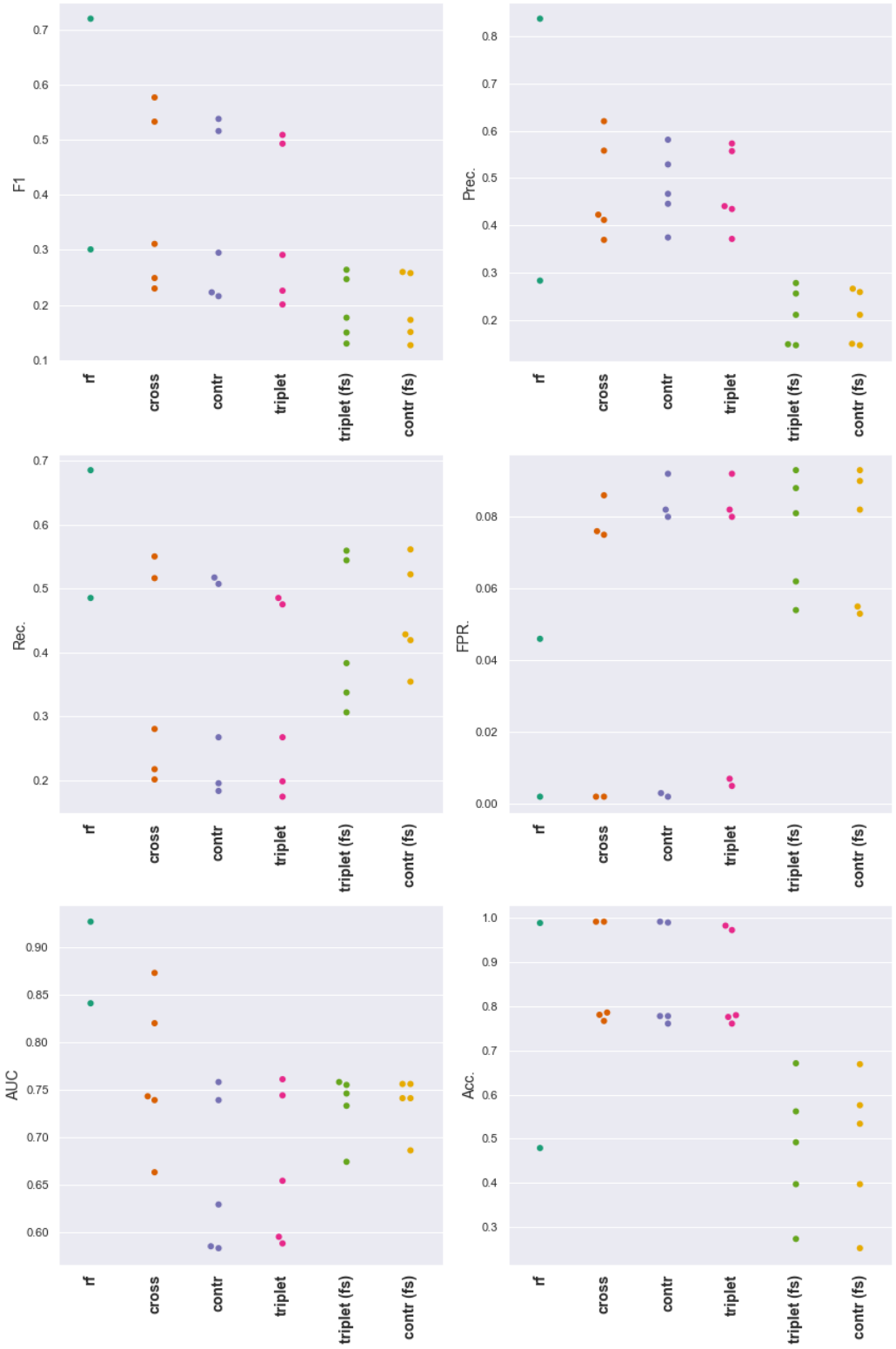


Figure 4.12: MAWI performance metrics per model (Multiclass)

4.2 Discussion

Looking at the results, the main takeaway here is that approaches based on our proposed feature representations could not outperform the baseline multi-key approach. We need to keep in mind that feature vectors and images don't cover the same data. In our case the feature vectors contain aggregated statistics while the images can be seen as time series representation of a flow containing per packet information. Although the images are based on similar data as the feature vectors, they perform worse. There are several reasons why this could be:

1. The image representation itself might store too little information or too much information is lost in the transformation process. This is also directly connected to the parameters used to create TUWpics. Aggregated feature-vectors use all of the flows information, TUWpic (in this thesis) only uses the first 128 packets.

2. The used model/architecture is not capable of learning all the features that are theoretically stored in the representation. This might be a valid reason since the used model is rather shallow. A very deep model on the other hand performed worse in the preliminary test than the LeNet-5 model.

3. The architecture is fine but its hyperparameters are wrongly tuned. No hyperparameter tuning was done in this thesis, all hyperparameters are typical values used across the literature and selected by manual testing. A perfectly tuned model can increase the detection rate.

4. The representations contain no additional information that is useful for this classification task, which is not also contained in the feature vectors.

If enough time and resources are available, points 2 and 3 could be checked with hyperparameter tuning and dynamic model generation using a grid-search like approach. Points 1 and 4 on the other hand are hard to evaluate since they impose basically the same questions that we are trying to answer in this thesis.

A promising result is that both datasets show similar results. This is good because it indicates that we can use synthetic datasets and still achieve high performance in real world data. It also strengthens the approach proposed by Sharafaldin et al. [81] in which they show how good IDS datasets can be constructed. But we need to keep in mind that the MAWI dataset might not be labeled perfectly and performance could be different if we knew the actual ground truth data of MAWI captures.

It also can be seen that few-shot learning scenarios tend to perform not as good as the other scenarios. A reason for why few-shot learning did not achieve better results, could be in the nature of network traffic itself. Benign network traffic covers a wide field of different types of communications. VoIP, games, streams, messengers and Email are just a selection of different areas of application where network traffic is generated. By

undersampling the data to only a fraction of the original amount, a lot of this information is lost. It is possible that there is benign traffic that looks similar to malicious traffic. If the network was not trained with such a sample, there is no way for it to know which class it belongs to. This problem needs to be addressed when samples are selected during undersampling. In the best case, undersampling chooses samples that cover a great variety of different traffic types. In the worst case, all the traffic is of the same kind. If only samples of video streaming traffic are chosen for example, then the model will perform poorly. Multiple approaches can be taken to reduce the problem. Data with a lot of variance could be clustered and then samples from each cluster can be taken. Additional Principal Component Analysis (PCA) could be used to identify the biggest variance in the features.

Looking at the swarm plots we can see that in multiclass scenarios, the points are much more scattered than in binary scenarios. In the binary case they are often grouped in two clusters, the multiclass cases show no clear pattern. This might be related to the problem we addressed using few-shot learning. Attack classes might look similar in some cases and are therefore hard to distinguish from each other. Especially when dealing with attacks that belong to the same attack family. But also attack from different attack families can look similar, especially when there is a lot of benign traffic in the same flow and attack packets only make up a small part of the total packets. In the binary scenario this has no impact because they are all treated as the same attack. In the multiclass scenario, this might lead to attacks misclassified as other attacks. While you could argue that it still was detected, the task was to find the correct class and not just a class.

When comparing different algorithms and feature representations one also has to consider the time and the resources that are needed to train these models or create and manage the feature representations. As well as the overall ease of use. Training a RF classifier with HalvingGridSearch and 5-fold CV takes less than 5 minutes on a Ryzen 1700 system. At the same time training a deep learning model takes over 1 hour on a data center grade GPU (V100) and more than 3 hours on a consumer GPU (GTX 1060 6GB). When distance metric learning is used, also the prediction part of deep learning based models becomes a time consuming task. If hyperparameter tuning is added to the deep learning model, time and resource requirements can get very expensive. Additionally, using images as feature representations comes with the problem of data management and storage. Feature vectors on the other hand have to be designed by hand.

While having failed to increase performance over vector based approaches, the proposed multi-key based image representations show better performance than the state-of-the-art competitor. In the next chapter we will summarize our findings and draw a conclusion.

CHAPTER 5

Conclusions

We have proposed multiple novel image feature representations for network traffic analysis and intrusion detection. Two of which are based on the concepts of *multi-key* feature vectors. By combining multiple feature vectors that cover different types of network communications (application level, host level, network level) a more comprehensive view of the network traffic behavior can be created. To achieve this a 2-tuple flow representing application level communication and 2 1-tuple flows representing host (attacker and victim) level communication are combined.

In addition to these 4 image representations called TUWpicA, TUWpicB, TUWpic-mkA and TUWpic-mkB, a state of the art image feature representation called FlowPic is used as comparison.

We trained 3 different types of CNNs models using these image feature representations. A traditional CNN with classification outputs for binary and multiclass scenarios and two CNNs using distance metric learning based on contrastive loss and triplet loss. The distance metric based CNNs create embeddings which are then used to train a KNN classifier. Additionally we use a random forest model as a baseline model for performance comparison. Lastly we also use distance metric learning to investigate the possibility of few-shot learning.

It was shown that *multi-key* based feature representations significantly outperform feature representations that only use a single flow including FlowPic. It was also shown that performance is largely dependent on the type of flow and less on the actual form of representation. In 3 out of 4 test scenarios, the random forest classifier performed best. A clear winner between DL algorithms could not be found. Results largely depend on the type of classification task (binary vs. multiclass), dataset and feature-representation. Considering the long training time and additional effort to create image datasets, there is no benefit in using CNNs or image based representations over traditional machine learning algorithms and feature vectors. There is no evidence that the distance metric

based CNN models could improve performance on the imbalanced dataset in comparison to a traditional CNN using crossentropy loss. Few-shot learning has significantly worse results than the other algorithms, independent of the dataset.

All experiments were done using two datasets. The synthetic IDS2017 dataset and the MAWI dataset, consisting of real traffic captured on a backbone network. In the binary classification task, performance of the best models in each dataset is very similar, with the MAWI dataset getting a slightly better F1 score. In the multiclass classification task the performance in both datasets drops significantly. The best model trained on the IDS2017 dataset achieves an F1 score of 0.978 in the binary case, and only 0.863 in the multiclass case. For the MAWI dataset the data looks similar. The best model in the binary scenario achieves a F1 score of 0.996, while the best performing model in the multiclass case only achieves 0.72. As the binary results show, the performance is not purely based on the type of data but also on the classification task.

With the gained knowledge and results we can answer the research questions formulated in Chapter 1 based on the IDS2017 and MAWI dataset.

How does the novel feature representation TUWpic compare to state of the art 2D-flow representations with regard to standard ML performance metrics?

Using data from a single flow, TUWpic based image representations show no significant improvement over FlowPic. In comparison to FlowPic, TUWpic also supports *multi-key* based representations, which significantly outperformed FlowPic.

Can siamese networks be used to improve performance in comparison to traditional DL models and therefore offer an algorithmic approach of dealing with imbalanced data? Do siamese networks offer any advantage over CNNs? Which architecture is best suited for NTA?

All CNN types investigated in this thesis (with the exception of few-shot based approaches) showed similar results. In our experiments, large differences in the results are mainly related to the used feature representation and not to the type of CNN used.

Can few-shot learning be used to lower training time and resource demands while maintaining the same level of performance as full training?

In our experiments, few-shot learning performed significantly worse in every tested scenario and is no alternative to full training.

Can synthetic datasets be used for selecting ML/deep-learning approaches and feature representations without having to change the architecture or parameters when the same process is applied to real world data?

Performance of different methods showed to be constant across the IDS2017 and MAWI datasets. This leads to the conclusion that synthetic datasets can be used to develop ID models. Using synthetic dataset yields many benefits like precise control over attack types and scenarios, attack and benign traffic distribution, or data integrity.

During the course of the thesis some points of improvement and possible future work were considered. Few-shot learning fails to cover the variances in normal learning. Using more sophisticated sampling strategies based on clustering and PCA should result in better performing models.

In this thesis the datasets were used as they are. No resampling strategies were applied. In recent works GANs are often used to generate new samples [43] for imbalanced datasets. Generating new samples could especially help in multiclass classification tasks to achieve better scores.

The proposed TUWpic feature representation is a dynamic image format. Meaning height, width, feature values and length can be controlled with parameters. Different parameters for different scenarios and dataset might lead to better results. In combination with few-shot learning, which allows for fast training and testing, grid search can be used to evaluate the performance of different hyperparameters applied to the same dataset.

In my opinion, feature-vectors are the best feature-representation at the moment. The long and resource demanding training and preprocessing that is involved in CNN based approaches support this argument. Nonetheless, CNN based approaches already show good results and with improving NN technology, deeper networks, faster preprocessing and better optimization techniques this method might be able to outperform the more traditional approaches in the future.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] Github - cn-tu/go-flows: Flow exporter implementation in go. cn contact: Gv.

[2] Ids 2017 | datasets | research | canadian institute for cybersecurity | unb.

[3] Internet assigned numbers authority.

[4] Ip flow information export (ipfix) entities.

[5] Keras: the python deep learning api.

[6] Mawi samplepointf-f may 1st.

[7] Mawilab samplepointf-f may 1st.

[8] Nsl-kdd | datasets | research | canadian institute for cybersecurity | unb.

[9] Pcap capture file format.

[10] Protocol buffers | google developers.

[11] Scikit-learn average precision score — scikit-learn 1.0.2 documentation.

[12] Scikit-learn confusion matrix — scikit-learn 1.0.2 documentation.

[13] Scikit-learn halvinggridsearch — scikit-learn 1.0.2 documentation.

[14] Tfrecord and tf.train.example | tensorflow core.

[15] Welcome to python.org.

[16] M. Abbasi, A. Shahraki, and A. Taherkordi. Deep learning for network traffic monitoring and analysis (ntma): A survey. *Computer Communications*, 170:19–41, 3 2021.

[17] M. H. Ali, B. A. D. A. Mohammed, A. Ismail, and M. F. Zolkipli. A new intrusion detection system based on fast learning network and particle swarm optimization. *IEEE Access*, 6:20255–20261, 3 2018.

[18] R. Alshammari and A. N. Zincir-Heywood. Investigating two different approaches for encrypted traffic classification. *Proceedings - 6th Annual Conference on Privacy, Security and Trust, PST 2008*, pages 156–166, 2008.

[19] A. Alshamsi and T. Saito. A technical comparison of ipsec and ssl. *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, 2:395–398, 2005.

[20] P. Bedi, N. Gupta, and V. Jindal. Siam-ids: Handling class imbalance problem in intrusion detection systems using siamese neural network. *Procedia Computer Science*, 171:780–789, 1 2020.

[21] P. Bedi, N. Gupta, and V. Jindal. I-siamids: an improved siam-ids for handling class imbalance in network-based intrusion detection systems. *Applied Intelligence*, 51:1133–1151, 2 2021.

[22] L. Bottou. Stochastic gradient learning in neural networks. *undefined*, 1991.

[23] J. BROMLEY, J. W. BENTZ, L. BOTTOU, I. GUYON, Y. LECUN, C. MOORE, E. SÄCKINGER, and R. SHAH. Signature verification using a "siamese" time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 07:669–688, 8 1993.

[24] V. Carela-Español, P. Barlet-Ros, A. Cabellos-Aparicio, and J. Solé-Pareta. Analysis of the impact of sampling on netflow traffic classification. *Computer Networks*, 55:1083–1099, 4 2011.

[25] Z. Chen, K. He, J. Li, and Y. Geng. Seq2img: A sequence-to-image based approach towards ip traffic classification using convolutional neural networks. volume 2018-Janua, pages 1271–1276. IEEE, 12 2017.

[26] K. Cho, K. Mitsuya, and A. Kato. Traffic data repository at the wide project.

[27] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde. Analyzing android encrypted network traffic to identify user actions. *IEEE Transactions on Information Forensics and Security*, 11:114–125, 1 2016.

[28] A. D'Alconzo, I. Drago, A. Morichetta, M. Mellia, and P. Casas. A survey on big data for network traffic monitoring and analysis. *IEEE Transactions on Network and Service Management*, 16:800–813, 9 2019.

[29] X. Dong and J. Shen. Triplet loss in siamese network for object tracking. 2018.

[30] H. F. Eid, A. Darwish, A. E. Hassanien, and A. Abraham. Principle components analysis and support vector machine based intrusion detection system. *Proceedings of the 2010 10th International Conference on Intelligent Systems Design and Applications, ISDA'10*, pages 363–367, 2010.

[31] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda. Mawilab : Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. 2010.

[32] I. S. R. Group. 2021 isrg annual report, 2021.

[33] M. He, X. Wang, J. Zhou, Y. Xi, L. Jin, and X. Wang. Deep-feature-based autoencoder network for few-shot malicious traffic detection. 2021, 2021.

[34] Y. He and W. Li. Image-based encrypted traffic classification with convolution neural networks. pages 271–278. IEEE, 7 2020.

[35] A. Hermans, L. Beyer, and B. Leibe. In defense of the triplet loss for person re-identification. 3 2017.

[36] A. Howard, M. Sandler, B. Chen, W. Wang, L. C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, Q. Le, and H. Adam. Searching for mobilenetv3. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-October:1314–1324, 5 2019.

[37] S. C. Hsiao, D. Y. Kao, Z. Y. Liu, and R. Tso. Malware image classification using one-shot learning with siamese networks. *Procedia Computer Science*, 159:1863–1871, 1 2019.

[38] Y. Hu, D. M. Chiu, and J. C. Lui. Application identification based on network behavioral profiles. *IEEE International Workshop on Quality of Service, IWQoS*, pages 219–228, 2008.

[39] S. ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5:185–196, 6 1993.

[40] F. Iglesias, D. C. Ferreira, G. Vormayr, M. Bachl, and T. Zseby. Ntarc: A data model for the systematic review of network traffic analysis research. *Applied Sciences 2020, Vol. 10, Page 4307*, 10:4307, 6 2020.

[41] F. Iglesias and T. Zseby. Time-activity footprints in ip traffic. 2016.

[42] F. Iglesias and T. Zseby. Pattern discovery in internet background radiation. *IEEE Transactions on Big Data*, 5:467–480, 7 2017.

[43] A. S. Iliyasu and H. Deng. Semi-supervised encrypted traffic classification with deep convolutional generative adversarial networks. *IEEE Access*, 8:118–126, 2020.

[44] N. Inoue and K. Goto. Semi-supervised contrastive learning with generalized contrastive loss and its application to speaker recognition; semi-supervised contrastive learning with generalized contrastive loss and its application to speaker recognition, 2020.

[45] M. Joshi and T. H. Hadi. A review of network traffic analysis and prediction techniques. 7 2015.

[46] A. S. Khan, Z. Ahmad, J. Abdullah, and F. Ahmad. A spectrogram image-based network anomaly detection system using deep convolutional neural network. *IEEE Access*, 9:87079–87093, 2021.

[47] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2:1–22, 12 2019.

[48] G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. 11 2015.

[49] I. Kotuliak, P. Rybár, and P. Trúchly. Performance comparison of ipsec and tls based vpn technologies. *ICETA 2011 - 9th IEEE International Conference on Emerging eLearning Technologies and Applications, Proceedings*, pages 217–221, 2011.

[50] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60:84–90, 5 2017.

[51] S. Kumar, J. Turner, and J. Williams. Advanced algorithms for fast and scalable deep packet inspection. *ANCS 2006 - Proceedings of the 2006 ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 81–92, 2006.

[52] D. Kwon, H. Kim, J. Kim, S. C. Suh, I. Kim, and K. J. Kim. A survey of deep learning-based network anomaly detection. *Cluster Computing*, 22:949–961, 1 2019.

[53] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, 5 2015.

[54] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278–2323, 1998.

[55] J. H. Lee and K. H. Park. Gan-based imbalanced data intrusion detection system. *Personal and Ubiquitous Computing*, 25:121–128, 2 2021.

[56] W. H. Lee, C. S. Lim, and B. N. Noh. Generation of similar traffic using gan for resolving data imbalance, 2020.

[57] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36:16–24, 1 2013.

[58] L. Liu, P. Wang, J. Lin, and L. Liu. Intrusion detection of imbalanced network traffic based on machine learning and deep learning. *IEEE Access*, 9:7550–7563, 2021.

[59] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian. Deep packet: a novel approach for encrypted traffic classification using deep learning. *Soft Computing*, 24:1999–2012, 2020.

[60] F. Meghdouri, F. I. Vazquez, and T. Zseby. Cross-layer profiling of encrypted network data for anomaly detection. *Proceedings - 2020 IEEE 7th International Conference on Data Science and Advanced Analytics, DSAA 2020*, pages 469–478, 2020.

[61] I. Melekhov, J. Kannala, and E. Rahtu. Siamese network features for image matching. *Proceedings - International Conference on Pattern Recognition*, 0:378–383, 1 2016.

[62] A. Moore, D. Zuev, M. Crogan, A. W. Moore, and M. L. Crogan. Discriminators for use in flow-based classification discriminators for use in flow-based classification *. 2005.

[63] S. Moustakidis and P. Karlsson. A novel feature extraction methodology using siamese convolutional neural networks for intrusion detection. *Cybersecurity*, 3:16, 12 2020.

[64] J. Nagi, F. Ducatelle, G. A. D. Caro, D. Cireşan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella. Max-pooling convolutional neural networks for vision-based hand gesture recognition. *2011 IEEE International Conference on Signal and Image Processing Applications, ICSIPA 2011*, pages 342–347, 2011.

[65] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath. Malware images. pages 1–7. ACM Press, 2011.

[66] C. E. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. Activation functions: Comparison of trends in practice and research for deep learning. 11 2018.

[67] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, and J. Aguilar. Towards the deployment of machine learning solutions in network traffic classification: A systematic survey. *IEEE Communications Surveys and Tutorials*, 21:1988–2014, 4 2019.

[68] J.-O. Palacio-Niño and F. Berzal. Evaluation metrics for unsupervised learning algorithms. 5 2019.

[69] E. Papadogiannaki and S. Ioannidis. A survey on encrypted network traffic analysis applications, techniques, and countermeasures. *ACM Computing Surveys*, 54:1–35, 7 2021.

[70] W. Park and S. Ahn. Performance comparison and detection analysis in snort and suricata environment. *Wireless Personal Communications*, 94:241–252, 5 2017.

[71] S. Rezaei and X. Liu. Deep learning for encrypted traffic classification: An overview. *arXiv*, pages 76–81, 2018.

[72] S. Rezaei and X. Liu. How to achieve high classification accuracy with just a few labels: A semi-supervised approach using sampled packets. *arXiv*, 2018.

[73] R. Riad, C. Dancette, J. Karadayi, N. Zeghidour, T. Schatz, and E. Dupoux. Sampling strategies in siamese networks for unsupervised speech representation learning. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2018-September:2658–2662, 4 2018.

[74] S. Ruder. An overview of gradient descent optimization algorithms.

[75] T. Saito and M. Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PLoS ONE*, 10, 3 2015.

[76] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab. A review on machine learning–based approaches for internet traffic classification. *Annales des Telecommunications/Annals of Telecommunications*, 75:673–710, 2020.

[77] W. Samek, T. Wiegand, and K.-R. Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. 8 2017.

[78] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. 3 2015.

[79] S. Sengupta, S. Basak, P. Saikia, S. Paul, V. Tsalavoutis, F. Atiah, V. Ravi, and A. Peters. A review of deep learning with special emphasis on architectures, applications and recent trends. 194:105596, 2020.

[80] T. Shapira and Y. Shavitt. Flowpic: Encrypted internet traffic classification is as easy as image recognition. pages 680–687. IEEE, 4 2019.

[81] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. A detailed analysis of the cicids2017 data set. *Communications in Computer and Information Science*, 977:172–188, 1 2018.

[82] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. 2018.

[83] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. 2018.

[84] M. Sikaroudi, B. Ghojogh, A. Safarpoor, F. Karray, M. Crowley, and H. R. Tizhoosh. Offline versus online triplet mining based on extreme distances of histopathology patches. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12509 LNCS:333–345, 7 2020.

[85] D. Sun, Z. Wu, Y. Wang, Q. Lv, and B. Hu. Risk prediction for imbalanced data in cyber security : A siamese network-based deep learning classification framework. pages 1–8. IEEE, 7 2019.

[86] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic. Robust smartphone app identification via encrypted network traffic analysis. *IEEE Transactions on Information Forensics and Security*, 13:63–78, 1 2018.

[87] N. Thapa, Z. Liu, D. B. KC, B. Gokaraju, and K. Roy. Comparison of machine learning and deep learning models for network intrusion detection systems. *Future Internet*, 12:167, 9 2020.

[88] E. Tjoa and C. Guan. A survey on explainable artificial intelligence (xai): Towards medical xai. *IEEE Transactions on Neural Networks and Learning Systems*, 32:4793–4813, 7 2019.

[89] P. Velan, M. Čermák, P. Čeleda, and M. Drašar. A survey of methods for encrypted traffic classification and analysis. *International Journal of Network Management*, 25:355–374, 9 2015.

[90] D. P. Vinchurkar, A. Reshamwala, and M. Tech. A review of intrusion detection system using neural network and machine learning technique. *Certified International Journal of Engineering Science and Innovative Technology (IJESIT)*, 9001:54, 2008.

[91] L. Vu, C. T. Bui, and Q. U. Nguyen. A deep learning based method for handling imbalanced problem in network traffic classification. volume 2017-Decem, pages 333–339. ACM, 12 2017.

[92] B. Wang and D. Wang. Plant leaves classification: A few-shot learning method based on siamese network. *IEEE Access*, 7:151754–151763, 2019.

[93] W. Wang, Y. Sheng, J. Wang, X. Zeng, X. Ye, Y. Huang, and M. Zhu. Hast-ids: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection. *undefined*, 6:1792–1806, 12 2018.

[94] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng. Malware traffic classification using convolutional neural network for representation learning. *International Conference on Information Networking*, pages 712–717, 4 2017.

[95] X. Wang, Y. Hua, E. Kodirov, G. Hu, and N. M. Robertson. Deep metric learning by online soft mining and class-aware attention. 11 2018.

[96] Y. Wang, G.-Y. Wei, D. Brooks, and J. A. Paulson. Benchmarking tpu, gpu, and cpu platforms for deep learning. 7 2019.

[97] Y. Xiao, C. Xing, T. Zhang, and Z. Zhao. An intrusion detection model based on feature reduction and convolutional neural networks. *IEEE Access*, 7:42210–42219, 2019.

[98] L. Xu, D. Dou, and H. J. Chao. Etcnet: Encrypted traffic classification using siamese convolutional networks. *NAI 2020 - Proceedings of the 2020 Workshop on Network Application Integration/CoDesign*, pages 51–53, 2020.

[99] X. Yang, P. Zhou, and M. Wang. Person reidentification via structural deep metric learning. *IEEE Transactions on Neural Networks and Learning Systems*, 30:2987–2998, 10 2019.

[100] Y. Yang, K. Zheng, B. Wu, Y. Yang, and X. Wang. Network intrusion detection based on supervised adversarial variational auto-encoder with regularization. *undefined*, 8:42169–42184, 2020.

[101] J. Zhang, X. Chen, Y. Xiang, W. Zhou, and J. Wu. Robust network traffic classification. *IEEE/ACM Transactions on Networking*, 23:1257–1270, 8 2015.

[102] X. Zhang, F. Wu, and Z. Li. Application of convolutional neural network to traditional data. *Expert Systems with Applications*, 168:114185, 4 2021.

[103] H. Zhou, Y. Wang, X. Lei, and Y. Liu. A method of improved cnn traffic classification. *Proceedings - 13th International Conference on Computational Intelligence and Security, CIS 2017*, 2018-Janua:177–181, 2018.

[104] M. Zhou, Y. Tanimura, and H. Nakada. One-shot learning using triplet network with knn classifier.