Dominik Wilhelm Mocher, BSc

Remote Traffic Analysis of Smartphone Applications

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Dipl.-Ing. Johannes Feichtner

Univ.-Prof. Dipl.-Ing. Dr.techn. Stefan Mangard

Institute of Applied Information Processing and Communications (IAIK)

Graz, February 2020

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date	Signature

Abstract

With the increasing usage of transport encryption in mobile applications, the identification of smartphone apps based on their produced network traffic becomes more challenging. Recent approaches apply machine learning models on the metadata contained in network traffic to predict the corresponding application. In this thesis, we propose three machine learning approaches to identify applications based on characteristics in the produced network traffic. We exploit similarities of network traffic metadata and natural language text documents to apply the existing text processing model Doc2Vec as well as neural networks. These models are used to find unique patterns in network traffic metadata, which can be used as a fingerprint to identify the source application.

To compare the effectiveness and performance of each model, we developed a prototype and performed a case study with real-world applications. We manually collected network traffic from Android devices and created four evaluation datasets using the extracted metadata. The neural networks were able to identify applications with up to 96 per cent precision and recall from traffic metadata, outperforming approaches based on Doc2Vec. The contextual information of the feature combination does not seem to have an impact on the prediction performance. From the results, we conclude that Doc2Vec is an unsuitable model to identify applications based on the chosen metadata features.

Keywords: app identification, traffic analysis, machine learning, Doc2Vec

Kurzfassung

Mit der zunehmenden Verbreitung von Transportverschlüsselung in mobilen Anwendungen wird die Identifikation von Smartphone-Applikationen zunehmend schwieriger. Neueste Ansätze verwenden maschinelles Lernen, um Anwendungen basierend auf Metadaten im Netzwerkverkehr zu erkennen. In dieser Arbeit werden drei Ansätze zur Identifikation von Anwendungen anhand von Charakteristiken im Netzwerkverkehr dargestellt. Dabei werden Gemeinsamkeiten von natürlicher Sprache und Netzwerkverkehr genutzt, um das für die Textanalyse eingesetzte maschinelle Lernmodell Doc2Vec sowie Neuronale Netzwerke anwenden zu können. Diese Modelle sind im Stande, charakteristische Muster in Netzwerkmetadaten zu finden, die für die Identifikation der Quellanwendung herangezogen werden können. Um die Effektivität und die Erkennungsrate der Modelle zu vergleichen, wurde ein Prototyp entwickelt und eine Fallstudie mit mehreren Anwendungen durchgeführt. Dafür wurde Netzwerkverkehr manuell von Android-Geräten gesammelt, auf deren Basis vier Evaluierungsdatensätze erstellt wurden. Die Neuronalen Netze konnten Applikationen mit bis zu 96 Prozent Genauigkeit anhand der Netzwerkmetadaten identifizieren und damit alle Modelle, die auf Doc2Vec basieren, übertreffen. Durch die Berücksichtigung von Kontextinformationen in den Kombinationen der einzelnen Merkmale konnte keine Veränderung in der Erkennungsrate festgestellt werden. Aus den vorliegenden Ergebnissen ist ersichtlich, dass Doc2Vec nicht zur Identifikation von Anwendungen anhand der ausgewählten Merkmale in den Netzwerkmetadaten geeignet ist.

Stichwörter: Identifikation von Anwendungen, Netzwerkanalyse, maschinelles Lernen, Doc2Vec

Acknowledgements

First of all, I would like to thank Johannes Feichtner for his feedback and guidance during this work. My sincere thanks to Peter Teufl for sharing his expertise in machine learning. I would also like to express my gratitude to my colleagues at IAIK for their encouragement and valuable discussions on the topic. Finally, I thank my friends and family for their support. I am especially grateful to Ursula, Eric, Theres and Wilhelm.

Dominik Mocher

Contents

List of Figures

Chapter 1

Introduction

With smartphones becoming a more and more important part in our everyday life, the amount of online services we use in the form of apps increases steadily as well. A significant amount of people tend to use mobile applications like Whatsapp[1] or Skype[2] for the majority of their daily online communication. It is common to have data available by synchronising files and folders across multiple devices or share media with smartphones of other people. Even collaborative work can be performed using mobile applications. These developments lead to an always-online mentality, resulting in a continuing increase in network traffic produced by each device.

With this effect, the need to strengthen the protection of data during transport and the privacy of the user becomes more important. For applications running on Android or iOS, Google and Apple recommend that applications transmit data in encrypted form using *Transport Layer Security* (TLS).[3] [4] TLS constitutes a defensive mechanism against man-in-the-middle and injection attacks, reducing the risk of leaking possibly sensitive data during transport. For this reason, an attacker cannot easily redirect online banking transmissions, and authoritarian governments cannot monitor messages of citizens without forcing the installation of government-issued certificates. Even locally installed anti-virus programs cannot inspect the network packet payload and have to break transport encryption.

[1] https://www.whatsapp.com/
[2] https://www.skype.com/
[3] https://developer.android.com/training/articles/security-tips
[4] https://developer.apple.com/library/archive/documentation/NetworkingInternetWeb/Conceptual/NetworkingOverview/SecureNetworking/SecureNetworking.html

However, to transport a data packet between network nodes, the routing information has to be readable in each node. As a result, this information cannot be encrypted, and the use of transport encryption does not prevent the exposure of this metadata.

1.1 Motivation

Patterns in metadata can recur in seconds as well as in much larger time frames. In the latter case, the observation of network traffic over such a large time frame can lead to numerous network packets. All of those network packets have to be analysed in order to find potential patterns. Network traffic patterns emerge from characteristics of application program code as well as the used network infrastructure. Since these patterns are often well-distinguishable, they can be used to identify applications and backend services. By combining patterns produced from sufficiently many applications on a device, a fingerprint can be created to recognise the device within a network environment. Although metadata analysis methods can provide a significant amount of sensible information, it does not violate any security principles of transport encryption.

Information gathered in this way can serve a wide range of purposes. For example, services that require high bandwidth or low latency can be prioritised in order to optimise local network infrastructure and increase the service quality. On the other hand, this information can be used to gain an insight on applications installed on a device, allowing to profile users and their surroundings. For instance, if the app of a specific bank is installed, it is likely that the user has an account at this financial institute. On a more detailed classification level, application versions that contain known bugs can be identified remotely, if traffic patterns are distinct between multiple versions. An attacker can make use of this knowledge to target devices running this specific app version. By comparing traffic patterns from a known good app to abnormal application traffic, modified apps containing malware might be detected as well. This information can then be used to inform device owners. Creating awareness to remove such modified applications can prevent further spreading of infections. In the case that known patterns produced by widely used libraries are contained in the network traffic, it can be determined of which application the library is part of.

1.2 Problem

In order to perform meaningful analyses, the availability of an extensive amount of metadata is critical. Additionally, the number of features contained in this metadata is of equal importance. Although single features are useful as a first estimator of an application or service type, purely relying on this estimation is prone to errors. In [10] and [34], the authors base their methods primarily on hostnames of the backend services and the responses of *Domain Name System* (DNS) queries. This solution has several limitations, like the caching of DNS responses implemented in every major *Operating System* (OS). This mechanism allows reusing the result from previous name resolution queries. Due to this reuse, the cached mapping between hostnames and addresses does not necessarily reflect recent changes in the network. This leads to the problem that the mapping on a target device may differ from the response that the analysing machine gets.

Another drawback of relying on the hostname is the distinguishability between applications contacting the same backend service. For example, applications often use an authentication provider like Firebase[5] from Google for user management. All applications sharing an authentication provider would, therefore, contact a domain name belonging to this provider during the login of a user. If the application identification relied solely on the hostname in this traffic, all apps would be classified as a single one with relation to Google. A further scenario is the use of an advertisement framework in multiple applications, which connects to the hostname of the advertisement service. If these apps do not contact additional hostnames, they could be classified as the same application similar to the previous scenario.

To overcome the limitations of the scenarios outlined in these examples, the identification of relevant features in metadata is crucial. The combination of features can determine application-specific patterns in network traffic. In order to capture traffic patterns, a sufficiently large amount of data is vital, as these patterns can span over a vast amount of network packets. This extent of metadata requires a significant amount of computational performance. Evaluating the importance of single features leading to an identification tends to be a complicated task. Depending on the application, one feature can be more relevant than another. For instance, a well-distinguishable packet size might lead to better identification results than a service hostname shared between two applications. On the other hand, the packet size could be similar to a third application.

[5] https://firebase.google.com/docs/auth/

The hostname could be a better indicator in this case. For this reason, it is a complex task to balance the weighting of features leading to a classification decision. Another problem are features, that can change over time, like DNS information or hostnames. In order to avoid pitfalls originating from such unreliable sources, we focus on using generic network packet features directly related to the transmission.

1.3 Our Approach

We believe that the best way for identifying applications based on patterns in network traffic metadata is the use of machine learning methods. This approach draws advantage from the vast quantity of available metadata. This leads to a large corpus of training data, which allows a more precise adaption of the model in the learning phase. Another benefit of this approach is the fact that the detection can be performed in a short time, once the model is trained. Although the training of a machine learning model requires a significant amount of processing time, this task can be performed on a sufficiently powerful workstation. The trained model can be stored for later use and does not require retraining.

Unsupervised machine learning techniques lack the flexibility of correlating patterns to specific targets and tend to create clusters based on similar traffic. These models do not provide the option to define specific training targets like applications, application categories or shared libraries. For this reason, we opted to use supervised machine learning strategies in favour of control over the desired outcome classes, although the preprocessing effort increases due to labelling the data. We assume that the structure of network traffic metadata is similar to documents consisting of structured text. Thus we adapt an approach out of this field in the form of Doc2Vec. Doc2Vec is a supervised machine learning algorithm, tailored to determine the topic of a text document. This algorithm is built on the widely known Word2Vec. Doc2Vec predicts the result based on patterns in the text structure while additionally considering the context information of word combinations and adjacent words. We train one Doc2Vec model to identify applications, while a second one is built to detect predefined application categories. During the training of the first model, we use application names as labels. For the second model, these labels are replaced with categories. In both cases, the assigned labels are considered as document topic.

In order to train both machine learning models on realistic user behaviour, we

capture network traffic directly on the devices during normal use of applications. For gathering traffic, a local *Virtual Private Network* (VPN) server is installed on each device and serves as the capturing point. This approach provides the advantage of correlating names of running applications to the produced network traffic packets, which are then used as application name labels. The collected packet features can be restructured into documents containing single sessions, with each packet representing a single sentence. This restructuring allows us to use Doc2Vec with network data similar to textual documents. We use the first part of the session documents for training the Doc2Vec model in order to adapt the model parameters to network data. The other part is then used to evaluate the classification performance of the trained model on unseen data, which reduces the risk of fitting the model to biased data. We propose two configurations of Doc2Vec, differing in the used learning and prediction strategy as well as two neural network classifiers with different layer architectures. The metric results of all approaches are compared in order to find the most accurate model for detecting smartphone applications.

1.4 Outline

This thesis is divided into several chapters. Chapter 2 gives an overview of necessary background information. We discuss the structure of network traffic in detail, focussing on extracting metadata and gathering additional information. Afterwards, we explain Android-specific network components and introduce the Android VPN application Netguard. In addition, we discuss current supervised machine learning approaches for text processing and emphasise multi-class classification techniques as well as the Doc2Vec algorithm. We conclude the chapter by explaining the used evaluation metrics. In Chapter 3, recent work in related fields is discussed. We start by comparing recent papers proposing techniques for traffic analysis of workstations and compare these to methods for mobile device traffic. Lastly, we briefly analyse existing work in the related field of identifying *Internet of Things* (IoT) devices instead of smartphone applications. The methodology of our proposed solution is then discussed in Chapter 4, where we emphasise the connection between text processing and network traffic analysis. We present three supervised machine learning approaches to detect smartphone applications based on generic network traffic features and describe their theoretic aspects. Chapter 5 explains the architecture and implementation details of each involved module. Furthermore, we elaborate on the detection process and describe the necessary preprocessing of newly

captured data. Afterwards, the datasets used during the performance analysis are discussed in Chapter 6. For each approach, the classification performance on these datasets is summarised and compared to all other models. Finally, we present our conclusions in Chapter 7. We explain the performance differences between the tested approaches and present our findings on whether text processing methods are suitable for network traffic analysis.

Chapter 2

Background

In this chapter, we provide an overview of the foundations relevant to this thesis. It starts with the explanation of relevant features, that can be extracted from network traffic or queried from other public sources. Afterwards, we discuss techniques to capture traffic packets and summarise their advantages and limitations. We then focus on the network programming interface of the Android mobile OS and introduce Netguard, a device local firewall application. This chapter concludes by elaborating two methods to extract context information out of a text. We explain the underlying machine learning concepts and approaches to measure the performance of the trained model.

2.1 Metadata in Network Traffic

With the increasing amount of encrypted network traffic in applications, the collection of data gets more difficult. Although transport encryption schemes like TLS prevent accessing the payload data, the associated metadata is still available. Metadata consists of information the transport protocols need, in order to route each packet to the correct destination. This includes identifiers for sender and recipient, timestamps of different events like the reception of the packet or the transmission duration, but can also consist of flags indicating the special treatment of a packet. Each metadata part on its own provides only limited insight. Combining these parts allows drawing conclusions, for example, whether a packet is a response or request.

Address Range	Designation	Date	Whois	Status
...	...	...	...	...
75./8	ARIN	2005-06	whois.arin.net	ALLOCATED
76./8	ARIN	2005-06	whois.arin.net	ALLOCATED
77./8	RIPE NCC	2006-08	whois.ripe.net	ALLOCATED
78./8	RIPE NCC	2006-08	whois.ripe.net	ALLOCATED
...	...	...	...	...
2620::/23	ARIN	2006-09	whois.arin.net	ALLOCATED
2800::/12	LACNIC	2006-10	whois.lacnic.net	ALLOCATED
2a00::/12	RIPE NCC	2006-12	whois.ripe.net	ALLOCATED
...	...	...	...	...

Table 2.1: An excerpt of the IPv4 and IPv6 assignment administered by IANA.[1][2]

2.1.1 Gathering Information from IP Addresses

From *Internet Protocol* (IP) addresses, various other data associated can be gathered. *Internet Protocol version 4* (IPv4) and *Internet Protocol version 6* (IPv6) addresses are registered to companies or a person, the information of possessing a specific address is often available to the public. The number of addresses obtainable, especially for IPv4, is limited. The central organisation performing the administration of IP addresses is the *Internet Assigned Numbers Authority* (IANA). The IANA distributes blocks of address ranges to *Regional Internet Registries* (RIRs), which themselves manage these blocks in their geographical region spanning across multiple countries. Based on these assignments, it is possible to determine the geographic area, in which an IP address is located. The excerpt of the IANA address assignments in Table 2.1 shows that for example the IPv4 range 80/8 and the IPv6 addresses in 2a00/12 are assigned to *Réseaux IP Européens Network Coordination Centre* (RIPE NCC), the RIR responsible for Europe, the Middle East and parts of Asia. Thus, it can be concluded that IP addresses from this block are located in this service area.

The RIR itself distributes the assigned address ranges among large institutions and *Internet Service Providers* (ISPs) in their area of responsibility. Individuals or companies are able to rent single IP addresses or complete subranges from an ISP. To track possession, each RIR has to maintain a publicly available lookup service. This

[1] https://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xhtml

[2] https://www.iana.org/assignments/ipv6-unicast-address-assignments/ipv6-unicast-address-assignments.xhtml

```
inetnum:        129.27.0.0 - 129.27.255.255
netname:        TUGNET
org:            ORG-TUG1-RIPE
descr:          Zentraler Informatikdienst
descr:          Steyrergasse 30
descr:          Graz, A-8010
country:        AT
source:         RIPE
...
person:         Dept. of Communications
address:        TU Graz/Zentraler Informatikdienst
address:        Steyrergasse 30
address:        A-8010 Graz
address:        AUSTRIA
phone:          +43 316 873 6390
fax-no:         +43 316 873 7699
```

Listing 2.1: An whois response for the IP address 192.27.2.3.

information is contained in public databases, with the respective database servers listed in the IANA assignments. A database can be queried using the whois[3] protocol. The query contains an IP address, the database responds wether the address is part of a rented subnet, to whom it belongs and the postal address of the organisation or individual. For privacy reasons, detailed information of individuals is often redacted. An example of a whois response can be seen in Listing 2.1. In order to provide offline location databases, companies crawl whois responses for all possible IP addresses. If the coordinates are not contained in the response, the location of the included street address is resolved as best approximation.

In the case that the IP address associated with a hostname is part of a subnet range, it could indicate a network of an organisation or company providing backend services. Such services have to handle a significant amount of server load, often distributed among multiple server instances with their instance address being part of this subrange. The distribution is performed by a load balancer, which redirects requests to the servers in order to distribute their workload. The amount of servers used can scale dynamically, depending on the current degree of utilisation and required computational capacity. As such services are provided among a greater geographical region, *Content*

[3]https://tools.ietf.org/html/rfc3912

Delivery Networks (CDNs) are often used to serve static files. Based on the user's location, the hostname of the CDN service instance can resolve to different servers with different addresses. This approach speeds up response times, as the physical proximity brings along a faster connection and can be used for load-balancing purposes as well. The majority of CDNs run their own data centres, providing computing power to their customers for rent. This leads to the situation that a vast amount of large services from different companies resolve to the same IP address range, namely that of the CDN.

In contrast to static addresses, IANA reserved three network ranges as private addresses[4] due to the limited availability of IPv4 addresses. These addresses are assigned dynamically by a local gateway and are not routed on the internet. Instead, *Network Address Transposal* (NAT) is used to replace source and destination addresses within this range with the public gateway address.

On the level of single IP packets, associated information can be extracted as well. Among those, the packet length and header length are of special interest, as both can vary depending on the involved network nodes and higher-level protocols.

2.1.2 Metadata in TCP

Transmission Control Protocol (TCP) is a protocol managing stateful connections between a sender and a recipient. Prior to transmitting data, a session has to be established between the ports of both participants. Every session starts with a handshake indicating that both stations are ready to send or receive data. During a session, data of arbitrary size is split into multiple packets, which are transmitted in order. Every packet received is acknowledged, allowing to resend lost packets automatically. As soon as the packet transfer is complete, the session ends with the connection teardown. The outgoing ports are usually assigned dynamically, whereas static ports are associated with widely used high-level protocols by IANA. Besides providing information which protocol uses this data, each packet contains a number of additional flags in the header. As these are largely dependent on the contacted service and capabilities of the sending station, the sum of flags can be used to recognise repeating connections between these participants. By removing the header padding, the remaining header length can differ from packet to packet. For this reason, the length of the header without padding can serve as another information source.

[4] https://tools.ietf.org/html/rfc1918

2.1.3 Resolving Hostnames

Hostnames can provide valuable insight into the service contacted or the company behind. For example, if an application established a connection to "translate.google.com" and transmits a large payload, it can be an indicator that text should be translated using the translation service of Google. There are a number of ways to extract the hostname out of network traffic and map it to the destination IP address. If the request is made relying on *Hypertext Transfer Protocol* (HTTP), the request header is available in plain text. Version 1.1 and later of this protocol require the presence of the *Host* field in the header, which contains the full hostname of the destination. The more secure equivalent, *Secure Hypertext Transfer Protocol* (HTTPS), instead preserves confidentiality, encrypting the whole request header along with the payload. An exception to this is the protocol extension *Server Name Indication* (SNI)[5]. SNI effects that the *Client Hello* message includes the hostname as plain text in the extension field. Based on the included hostname, the server returns the appropriate server certificate, as multiple names can be hosted on a single machine. If the *Client Hello* message cannot be captured, the hostname can be resolved for an address by querying the DNS. DNS relies on distributed name servers, which contain a mapping between hostnames and static IP addresses. A problem of resolving hostnames in this way is the reliability of the query result. In order to prevent repeated queries for an IP address, the response to a query is cached locally. The mapping of hostnames to addresses can change over time and differ even between multiple nameservers.As a result, the cached response on a device can differ to a newly acquired mapping on another one. Companies with offices in different geographic areas often run their own nameservers to provide language or area-specific content and services.[6] If the address is queried from another country as the original request was made, it could likewise lead to another hostname.

2.1.4 Traffic Gathering

For analysing network traffic, packets have to be captured. This can happen on different points of an active connection. The first possibility is to sniff the traffic directly from the network interface of a device. However, on most platforms, enhanced privileges are needed to do so. Thus, a common practice is to redirect all traffic through a local proxy or VPN server. As both methods run directly on the device, the application producing

[5]https://tools.ietf.org/html/rfc6066
[6]https://kb.isc.org/docs/aa-01149

each packet can be determined. This allows using filter techniques so that only traffic of certain applications is logged. A drawback of these methods is that the implementation of the sniffer has to be adapted for each supported OS. Instead of on-device approaches, traffic can be captured directly on a gateway. This involves local network infrastructure equipment like wireless access points or routers as well as remote proxies or VPN servers, eliminating any platform dependencies. If a gateway is connected to multiple devices transmitting data, the sniffed traffic can get large in a short amount of time, potentially containing a considerable amount of unwanted noise from irrelevant devices. Although traffic can be filtered for single devices based on the IP address, such filters are not easily extendable to an application level, unless these apps are using custom ports.

A vast amount of tools to capture network traffic exists, with Wireshark[7] being one of the most widely known. It is designed as multi-platform protocol sniffer and analyser, supporting a wide range of different protocols. Able to perform live capturing from physical or virtual interfaces, it can provide detailed insight on all network levels. Wireshark utilises a tailored filter syntax to display captured traffic of interest. It allows to export and import gathered network traffic in various formats, for example in the *Packet Capture* (PCAP) library structure, which is often used to exchange recorded traffic with other programs. Additionally, multiple other tools are part of Wireshark, like the command line interface *tshark* or the remote capturing application *tcpdump*.

2.2 The Android Network Interface

All applications developed for the Android mobile OS run in their own sandbox, which prevents uncontrolled access to the outside world. This concept is implemented by assigning a device-unique *User Identification* (UID) to each application upon installation and enforcement through the kernel. To access other data, hardware interfaces or to transmit data in a network, applications need to acquire the consent of the user in the form of permissions. Permissions are divided into multiple categories, with *installation* and *runtime* permissions being the most popular groups. The latter, often referred to as dangerous permissions, consists of those which potentially compromise the security of the device or have an impact on the privacy of the user. Examples of this category are the permissions to use the camera or record audio. Prior to using such interfaces, an app has to inform the user about the purpose and ask for confirmation. The user can withdraw the confirmation at any time. Install permissions involve all those, which

[7] https://www.wireshark.org/

are considered as not harmful. Thus the OS grants them automatically during installation. The user is informed about the need for access, but cannot reject them at a later point. A prominent example is internet access. With this permission implicitly granted, applications are able to utilise the high-level interface provided by Android for network communication. Requests are sent asynchronously to the destination *Uniform Resource Locator* (URL) by the `HttpUrlClient` class, with possible payload data serialised to plain text. The response is delivered to an application callback. The Android OS supports HTTPS connections out-of-the-box, either by instantiating `HttpsUrlClient` or passing an HTTPS URL scheme to `HttpUrlClient`. Internally, the latter method performs an upcast to `HttpsUrlClient`. Google encourages the use of HTTPS, with Android version 9.0 disabling plain text transmissions by default. For enhancing the security configuration of the connection, no modification of the application code is needed. Instead, custom trust anchors or pinned certificates can be defined in an additional resource file.

Additionally, Android enables applications to serve as VPN client by extending the `VpnService` class. Although the permission needed is not considered dangerous and thus part of the install category, the user is notified on the first application start. Furthermore, the OS indicates the VPN connection as long as it is active. The `VpnService` class is able to create and configure virtual IP network interfaces, which acts as file descriptor in order to receive and inject packets. The application can then establish a connection to a remote VPN server, forward all packets to it and deliver the response back to the file descriptor.

2.2.1 Netguard - A non-root Firewall

A VPN server does not have to be a dedicated machine but can be included in an application as well. *Netguard*[8] utilises this idea to serve as local firewall. As it only requires install permissions to permit or reject other applications from accessing the internet, no rooting is needed. To allow the user to block network traffic of single applications, Netguard must be able to link the produced traffic to the producing application. By querying the Android `PackageManager`, Netguard can determine the UID of the current active app when a packet was sent. The `PackageManager` provides two more functions, which are useful for Netguard in this context. The first is `getPackagesForUid(int uid)` which returns the application package names assigned to a UID. With this method,

[8] https://www.netguard.me/

the network packet can be linked to the application producing it. The other method is getInstalledPackages(*int* flags), which allows the user to define blocking rules for all installed apps. Netguard compares the application responsible for every encountered packet against the filter list. If the application is not allowed to access the internet, the packet is dropped. Since all packets have to pass the virtual interface of Netguard, the traffic can be recorded and exported in the PCAP format.

2.3 Text processing

Automatically extracting the context of arbitrary text documents is an active research field with a vast amount of different approaches. During the creation of this thesis, we picked two of these methods and adapted them to find patterns in the structure of network data. Both methods are based on supervised machine learning models and require input in the form of a labelled dataset. For both models, the dataset consists of a number of text documents, each consisting of multiple sentences itself. To every sentence, a textual label has to be associated, describing the related context class.

As does the majority of approaches, both models require preparation of the available textual data in order to perform well. The first step is the removal of words, which do not contribute to the derivation of context. A prominent category of these are conjunction words like “and”, “or”, “but” or “for”. While being useful in a conversation between humans, so-called stopwords are not related to specific topics and would only increase the needed processing time and space.

After this step, the remaining words in the dataset are reduced to their respective base form and encoded into a set of integers. This representation enables the application of mathematical operations, as every sentence can be considered as n-dimensional vector, where n is the length of the sentence. Such vectors are often referred to as feature vector. The associated labels are transformed in a similar manner. Instead of being encoded as integers, one-hot encoding is used. In comparison to the previously described integer encoding, this method has the advantage of removing any implicit hierarchies from the data. With this method, each label is encoded into a k-dimensional vector, with k being the number of labels. Every vector consists of 0’s and exactly one 1, with its position being unique for all vectors. An example of this encoding technique is given below.

$$
\begin{aligned}
cat &\rightarrow \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \\
dog &\rightarrow \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \\
mouse &\rightarrow \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}
\end{aligned}
$$

As the last part of the preprocessing, the data is converted into sequences of equal length. Shorter sequences are either padded up to the maximum length of all or to a fixed size. In the latter case, words of longer sentences are neglected above the limit and information is lost. For this reason, this technique is usually applied on datasets containing only a small amount of outliers, so that the tradeoff between reduced information and increased training speed is within an acceptable boundary.

Depending on the implementation, some or all of the described steps can be included in the framework. This is the case in the first approach, *Doc2Vec* [15], which is implemented in the *Gensim Framework*[9]. Doc2Vec extends the idea of *Word2Vec* [23] to process multiple sentences or documents. Word2Vec is a machine learning model that allows deducing the relation between words. In order to do this, the model transforms each word of a text into a word vector. Using vector algebra, the similarity of these vectors can be measured, and the model is able to figure out whether two words are synonyms, analogies or antonyms. An example scenario would be deriving the relation between Paris and France based on the fact of knowing that London is the capital of the United Kingdom. In vector algebra, this can be modelled with the equation $\vec{v}_{uk} + \vec{v}_{london} - \vec{v}_{paris} = \vec{v}_{france}$. The graphical representation of this equation can be seen in Figure 2.1.

The underlying neural network can use two different algorithms for deriving the word vector from a feature vector, either *continous bag of words* (CBoW) or *skip-gram*. The former utilises a sliding window in order to predict the middle word based on the surrounding words. To be able to do this, the numerical representation of the surrounding words is averaged. This means that the ordering of the words in the sentence does not influence the outcome. Skip-gram instead uses the current word to predict the next and previous words. With this algorithm, the influence of words is weighted according to their distance to the current one. Both models lower the internal complexity

[9] https://radimrehurek.com/gensim/models/doc2vec.html

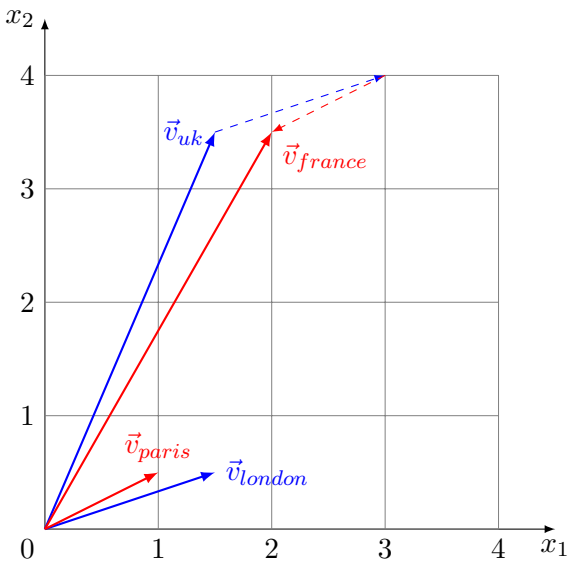


Figure 2.1: Graphical representation of the equation $\vec{v}_{uk} + \vec{v}_{london} - \vec{v}_{paris} = \vec{v}_{france}$.

of the neural network, increasing the computation performance while still providing well-performing word vectors. However, a limitation of Word2Vec is that it does not consider any information contained in larger structures like sentences, paragraphs or documents.

Doc2Vec overcomes this weakness by adding the context information of paragraphs to simple word vectors. Le and Mikolov propose two methods to perform this task in [15], which replace CBoW and skip-gram in the neural network. PV-DM predicts the next word from a feature vector that considers the paragraph context. This context represents information, that is missing if only the feature vector alone was used. The paragraph context is modelled as a vector similar to the feature vectors of single words, thus referred to as paragraph vector. During the training process, the neural network infers these paragraph vectors and adapts the associated weights. An example of PV-DM is shown in Figure 2.2. While the word vectors are shared over all paragraphs in the text, the influence of each paragraph vector is limited to its own context. As the input feature vector for the prediction depends on the current and previous words, the ordering of these matters. For this reason, semantics between words and between paragraphs are preserved in the trained model. In contrast to PV-DM, the paragraph vectors used in PV-DBoW do not depend on any input words out of the paragraph. Instead, the vector is trained to predict randomly sampled words from the paragraph as output. This method has the advantage of a reduced memory footprint compared to

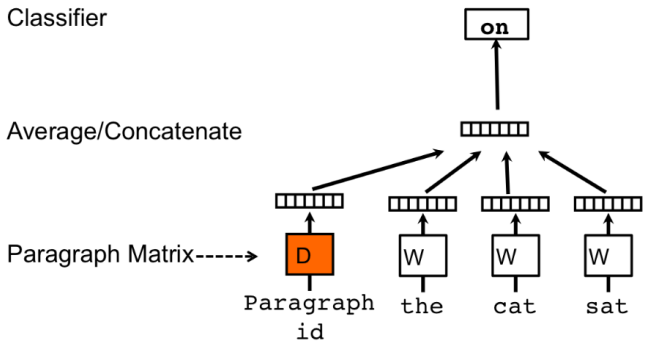


Figure 2.2: Using PV-DM to predict the next word from three feature vectors and the paragraph vector[15, p. 1190].

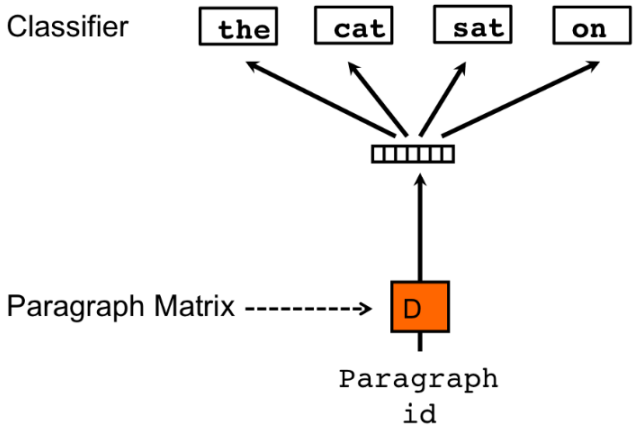


Figure 2.3: PV-DBoW predicts words from the paragraph without the need of input feature vectors[15, p. 1191].

PV-DM, as only the weights have to be saved during the training. The simple structure of PV-DBoW can be seen in Figure 2.3. Although the authors state that the usage of PV-DM alone should work well, they recommend to combine both methods in order to achieve more consistent results across different applications.

With the described methods, the neural network is trained to classify the context. In each training epoch, the outcome is derived according to the used method and compared to the ground truth, being the assigned label encoding. The weights used to derive the resulting vector are adapted accordingly, emerging in an high-dimensional vector space. For classifying new text, the network infers a new vector for this input. The cosine similarity to the surrounding vectors is computed and compared against others. Since

each vector represents one class label, the probability that the newly inferred vector belongs to each class relates to the associated distance.

2.3.1 Deriving Context with Multi-Class Classification

Another widely used approach for text analysis is to build a neural network by defining the layer structure oneself. In the most simple way, each layer of a neural network consists of a number of equal neurons. When these neurons receive input, the result of each neuron is computed using an activation function and provided as output to the next layer. The activation function should ideally be non-linear in order to enable the model to learn complex decisions. The neurons of every layer are connected to their counterparts in the next, where the number of neurons can differ between layers. The connections themselves have weights associated, which are adapted during the training phase. For adapting the weights, the network performs the intended task, for example, classifying the training set, and evaluates the outcome. By using backpropagation, the distance to the correct result is measured with a loss function. Prior to the next training cycle, the weights responsible for this outcome are adjusted using an optimisation strategy. The training phase should last until the loss function converges, meaning that nothing new can be learned from the input. For reducing the time needed for training the neural network, the training dataset can be split into multiple batches, which can be processed in parallel. A part of the training dataset can be reserved as a validation set. The model never learns on this data but evaluates samples of it during each training epoch. As the validation set is separated from the training dataset, the performed evaluation outputs unbiased results of the model performance on the training set. This results can be used as a basis for improving the model parameters.

One of the main challenges in the field of machine learning is to find well-performing parameters for each network layer, the optimisation strategy, as well as appropriate loss and activation functions. While the latter relies on experience, the former can be determined empirically. Grid search is a widely used strategy to find a well-performing model parameter combination. The model is trained for multiple iterations. Between each iteration, one value of the model parameter set is changed according to a given step size and the model performance is evaluated. This process of changing parameters, training and measuring the model performance is repeated until a given number of iterations is achieved, or a given performance threshold is exceeded.

For creating a neural network, we used the *Keras*[10] framework. It abstracts different machine learning backends like *Tensorflow*[11] with a high-level syntax. Keras allows creating a neural network by initialising the used layers with their respective parameters and connecting them. The model has to be compiled with a loss function and optimisation strategy before it can be trained for a specified number of epochs using a training and a validation dataset.

In order to process text, the first layer of the neural network has to be an embedding layer. As the data preprocessing provides sequences of positive integers, this layer transforms these sequences into vectors of the desired output length. The output length should be chosen in a way that considers the input size of the following layer. To perform the classification, the model utilises a dense layer. This type of layers consists of a number of neurons of which each one has an weight assigned. Given an input in the form of an vector $\vec{u} \in \mathbb{R}^n$, the weights of the training iteration i as matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ and a bias vector $\vec{b} \in \mathbb{R}^m$, the layer can be represented with the following equation.

$$\mathbf{W}_{(i+1)} = f(\mathbf{W}_{(i)} \cdot \vec{u} + \vec{b})$$

The activation function f has preferably non-linear characteristics and has to be differentiable, in order to calculate the loss. A function commonly used for this application is the ReLU function, which is defined in the following way.

$$f(x) = max(0, x)$$

As can be seen in Figure 2.4, the derivative of the function for any negative input is 0, whereas for positive integers the slope is 1. This property is important for efficient backpropagation. The dense layer can either be used as a middle layer or as the last layer of the network, outputting the classification results. In the latter case, the number of neurons represents the available output classes.

There are multiple layers dedicated to decreasing the occurrence of overfitting. A naive approach is the use of a dropout layer, which simply drops a part of the received input. This randomly eliminates some information during the training phase. In this way, a dropout layer can remove outliers with a given probability, leading to a more generally applicable model. In this layer, only the percentage of removed data can be specified. An alternative to a dropout layer is the reduction of data dimensionality using

[10] https://keras.io/
[11] https://www.tensorflow.org/

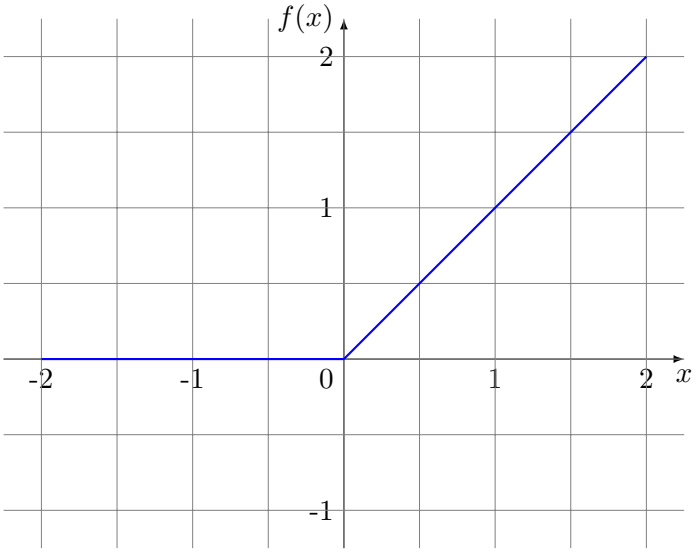


Figure 2.4: Graphical representation of the ReLU activation function.

a pooling layer. Pooling layers apply a sliding window on the input vector and reduce its content to either the minimum or maximum value. Depending on the chosen type, these layers are referred to as min or max pooling. A recommendation for natural language processing is to set the size of the sliding window equal to the input size. A pooling layer with this window size is referred to as global min or global max pooling layer. Since the layers following a global max or min pooling layer only get the maximum or minimum value of a feature vector, the position of the value in the vector is irrelevant. For this reason, the layers afterwards cannot base the decision on one feature position. Another benefit of this layer is the increase in performance, as the dimension of features is reduced. Pooling layers are often used to reduce the output dimension of convolution layers. This layer applies a convolution operator with a specified filter on an input, essentially multiplying the input function with the filter resulting in a modified signal. In mathematical terms, the convolution of two functions f and g can be expressed in the following equation.

$$(f * g)(i) = \sum_{j=1}^{m} g(j) \cdot f(i - j + \frac{m}{2})$$

To compile the defined layers into a usable model, a strategy to update parameters during learning has to be defined. A widely used optimiser is *Adaptive Moment Estimation* (Adam), an adaptive learning method. It calculates individual learning rates

for each parameter and takes the momentum of change into account. This momentum is represented as exponentially decaying average of past gradients and allows a fast convergence, resulting in a faster and more effective learning process.

2.3.2 Evaluating the Model Performance

To fine-tune the parameters of a neural network using grid search, metrics to assess the performance of neural networks are needed. If the model would strictly memorise the training data, measurements using the same dataset would be distorted. For this reason, all metrics are evaluated with a dedicated test set. This set consists of previously unseen data, testing the ability of the model to generalise the learned reasoning. Commonly used key figures to rate the classification performance of a neural network are accuracy, precision and recall. Accuracy is defined as the number of correct classifications over the amount of all classifications, represented by the cardinality of the test set S. The number of correct classification is often substituted using the zero-one loss L. This loss function yields the number of misclassifications by comparing the predicted label i to the ground truth j.

$$Accuracy = 1 - \frac{L}{|S|}$$

$$L(i,j) = \begin{cases} 0 & i = j \\ 1 & i \neq j \end{cases}$$

The accuracy of a model should preferably be high on the test set. A drawback of this metric is that it only indicates that a number of misclassifications have happened. The metric does not express, if misclassifications are occurring mainly between two classes or if they are distributed evenly across all.

Because of this characteristic, the prediction accuracy alone is an unsuitable metric for model evaluation and parameter tuning. Instead, precision and recall are used since they show more fine-grained information on the model. For calculating both metrics, the number of true and false positives and negatives has to be computed. True positives are samples of the dataset, that are predicted as the target category it was labelled. If the sample would instead belong to another category, this classification would represent a false positive. Assume a classifier that should detect all cats from a dataset consisting

of cats and dogs. In this case, a true positive is a cat that would be predicted as a cat. Conversely, if a dog would be classified as a cat, the prediction would be a false positive. Similarly to true positives, true negatives were correctly predicted as not belonging to the queried category. In the previous example, a true negative would be a dog that is classified as a dog. False negatives instead belong to the queried category but were predicted as another category. This case occurs if a cat is classified as a dog. This scenario can be seen as two-class classification problem and represented as confusion matrix, shown in Table 2.2.

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Table 2.2: This confusion matrix can be created by comparing the predicted with the assigned labels.

The precision of a classifier measures the proportion of correctly classified positives with respect to all samples that were classified as positives. It is defined as follows.

$$Precision = \frac{True\,Positive}{True\,Positive + False\,Positive}$$

Most of the time, this metric is used in combination with recall, which calculates the number of true positives out of all positive labelled samples. This yields the following formula for recall.

$$Recall = \frac{True\,Positive}{True\,Positive + False\,Negative}$$

Both metrics are computed for each prediction class of the neural network. The results can lead to a better understanding of the classification, as similarities between classes can be revealed by analysing mispredictions regarding a target class. Additionally, they allow rating the model in regard to the costs of wrong predictions. Applied in an image search engine, false positives or false negatives may have less impact than in a breast cancer detection. For this reason, precision can be sacrificed for a better recall rate or vice versa, depending on the use case. In most cases, it is desirable that precision and recall are in balance since the change of one reflects in the other. For providing an easy measurement method for such a scenario, the F_1 score is taken into account. This

metric combines precision and recall in this formula.

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

The above-described confusion matrix can be extended to a matrix of the form $n \times n$, with n being the number of classes. The output classes are symmetrically distributed, such that the diagonal entries represent correct classifications. Compared to the previously described two-class classification problem, these entries reflect true positives. The amount of incorrect classifications is split, depending on the misclassified category. An example of a confusion matrix with three classes is given in Table 2.3, with a total of 20 samples per class.

		Actual		
		Cat	Dog	Mouse
Prediction	Cat	15	4	1
	Dog	4	16	0
	Mouse	2	1	17

Table 2.3: An example of a confusion matrix for a 3-class classification problem.

In a classification problem with multiple classes, a confusion matrix provides valuable insight into the type of error the network is making. From the resulting figures, it can be detected if misclassification occurs between two similar classes or if it is distributed among all other classes. On a dataset with an unbalanced amount of samples per class, the confusion matrix can reveal if classes with a greater amount of samples perform better than others. This prevents tuning parameters of the network only on the biggest class, neglecting categories with a smaller amount of samples.

For gaining an insight into the classification process in the intermediate layers of the neural network, the layer output can be visualised. If the Keras framework is configured to use the Tensorflow backend, data can be visualised with *Tensorboard*[12]. As a requirement, the according interfaces have to be implemented in the prediction process. Since the data captured in this way is high-dimensional, the dimensional complexity has to be reduced to be human-understandable prior to displaying. Tensorboard performs dimension reduction using *Principal Component Analysis* (PCA), *t-Distributed Stochastic Neighbour Embedding* (t-SNE) [19] or UMAP [21]. PCA is a linear reduction method that preserves the relation between data points during the transformation. For reducing

[12] https://www.tensorflow.org/tensorboard

each data point to 2 or 3 dimensions, the variance between all data points is calculated. In order to perform the calculation, each point is projected onto hyperplanes, which pass the centre of gravity of the point cloud. The variance is the distance between the centre of gravity and the projected point on the hyperplane. Based on these hyperplanes, corresponding transformation vectors can be found, which are applied on each point to convert the data to low dimensions.

T-SNE is an unsupervised non-linear reduction technique. This method tries to retain local data relations during the transformation while avoiding the cluttering of linear methods like PCA. Between each pair of points in high-dimension, a similarity score is computed. During the following reduction, the distance between all points is optimised, such that the similarity score remains near the original value. The results of t-SNE depend largely on the chosen parameters, but can outperform PCA due to the non-linear characteristic.

Being faster than t-SNE, UMAP is another non-linear approach based on manifold learning techniques. By using simplices, topological structures are modelled onto the high-dimensional data to create a manifold. It then reduces the manifold to low-dimensional space, with the constraint of optimising these representation to be as close as possible to the original topology.

As these dimension reduction techniques preserve the relation between the data, possibly overlapping of classes can be detected. This allows adapting the associated parameters, resulting in a better performing classifier.

Chapter 3

Related Work

Gathering information by observing network traffic is a well-studied field, and thus a vast amount of research exists. Since extracting data out of plaintext traffic does not present a challenge to an observer, most of this work focuses on encrypted traffic. Although traffic analysis on mobile devices seems to be just an application of traditional approaches, there are some challenges. Smartphones are highly mobile devices, switching between different wifi networks or cellular data many times during a single day, leading to missed or out-of-order network packets during data capturing. In extreme cases, even the destination server can change if the user travels to another location. The first part of this section describes robust approaches to identify protocols and programs on workstations. These ideas are comparable to the concepts used in methods to identify smartphone applications, which are discussed afterwards. The identification of applications based on the produced traffic is quite similar to the problem of identifying IoT devices based on captured traffic. The chapter closes with an overview of this topic.

3.1 Workstation Traffic Analysis

Traditional traffic analysis on workstations often relies on single features of network traffic. Some applications, for example, are using well-known TCP or *User Datagram Protocol* (UDP) ports to communicate with the corresponding backend service. If the target machine receives network traffic on this port, the presence of this application is leaked. In the same way, the use of transport encryption does not hide the target's IP address, providing a way to identify the contacted service and in a further step the

related application. Such identification attempts suffer from several limitations. Modern applications often rely on CDNs like *Akamai*, *Cloudflare* or *Amazon Web Services* for scalability reasons. To use a CDN efficiently, dynamically created and static application content can be separated and served from different locations. These locations can be distributed among different servers, resulting in multiple contacted IP addresses. Most CDNs support adding or remove additional mirror servers based on the current load, which can lead to frequently changing server addresses. The described reasons tend to complicate service identification based solely on IP addresses. With the advance of data serialisation techniques like *JavaScript Object Notation* (JSON), the only used network ports in most applications are the HTTP and HTTPS default ports, providing an effective countermeasure for methods relying only on ports.

In [14], Kim et al. combine multiple network traffic attributes into network states. The resulting states are clustered, allowing the visual identification of anomalous traffic. Kim et al. expand their original idea in [13] to overcome the limitations encountered in [14] like online clustering of streaming traffic. In order to identify unexpected network events in this data, they introduced a grid-based model. In [25], Nazari, Noferesti, and Jalili use deep packet inspection to extract available features as ground truth data dynamically. This data is used to update and evaluate different stream classification algorithms, of which adaptive random forests yield the best results. As a significant share of applications enforces transport encryption, deep packet inspection cannot be used to extract features automatically in this way. For this reason, we rely on identifying relevant features manually. The authors of [8] picked up the idea of using multiple neural networks in combination. To boost the overall precision and gain advantage from parallelisation possibilities, Dong and Li utilised one tailored neural network per identifiable application. Depending on the number of applications in scope, this approach could lead to a large number of models, possibly resulting in significantly longer training times. As the models are based on two-class classification problems, the effort of labelling training data can be substantially higher compared to the multi-class approach in our work. In [39], the impact of the used network feature set in different machine learning techniques is evaluated. The authors began with 111 possible features, reduced them step-wise and tested the performance of J48, random forest, k-nearest-neighbours and a Bayes network. They found 12 essential features which slightly increased the original classifier accuracy. Jain describes the idea of adopting image processing convolutional neural networks to process network data and identify used protocols in [12]. In [18], the authors propose the combination of a convolutional neural network with a stacked autoencoder for feature extraction. Their framework is able to classify the network

protocols used as well as the applications themselves. Pluskal, Lichtner, and Rysavý present a statistical approach for protocol identification in [29], which results in a slightly lower accuracy but better performance than Bayesian networks and random forest.

In [11] Ichino, Maeda, and Yoshiura used k-nearest-neighbours to cluster web applications based purely on the statistical information of all packet headers. They infer protocols using a similar clustering method based on packet size, timing information and packet direction in [38]. Compared to the work of Ichino, Maeda, and Yoshiura, the Kullback-Leibler distance was used to infer new clusters. In order to calculate the needed statistical information in both approaches, the complete network traffic is required. For this reason, both approaches do not work online.

In contrast to the above, Hajjar, Khalife, and Diaz-Verdejo do not rely on features gathered at network-level like IP addresses in [9]. By using only the first application-layer message in a flow, the authors are able to identify protocols utilising a combination of Gaussian mixture and Markov model, disregarding patterns introduced by user interaction.

3.2 Mobile Device Traffic Analysis

A vast amount of research applied or extended previously described methods to capture network traffic of mobile devices. Conti et al. provide an overview of recent developments in this field and present their survey results in [6]. They propose a classification system for approaches based on the goal of the analysis, the source of captured traffic and targeted mobile platforms. The majority of examined work gathered automatically generated network traffic directly on a mobile device, prior to applying machine learning techniques. In addition, they discuss possible countermeasures, for example, the use of transport encryption to prevent deep packet inspection. To counter application identification approaches, Conti et al. suggest to either apply random padding to obfuscate the packet size or delay network packets. These methods either increase the used bandwidth or the computational costs significantly, both being limited resources on mobile devices. For this reason, the proposed application identification countermeasures are not feasible in practice.

In [37], Wang et al. adopt the method of fingerprinting website traffic originating from workstation traffic analysis to the mobile sector. In their work, the authors compare the privacy impact between the usage of a dedicated app and the use of a mobile

browser. They analyse packet-level traffic to find traffic patterns. By using random forest classifiers, Wang et al. are able to identify applications and show that distinguishable traffic poses a serious threat to the user's privacy. Sivan, Bitton, and Shabtai support the results of Wang et al. in [33]. Although this work uses deep packet inspection methods and focuses only on the leakage of user location information, the authors show that they are able to identify applications which send sensible user data to remote servers.

Smartphones are subject to malware infections in the same manner as workstations. The detection of device infections is a challenge in which network-based application identification constitutes a well-performing solution. Arora, Garg, and Peddoju make use of 16 network features to define a set of rules in [3]. Afterwards, a classifier identifies malware-infected devices in the network based on these rules instead of features. Conversely, Zaman et al. detect malware by inspecting packet headers instead of whole network packets in [41]. Malicious domains are blacklisted, the destination of each packet is checked against this list and classified as malware if a match occurs.

Packet header inspection is also the main idea in [30], which describes an extended approach to identify user activities. The header information of encrypted wifi traffic is used in a multi-class *Support Vector Machine* (SVM) to allow a fine-grained activity inference.

In [10], the authors try to find correlations between unencrypted and encrypted traffic of single apps. They focus on temporal, lexical and metadata similarity to identify applications. If no correlations are found, or plain traffic is not available, their method relies on clustering the application's DNS queries and server hostname lookup as a fallback. As more and more mobile applications tend to use transport encryption, the described approach will have to use the fallback method more frequently, decreasing the detection rate. Thus more reliable techniques are needed, like combining multiple network packet features used in our work. In [2], Alan and Kaur make use of supervised learning models originating from web page identification methods, which analyse information in TCP headers captured only during the launch of an application. Focussing on this timeframe, patterns introduced by user interactions are ignored in this approach. The authors tested their method on 1595 applications and achieved an accuracy of 88 per cent. In [31], time-series patterns are found in encrypted traffic. By implementing a probabilistic state transition model, the authors can identify applications based on the estimated network flow. To predict apps that a user will most likely use, the approach in [24] utilises *Call Detail Records* (CDRs) of cellular network data. Mizumura et al. compare the prediction performance of a SVM, *k-nearest-neighbours* (k-NN), decision-tree,

neural network, the naive Bayes method and random forest, the last one performing best in a real-world scenario. Since mobile applications can reduce network traffic if a cellular network connection is used, the model performance could be different if used on other traffic from other connection types.

In [35], a framework called AppScanner is able to fingerprint applications and to identify them in real-time. Unlike in our work, the captured network traffic of real devices is split into bursts and flows. Bursts are all packets occurring beneath a given time threshold, while flows consist of all packets inside a single burst with the same destination. Using SVM and random forest classifiers, they achieve a classification accuracy of 99 per cent. In [36], they extended their previous method to mitigate traffic shared between different applications and evaluated the change of application fingerprints over time. Aiolli et al. show in [1] using supervised learning that even applications of the same type, as well as individual user activities inside one app, can be distinguished. They captured the traffic of several bitcoin wallet apps on real hardware, extracted network features as described in [35] and used them in a SVM and random forest classifier. Compared to our work, their classifier is tailored to detect only wallets and does not seem to be applicable for identifying other types of apps. Chaddad et al. make use of AppScanner in [5] to identify mobile applications in order to evaluate the performance of packet size obfuscation as a countermeasure. In their work, Chaddad et al. are able to change network statistics to a given target app, effectively hiding the presence of applications. As a complete contrast to previous approaches, [16] use a variational autoencoder network to transform mobile network traffic into vision-meaningful images. This allows for learning on unlabelled data since features are extracted automatically. The authors achieve an identification accuracy of 99.6 per cent.

3.3 IoT Traffic Analysis

Related to identifying applications running on smartphones and workstations by using pattern detection on corresponding network traffic is the problem of distinguishing IoT devices in an arbitrary network environment. Martin et al. combine in [20] a recurrent neural network with a convolutional neural network and correctly identify IoT devices based on their network traffic in 96 per cent of all scenarios. This result is outperformed by changing the classifier in [32], where the classification results of random forest, decision tree, SVM, k-NN, neural network and Gaussian naïve Bayes techniques are compared. The random forest classifier achieved the best results with an accuracy of 99.9 per cent

on the test set. The authors gather the first network packets until a given time threshold is exceeded to cut off long-lasting sessions. The classification depends on the packet size and the calculated inter-arrival time of these packets. Meidan et al. instead tested their approach in a noisy network environment containing smartphones, workstations and IoT devices in [22]. Their method consists of a binary and a multi-class classifier. The first one decides whether the traffic originated from an IoT device, while the latter identifies the type of device. Instead of network packet features, the detection process only takes network session properties in combination with the HTTP user agent into account.

With the increasing distribution of IoT devices, the danger of malware infections on these devices increases as well. Nguyen et al. detect such infections based on anomalous network traffic in [27]. As a first step, the used neural network classifies device types and builds the communication profile on statistical features. This has the advantage that features like dynamically assigned ports are reduced to a finite number of tokens, which boosts the detection rate and speed. Although their model is designed for anomaly detection, we applied the same reduction of dynamic features in our work.

Chapter 4

Approach

In this chapter, we examine the theoretical aspects of this thesis. We begin by highlighting the connection between natural text and network traffic metadata and discuss the reuse of existing text processing techniques to analyse network data of smartphones. We propose three supervised machine learning approaches based on natural language processing models. In the first one, we use Doc2Vec in combination with a logistic regression classifier to predict applications that produced the captured network traffic in the dataset. The second approach makes use of a tokeniser and dedicated neural networks to perform this task. The last approach combines Doc2Vec for processing textual data with a dense classification network to predict the application labels.

4.1 Similarities of text processing and traffic analysis

On closer inspection, natural language text documents and network traffic exhibit certain similarities. Arbitrary text, as well as captured traffic, consist of structured data. In the former case, this structure is defined using a grammar appropriate for the language in which the text is written. If network traffic is considered as another language, it can be seen that the underlying protocols define the structure. In captured traffic logs, it is easy to observe that an HTTP response always appears after a corresponding request. Similarly, a TCP handshake initiates a data transfer, that ends with either the teardown or a timeout. For this reason, we consider the structure dictated by protocols equivalent to the grammar of a language. The foundation of this language are single packet features, representing words. The protocol grammar dictates the possible placement of words

in order to form sentences, which are the representation of a single network packet. All sentences built from packets of one TCP session can be grouped in one document. Different to grammar consisting of static rules, words are adaptive components, and their placement may vary. This is true as well for network traffic, in which for example the position of the source port number inside the capture log remains the same, whereas the port itself can be an arbitrary number between a given lower and upper bound. The described similarity between text and traffic metadata allows for applying existing text processing methods to predict the smartphone applications producing individual traffic patterns. We rely on the following network packet features in order to create sentences, with all sentences inside one TCP session considered a document.

- **Source and Destination IP Address** The source and destination IP addresses can either be static or dynamically assigned. The latter category consists of private or link-local IP addresses. This type of addresses is assigned to end-user devices like smartphones and workstations by a gateway. The assignment happens randomly in a given range and is only valid during a specified lease time, after which it can be reassigned or extended to another device. Besides the indication of end-user devices behind this address, the dynamic characteristic of private addresses only increases the level of noise in the dataset. For this reason, we replace the private IP address with the placeholder *dynamic*.

  Static IP addresses in network traffic can indicate service backends. These services can make use of load-balancer or CDNs to distribute the incoming traffic among multiple servers. Load-balancing can be performed by using dedicated hardware or by a DNS server. The former method does not provide any additional information since the load-balancing hardware appears as a single destination. In the latter, multiple IP addresses are registered for a single domain name. The server responds with a different address to each DNS query, with the changing addresses being logged in network traffic. As these alternating addresses complicate the recognition of already contacted services, we decided to generalise static IP addresses to the containing subnet range. The subnet range usually remains the same for all servers behind such distribution techniques, effectively treating all servers as a single destination. However, this generalisation obscures structural patterns of requests and responses, for example, possible redirects to dedicated authentication servers.

  From the placement of dynamic and static server addresses in the source and destination fields of an IP packet, the packet direction can be determined. From

the point of view of a smartphone, a dynamic source and static destination address imply an outgoing request, whereas a static source and dynamic destination address indicate a response.

- **Source and Destination TCP Port** The first part of the available TCP port range consists of ports associated with widely used protocols. Therefore, the presence of such well-known ports can provide insight into the protocol utilised in the application. In most cases, smartphone apps tend to transmit data as serialised text on top of the HTTP and HTTPS protocol, which is associated with TCP ports 80 and 443. For exceptional use cases like sending and receiving e-mails, custom destination ports are used, often deviating from the associated well-known ports. In traditional traffic analysis, these custom ports are often combined with the IP address of the destination to determine the service. Since the OS assigns TCP ports randomly to applications initiating a transmission, they can change between multiple connections to the same service. Similarly to private IP addresses, dynamically assigned port numbers are replaced by the placeholder *dynamic*. The combination of well-known and dynamic ports is another direction indicator, useful if packets transmitted between two static IP addresses are contained in the dataset.

- **TCP Packet Flags** Each TCP session starts with an initial handshake, in which the *SYN* and *ACK* flags are set. In cases where the destination of the initial handshake is an unreachable port of an existing machine, the destination server responds with the *RST* flag set and the connection is terminated. After the data was transmitted, the session ends with a connection teardown, using the *FIN* flag in combination with *ACK*. With these flags contained in the packet information, the captured network traffic can be split into individual sessions. The *ECE* and *CWR* flags are used for congestion management. If these flags are set, a connection to a service with a high load level can be assumed. This can indicate that the service runs on a server without an appropriate load-balancer. As patterns created from these flags are observable, the flags of each packet are considered as another uniqueness characteristic of a connection in addition to IP addresses and TCP ports.

- **TCP Header Length** The length of the TCP protocol header varies between a minimum of 20 bytes and a maximum of 60 bytes. This range results from the inclusion of additional options, the most common being the maximum segment size, window scale, selective acknowledgement and TCP timestamps. In the same

way as TCP flags, the presence of flags is an additional uniqueness factor of a connection. As the content of most options changes from packet to packet, the size of options remain the same and correlates directly to the header size.

- **IP Packet Length** As the underlying protocol constrains the length of IP packets, the packet length varies between a minimum of 20 bytes and a maximum of 65 535 bytes. In most cases, the maximum length is restricted by the *Maximum Transmission Unit* (MTU) of the ethernet or wifi hardware used during the connection. Because of this connection dependency, the packet length serves as an additional distinction characteristic.

4.2 Processing Network Traffic using Machine Learning

To detect the smartphone applications responsible for each network packet, we apply different machine learning models common in the field of natural language processing. In comparison to traditional traffic analysis techniques, approaches that are based on machine learning provide several advantages. Since captured network traffic logs can be quite large, traditional approaches can overlook intricate patterns hidden in this amount of data. Machine learning models are more suited to detect semantics and instead benefit from a large number of available training examples in order to increase the classification accuracy. The trained model itself has a small memory footprint and can classify new input data efficiently. As a result, the prediction itself can be performed on a resource-constrained device. During the learning phase, the model adapts itself to the dataset. It discovers features that are important for the distinguishability and weights their impact according to their relevance for the prediction.

Unsupervised machine learning techniques cluster similar data into one group. Since smartphone applications can contain third party libraries like advertisement frameworks, the traffic of these libraries would be grouped into a single cluster. To prevent this scenario, we instead rely on supervised learning approaches. Although supervised models allow inferring application context in a more detailed way, these techniques come with the downside of requiring appropriately labelled training data. We utilise the application name as label and associate it to every produced network packet. In order to do so, we capture the traffic on the device itself. The gathered traffic is split and used as a training and test set. Based on the application names in this data, we can generalise the labels into application categories or the identifier of shared network libraries at a

later stage. For performing a classification on network metadata, we propose the following three supervised machine learning techniques. The first approach adapts Doc2Vec to process the textual input documents, as it is widely known for performing textual analysis. A logistic regression classifier predicts the application label based on the processed documents. We compare this model to a convolution neural network and a dense neural network with prior tokenisation of the input documents, explained in the second approach. In the third approach, we use Doc2Vec to create document vectors from the dataset but combine it with a dense neural network instead of the logistic regression classifier.

4.3 Approach 1 - Pure Doc2Vec

The first approach makes use of Doc2Vec for transforming the input documents given in text form into vectors. The structure of this approach is shown in Figure 4.1. We consider each TCP session as a single document, consisting of sentences in the form of the described packet features. Doc2Vec is trained on a corpus of multiple input documents, transforming them into document vectors. A logistic regression classifier is trained using the associated labels in the form of application names to predict the related output classes. Le and Mikolov state that Doc2Vec performs well on textual pattern processing and is able to derive context from arbitrary sequential data. The used paragraph vectors representing the document semantics are "a strong alternative to bag-of-words and bag-of-n-grams models"[15, p. 1195], being on par with state-of-the-art methods in their studies. In [15], the error rate of the experiment evaluation outperforms all comparable methods. As Doc2Vec creates the vector representation directly from textual input, no preceding tokenisation is required. It considers pattern information contained in TCP sessions, as the document context influences the weights during the training process of each network packet within. For unseen data, the trained model is able to infer a new vector in the document vector space by contextual comparison to existing ones. The comparison is performed by measuring the cosine similarity to the surrounding vectors. In order to use the resulting measurements for performing the classification task on arbitrary sequential data, Le and Mikolov recommend combining their algorithm with a logistic regression classifier as the predictor.

Doc2Vec can be invoked using either PV-DM or PV-DBoW. Both algorithms capture the context information of each paragraph. The former considers the paragraph context based on the surrounding words in order to predict the following word. The

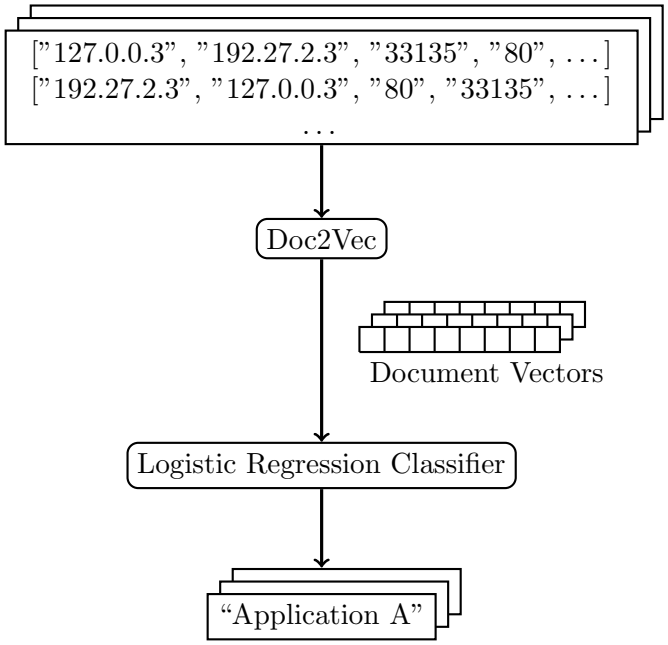


Figure 4.1: Using Doc2Vec for input transformation followed by a logistic regression classifier.

similarity of the prediction to the ground truth is measured, and the weights of all involved vectors are adapted according to the result. Thus, the order of words influences the prediction and paragraph context. The latter algorithm does not consider the word ordering, as the contextual information of the paragraph is learned without depending on the surrounding words. Instead, the paragraph vector predicts randomly sampled words from the paragraph. Although the authors recommend the usage of PV-DM, we compare both methods in order to find the appropriate algorithm for a network traffic dataset. As the order of packet features is artificially created, relying on the placement of words can result in unintentionally biasing the model.

4.4 Approach 2 - Dedicated Classification Network

This approach makes use of two neural networks with different internal layer structure for performing the multi-class classification task. Since a neural network cannot handle the document structure used in the first approach, all sessions have to be transformed into a token representation. In order to do so, a tokeniser replaces all words in the document sentences by an integer. The resulting documents then consist of encoded feature vectors, which can be used as input for an embedding layer. The afterwards

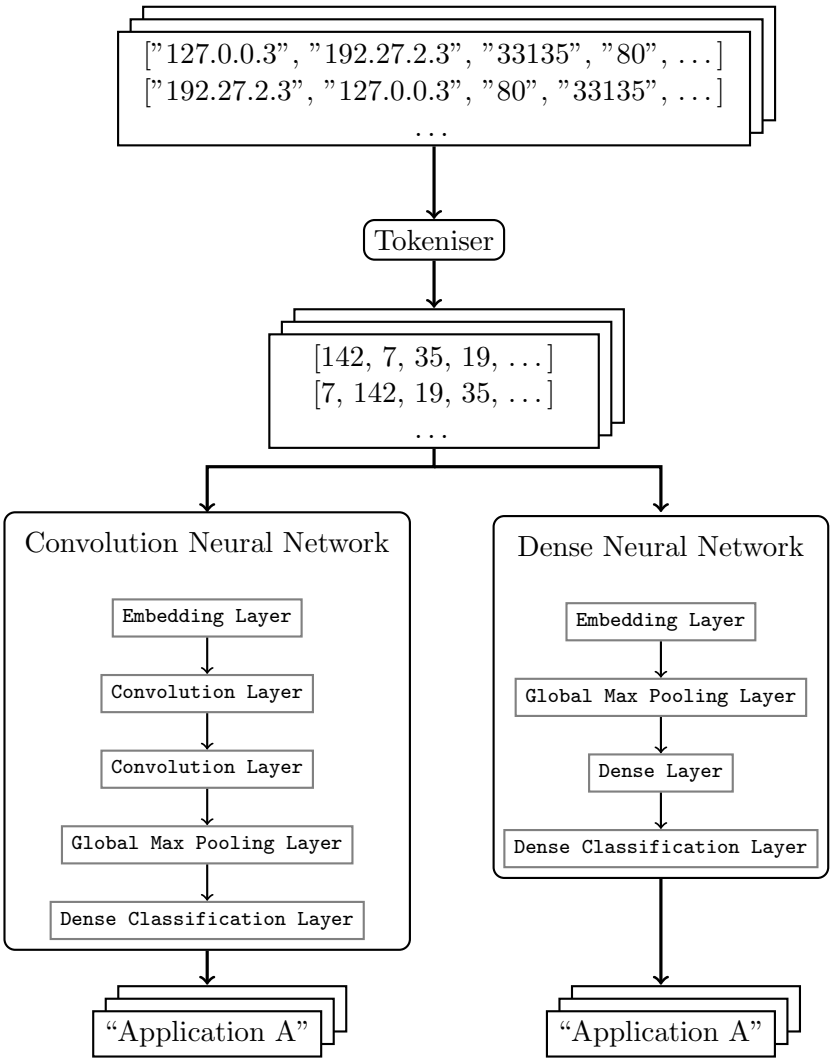


Figure 4.2: Classification using a convolution neural network and a dense neural network.

following layers depend on the specific network configuration. Similar to the tokenisation of features, the associated information of labels is removed using one-hot encoding.

The first network in this approach is a convolution neural network, in which a convolution layer follows the initial embedding layer. During the training phase, the convolution layer empirically fits a suitable filter function on the input features. The filter function depends on the size of the filter kernel and the output dimensionality. It is adapted according to the classification performance of the network. Another convolution layer is applied with a different filter function to shape the vectors into a form suitable for the classification layer. The classification is performed using a dense layer, of which the number of nodes is equal to the distinct labels contained in the dataset. Afterwards, a global max pooling is applied to the resulting vectors. This layer reduces the vector dimensionality, as it extracts the global maximum of the vector independently from its position. In comparison to other pooling techniques, global max pooling reduces overfitting as there are no layer parameters to optimise in dependency of the dataset. Although convolution neural networks tend to be more common in the field of image recognition, recent work has shown that this type of model performs well in natural language processing tasks too [40, 7, 17, 26, 42, 28, 4].

The second neural network is a dense classification model. Instead of a convolution model following the initial embedding layer, global max pooling is applied directly. Afterwards, a dense hidden layer is used in the network. The number of nodes in the hidden layer is configurable and chosen appropriate to the dataset. For performing the final classification, another dense layer is utilised similarly to the convolution neural network. The structure of this network resembles a multi-layer perceptron. The two network configurations and the prior tokenisation are displayed in Figure 4.2.

During the training of both models, we optimise all layer parameters using Adam and compute the loss of each training cycle by utilising the binary cross-entropy loss function. These choices represent common best practice values for neural networks. All convolution layers and the dense hidden layer use ReLU as the neuron activation function. In the dense classification layer, we instead use softmax, since this function limits the values of the prediction vector to the range between 0 and 1. For converting the predicted vector into a classification label, one-hot decoding is performed on the prediction vector in order to transform the result into the corresponding class label. The neural networks of this approach are designed as a comparison reference to the first approach, which is the reason that the structure has been kept simple intentionally. The possibility to compare both approaches allows us to comprehend whether a bad

performing model results from an inappropriate dataset or ill-chosen network parameters.

4.5 Approach 3 - Doc2Vec with External Classifier

The last approach extends the idea of the first one, as can be seen from the structure shown in Figure 4.3. Similar to the approach described in Section 4.3, Doc2Vec is used to compute the document vector representation of textual documents containing single sessions. Conversely, the afterwards following predictor is changed to a dense neural network. Since the structure of document vectors resembles tokenised word vectors, an embedding layer is used to process the input. As the logistic regression classifier from the first approach can be prone to overfitting, a global max pooling layer is utilised in the same way as in the second approach. Directly after the global max pooling layer, a dense hidden layer allows more fine-grained control on the training parameters in comparison to the logistic regression classifier. The last layer of the network is a dense classification layer using a softmax activation function to perform the prediction. Analogous to the first approach, the contextual information of the associated labels is not removed, since no encoding is performed. Furthermore, the words contained in the session documents are not tokenised, conversely to the second approach. As the tokenisation of words is a simple mapping between words and integers, it does not take larger patterns between the words into account. In contrast, the document vectors created with Doc2Vec preserve the textual context of the input.

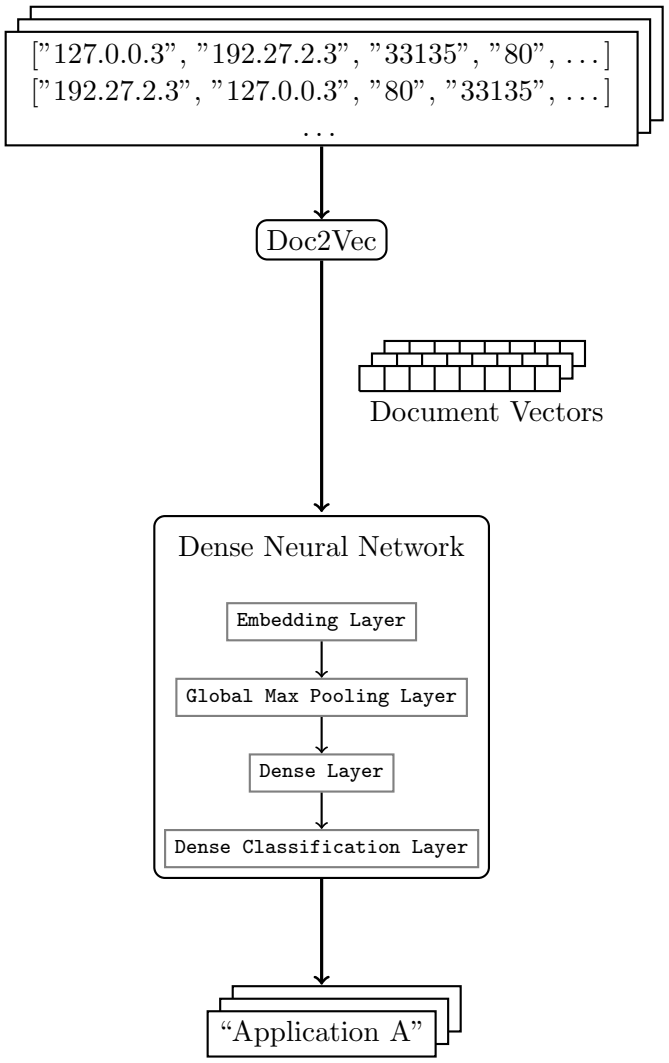


Figure 4.3: Transforming documents into vectors using Doc2Vec, prior to performing the classification with a dense neural network.

Chapter 5

Implementation

This chapter describes the practical aspects of the thesis. We begin by giving an overview of the different modules involved and the interfaces used for communication between them. Each model is then explained in detail, starting with the used tools for capturing network traffic. Afterwards, we describe the necessary preprocessing steps in order to use the data in the machine learning models. The implementation of the models, the training process and the performance evaluation are detailed in Section 5.4. This chapter finishes by discussing the detection of applications and application categories based on freshly captured data.

5.1 Toolchain

The implementation is separated into four modules, namely *traffic capturing*, *data preprocessing*, *model training and evaluation* as well as *application classification*. Each module consists of one or multiple process steps and is explained in more detail in the following sections of this chapter. For capturing network traffic, we use a modified version of the open-source firewall Netguard. This application allows us to gather traffic from multiple Android devices and determine the responsible application for each packet. The resulting collection of network traffic files in the PCAP format and the related application name files serve as input for the data preprocessing module running on a workstation.

This module consists of multiple Python scripts and starts by merging all traffic capture files into a single one. Likewise, the same script concatenates all files containing

the names of the recorded applications. In the following data preprocessing script, the resulting PCAP file is split into single TCP sessions. All essential features are extracted from the packets of each session and associated with the corresponding application label from the related file. The resulting labelled feature vectors are serialised into multiple *Comma Separated Value* (CSV) files, preserving the session structure. On this textual representation, additional scripts can be applied to filter out undesirable applications or alter labels or features. In the third module, the serialised session files are used to train the machine learning classifier. The resulting model is stored on disk, allowing to use it in the detection module afterwards without performing the time-consuming training again.

Any newly captured traffic that should be classified using the predictor in the detection module has to be serialised in the same way as the training dataset. The predictor then loads the model and outputs the predicted similarities of the input traffic to the trained labels. In Figure 5.1, the structure of the modules and the interfaces are shown. A benefit of this module separation is that single components can be exchanged without influencing others. For example, the machine learning model used can be replaced by another approach without the need to adapt the preprocessing steps. Although the implementation heavily relies on Python scripts, the programming language of the components can be exchanged as well.

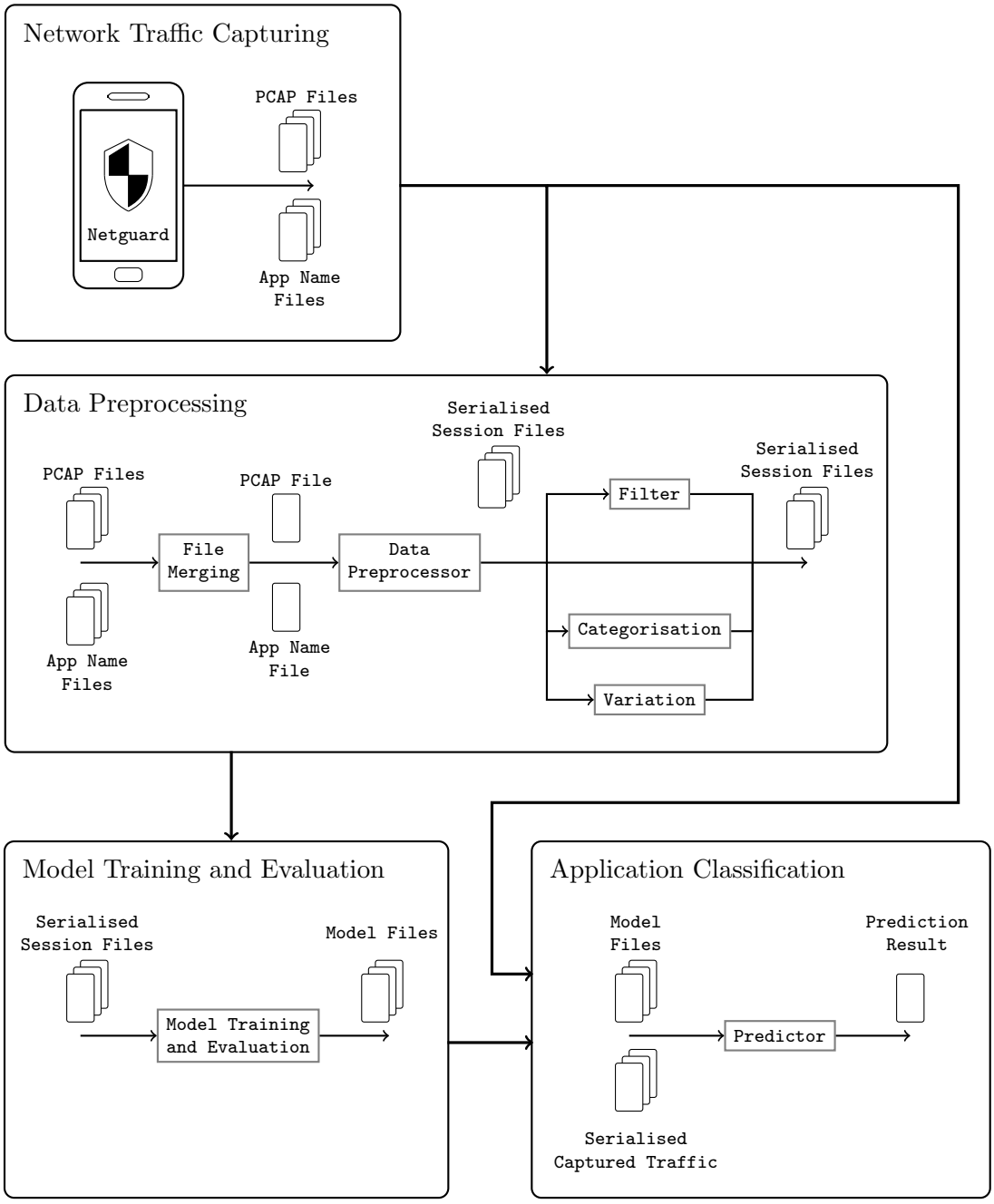


Figure 5.1: Structural overview of the module components.

5.2 On-Device Capturing of Network Traffic

In order to capture network traffic of Android applications, we extended the capabilities of the open-source app Netguard. This device-local firewall is installed as system-wide VPN service, meaning that all packets pass the local VPN server. As this server terminates eventual tunnel encryption protocols, the packet metadata is available in plain text. Netguard logs all passing network traffic and is able to export the log in PCAP format. Since the ethernet header is not exported in the original implementation of Netguard, the exported file cannot be used directly for traffic analysis. For this reason, we extended the application to extract the ethernet header information of each IP packet and prepend it to every exported packet. Additionally, we log the currently active application when a packet is encountered. The application name is exported to a dedicated file along with a microsecond timestamp. This timestamp is added to the packet as well, allowing a later mapping.

5.3 Preprocessing of Raw Data

For performing the necessary preparation of traffic data, the preprocessing module requires the captured traffic to be in PCAP format and the application names in a textual representation. By using the PCAP format, the preprocessing steps can be performed on traffic captured on the device as well as on already recorded traffic of a known application. In the latter case, a file containing the mapping of application names has to be created manually. As the first step of the module, multiple PCAP files are merged into a single one. Since PCAP is a binary format, we use the merge capability of the *tshark*[1] library. Additionally, all application name files are concatenated into one. In the next step, the data preprocessor script loads both files and splits the network traffic into single TCP sessions. The separation is performed using the PcapPlusPlus library[2]. Each session is then processed in parallel. First, the source and destination IP addresses and TCP ports, the TCP flags and header size, as well as the total IP packet size is extracted from the available information and stored in a dedicated Python class. Dynamic features like private IP addresses and dynamic TCP ports are replaced by static tokens, in order to decrease noise contained in the dataset. For static IP addresses, the whois database is queried. If the response contains a subnet range, the static address is

[1] https://www.wireshark.org/docs/man-pages/tshark.html
[2] https://github.com/seladb/PcapPlusPlus

replaced by the subnet range. This method allows representing eventual load-balancers or CDNs as a single destination. The timestamp of each packet is used to associate the corresponding label, which is loaded from the application name file and accessible from all sessions. All labelled objects of a session are finally serialised into a single CSV file, preserving the order of packets within the session. As captured traffic can contain sessions that are out of scope, the textual representation allows removing those. A filter can be applied to remove all sessions of apps that are not contained in a whitelist. This method can reduce noise in the dataset and allows to detect applications with similar traffic, as such a scenario could confuse the reasoning of the machine learning model.

Especially for a small amount of captured network traffic, it can be beneficial to create a more diverse dataset artificially. In order to do so, a configurable amount of all sessions or sessions of a single application is duplicated. The features of packets in the duplicated sessions are then altered randomly inside a given range. All altered sessions are serialised to additional files and considered as new sessions. This method can decrease the possibility that the model bases the decision on a single feature of a packet or ignore others. Additionally, the session labels can be replaced in order to group applications into app categories. If a prediction is performed for applications that are not in the training set, only the similarity to known apps is calculated, which can be misleading. To counter this, replacing the application name by predefined categories provide a more generalised dataset to the cost of fine-grained detection capabilities.

5.4 Model Training and Evaluation

This module creates different machine learning algorithms, depending on the chosen approach. The machine learning models are implemented as a single Python script, in which the used approach and the respective parameters can be configured. The input for this script is the directory containing the serialised session files. As each approach provides dedicated functions to save the trained model to disk, we rely on these. For this reason, the output files of this module differ between all approaches.

5.4.1 Approach 1 - Doc2Vec

In this configuration, the script starts by transforming the session files into documents. From each packet within a session, a tagged sentence object is created, consisting of the ordered list of features and the associated labels. Every session is considered a

document, with all transformed sessions together building the document corpus. The document corpus is then split into a training and test set. The sessions are divided randomly, according to a preconfigured ratio.

We use Gensim[3], an open-source Python library implementing several popular machine learning algorithms for natural language processing and information retrieval, to create a Doc2Vec model. The model is initialised with the vocabulary built from the words contained in the training set. Afterwards, this dataset is used to train the model for a configured number of epochs. During this process, the training set itself is randomly split into smaller training batches. No validation set is used, as the Gensim Doc2Vec implementation provides no support. The trained Doc2Vec model is used to infer vectors from input documents, as the sentences are transformed into the learned vector space. A logistic regression classifier is then fit on the resulting feature vectors. We rely on the classifier implementation of the Scikit-Learn[4] library, an open-source collection of data analysis tools. Besides specifying the number of epochs for the training process of the Doc2Vec model, various other parameters can be altered. The *min count* parameter determines the minimal occurrence of a word in order to influence the model. If the *dm* flag is set, Doc2Vec uses PV-DM instead of PV-DBoW. For both algorithms, the size of the sliding window is configurable with the *window* parameter. The dimensionality of the vector space used internally in the model can be specified by changing the *vector size*. Similarly, the performance of the logistic regression classifier depends on the inverse regularisation parameter *c*. This value is represented by a positive floating-point number and controls the regularisation strength of the classifier during fitting.

For evaluating the model performance, we use the metrics of Scikit-Learn. For all documents in the test set created in the first part of this approach, the Doc2Vec model infers vectors. The logistic regression classifier predicts labels for these vectors, which are compared against the ground truth in the form of the original labels of test set documents. Based on the outcome, we calculate precision and recall for the classifier. From this, we can compute the resulting F_1 score. Additionally, a confusion matrix for all classes visualises the comparison outcome. The trained and evaluated Doc2Vec model is serialised to a JSON file, whereas the logistic regression classifier is stored using Pickle[5]. Pickle is a Python serialiser, which saves arbitrary objects in an automatically determined binary format.

[3]https://radimrehurek.com/gensim/models/doc2vec.html
[4]https://scikit-learn.org/stable/
[5]https://docs.python.org/3/library/pickle.html

5.4.2 Approach 2 - Dedicated Classification Network

The second approach is built on top of the Keras[6] framework, which provides an interface for implementing different machine learning backends and unifies their syntax. The structure of this approach is shown in Figure 5.2. Similar to the Doc2Vec configuration, the input documents are deserialised into a list of document objects, each containing single sessions of network packet features and associated labels. The document objects are shuffled and split into the training and test dataset according to the configured ratio.

A tokenisation has to be performed on the packet features of both datasets. Keras includes a tokeniser in the preprocessing module, which maps the feature words to an integer. Since the features of a network packet are already in their base form and do not have suffixes or prefixes, the stemming included in the tokeniser does not alter the datasets. Likewise, the labels attached to all packets are encoded using the categorical label encoder of the Scikit-Learn library. This encoder creates a one-hot encoding, which is a mapping from the labels to distinct binary vectors. The values of the vectors are set to 0 with the exception of exactly one value that is set to 1. The position of this 1 is unique in the set of all encodings.

We use the sequential model included in Keras to combine the layers of the classification network linearly. The first layer that is part of the sequential model is an embedding layer. The length of the input vector is chosen equal to the size of the feature word vector, while the input dimension corresponds to the size of the vocabulary. This size is given by the highest integer used during the tokenisation process. Furthermore, the dimensionality of the output vectors relates to the input size of the following layer and is configurable. Depending on the configuration, different layers are used afterwards. The first network configuration creates a convolution classification network, whereas the second constructs a dense classification network.

In the first configuration, a convolution layer is added to the sequential model after the embedding layer. The output of this convolution layer is defined by the size of the filter kernel and the number of filters, which directly corresponds to the output dimensionality. Both parameters of the convolution layer are specified in the configuration file. A global max pooling layer follows, further reducing the feature vector dimensionality prior to another convolution layer. In this layer, the kernel size is fixed to the same size as was used in the first convolution layer. The number of filters, on the other hand, is set to a lower number, compared to the previous convolution layer. In the dense

[6] https://keras.io/

classification layer, a global max pooling layer is applied directly after the embedding layer. Afterwards, a dense layer is used as a hidden layer. The number of neurons can be freely specified in the configuration. The final layer of both network structures is a dense classification layer. The amount of neurons in this layer is fixed to the number of distinct labels in the dataset. All layers except the classification layer use the ReLU activation function, whereas the last layer relies on the better-suited softmax function.

After the layer definition, the sequential model is compiled into a network using the Adam optimiser and binary cross-entropy as a loss function. The model is trained for a given number of epochs, in which the training dataset is again divided into smaller batches. A configurable amount of the training set is used as a validation set, reducing the number of available training examples in order to provide a more objective measurement of the efficiency of the learning process. As the last step, the performance of the model is evaluated on the test set. For every feature vector, a classification is performed. This classification predicts the most likely label for each feature vector. The label encoder inversely transforms the results, as the predictions are one-hot encoded. After this step, the transformed labels are compared to the ground truth, which is represented by the labels associated with the feature vectors. Based on this comparison, precision, recall and the F_1 score are computed using the Scikit-Learn library. Additionally, a confusion matrix is created to display the performance on a class level. Prior to the classification layer, the processed feature vectors are available as raw data points. This high-dimensional data points can be visualised in Tensorboard[7]. For creating the visualisation, the data points are exported to metadata files. The training loss and accuracy are visualised in a line graph to provide further insight into the effectiveness of training parameters.

In order to reuse the trained model at a later point in time, the necessary objects are saved to disk. The structure of layers in the Keras sequential model is written into a JSON file, whereas the layer weights obtained in the training phase are saved in *Hierarchical Data Format 5* (HDF5). For preserving the mapping information of features and label encodings, the used tokeniser and label encoder are serialised with Pickle.

[7] https://www.tensorflow.org/guide/summaries_and_tensorboard

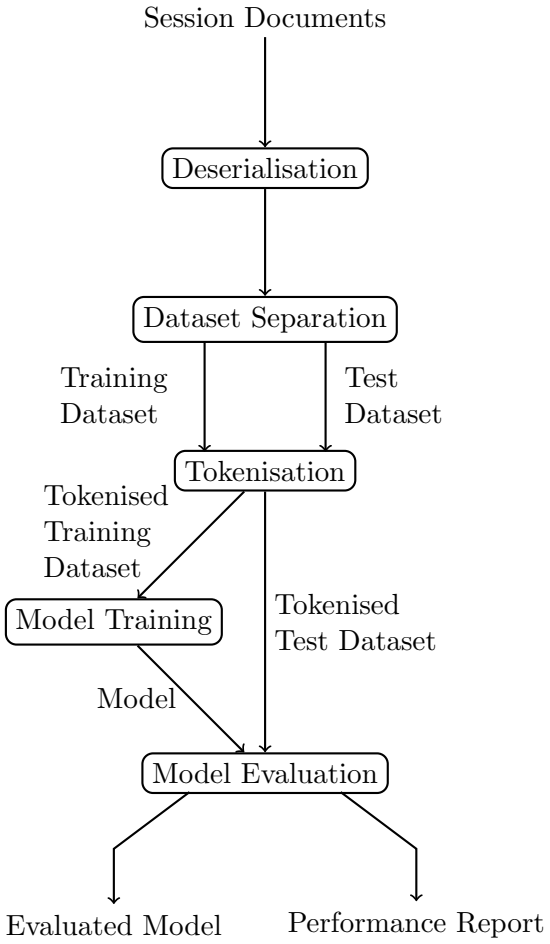


Figure 5.2: Structural overview of the dedicated neural network components.

5.4.3 Approach 3 - Doc2Vec with External Classifier

In the third configuration, elements of the previous two approaches are combined. As before, the script transforms the sessions from CSV format into document objects. They are split into the training and test dataset according to a configurable ratio in a random manner. Similar to the first script configuration, the Doc2Vec implementation of Gensim is applied to the first dataset for a specified number of epochs.

After the training of this model, a document vector for each sample in the training set is created using the learned vector space. Figure 5.3 gives an overview of this approach and shows the differences to the previous one. In contrast to the second approach, no tokenisation of features or encoding of associated labels is performed. Instead, the document vectors are directly used as input for the afterwards following Keras sequential model, related to the dense classification network of the previous configuration. The initial layer of this model is an embedding layer with an input vector size equivalent to the length of the document vectors and the input dimension corresponding to the number of document vectors. Global max pooling is applied to reduce the risk of overfitting prior to a dense hidden layer with a configurable number of neurons. For the activation function, the dense layer uses ReLU. The last layer of the sequential model is another dense layer. Other than in the previous one, a softmax activation function is chosen. In order to perform the classification, the amount of neurons is set equal to the number of distinct labels. The layers are compiled using Adam as an optimisation strategy while relying on the binary cross-entropy function for computing the loss. During the following training of the sequential model, a configurable part of the training dataset serves as a validation set. The training itself is performed for a specified number of epochs on single batches of the training set. The number of batches used in each epoch can be set in the configuration.

Afterwards, the model performs a classification of the test dataset in order to measure the performance. The prediction values can be compared directly to the ground truth. The number of correct classifications and false predictions are displayed in a confusion matrix. Based on these results, the Scikit-Learn library is used to compute precision, recall and the F_1 score. The Keras model layer structure allows extracting the raw data points prior to the classification layer. This information is written to a metadata file and visualised in Tensorboard for gathering insight whether the processed document vectors are clearly separable. Furthermore, the effectiveness of the training process is visualised as a line graph based on the results of the loss function and the

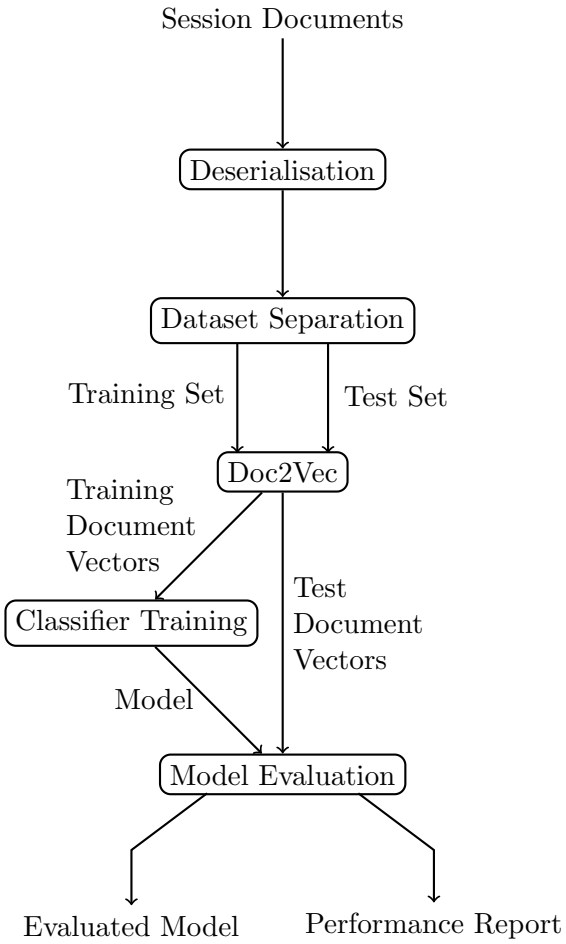


Figure 5.3: Structural overview of the components of Doc2Vec with external classifier.

computed accuracy during each epoch. In the same way, as in the previous approaches, the trained models are saved for later predictions. The Doc2Vec model and the layers of the sequential models are exported as JSON files. Similar to the previous configuration, the weights of the Keras layers are stored as HDF5 file.

5.5 Detecting Applications and Categories from Unseen Traffic

The last module of the toolchain consists of a single Python script predicting the source application of newly captured network traffic. For performing the classification, this

module relies on the machine learning models from the previous step. This requires that the same preprocessing steps are applied to the input traffic as needed prior to the training of the models. For this reason, the traffic is transformed into documents containing single sessions. All sessions consist of sentences that, in turn, are made out of packet feature words. The transformation is performed using the script described in Section 5.3. Afterwards, the prediction module deserialises the resulting session documents into session objects.

The classification process depends on the configuration of the previous module, which is determined automatically based on the model files specified as input. If the Doc2Vec model and the logistic regression classifier are given, the first configuration is assumed. No further transformation of the input documents is performed, as the Doc2Vec model creates the document vector representation of all session documents. The logistic regression classifier then predicts the most probable labels based on the document vectors.

The presence of the Keras model structure and the related weights in combination with the tokeniser and label encoder indicates the second configuration. The embedding layer of the model structure requires additional preprocessing of the document objects. This is performed by mapping all feature words in these documents to their integer counterparts using the tokeniser. The Keras model is recreated from the specified network structure and the given weights, in order to classify each session. Since the prediction results are one-hot encoded, the label encoder performs an inverse transformation to restore the readable class label. In the case that the Doc2Vec model as well as the Keras layer structure and the weights are given, the third configuration is expected. A tokeniser or label encoder is not required, as Doc2Vec transforms the document object into document vectors. The prediction is performed using the document vectors as input for the Keras classification network, which outputs the most probable label for each document.

If the source application of the newly gathered traffic is not in the dataset that was used to train the model beforehand, it is predicted to be of the most similar class. In the case that multiple documents are given as input, the prediction is performed for every session, and the frequency of all resulting class labels is computed. The frequencies are collected in a probability vector, which gives the most likely source applications for the captured traffic.

Chapter 6

Evaluation

In this chapter, we assess the performance of the implemented machine learning models. The first section describes the process of capturing network traffic from an Android device. As the performance of machine learning models depends partly on the underlying data, we created four datasets from the gathered traffic for comparing the models. These datasets differ in the used labels, as well as the ratio between network packets and classes. The structure of all datasets is explained in detail in the following section. We then discuss the performance of all approaches on these datasets. For each approach, we start by detailing the chosen model parameters before examining the results. First, we describe the outcome of Doc2Vec with a logistic regression classifier, followed by the performance values of the dedicated classification neural networks. The result of Doc2Vec with a dense neural network classifier is discussed as the last approach. Afterwards, we briefly explain the encountered limitations and propose enhancements. In the last section of this chapter, we examine the reasons for the performance differences between the three approaches and summarise the results.

6.1 Datasets used for Evaluation

We capture the network traffic for creating multiple evaluation datasets on an Android smartphone running Android 5.1.1. The device is connected directly to a wireless access point and has a customised version of Netguard installed, which is configured to allow all network traffic to pass. Although Netguard would provide the possibility to limit the scope to a chosen subset of apps by changing this setting, we log network packets of all

installed applications. This allows us to perform a more detailed analysis at a later point and to create multiple datasets afterwards, which can consist of different applications.

In order to evaluate the machine learning models in a realistic environment, we aim to reflect the interaction of real users in the network traffic. For this reason, we choose not to use monkey tools for traffic generation since the focus of these programs lies on testing the crash robustness of applications. In the process of testing an app, monkey tools send random user events to an application in order to provoke errors. Even though these tools are easy to configure, the outcome does not necessarily reflect human decisions and the results following particular input are seldom reproducible. Additionally, monkey tools in their simpler form are not able to enter credentials and fail at simple verification methods. As an alternative, input events can be scripted to overcome these limitations. A drawback of this approach is that scripted scenarios have to be defined manually. With a small number of scripted scenarios, this approach results in repeated inputs which artificially introduce recurrent patterns in the network traffic data.

Because of the described drawbacks of monkey tools and scripted scenarios, we collect network traffic by monitoring connections during the interaction of users with the applications on their phone. Netguard logs the produced traffic and exports the resulting log files. Observing the interaction of real users guarantees that any patterns contained in the network traffic can indeed occur in everyday use as well. Since users tend to switch between multiple applications during interaction with their smartphone, several starting functions are triggered in each phase. Most startup routines require loading data from servers, resulting in several distinct sessions. However, a downside of this collection method is the effort required to generate a large amount of traffic for creating the dataset. Using the described way, we gathered 95MB of network traffic in 92 PCAP files, representing the base for the datasets described in the following subsections.

6.1.1 Application Dataset

The first dataset consists of the recorded traffic on a per-application level. For this reason, the package identifier of each app is chosen as the label. The dataset consists of 13 applications, as summarised in Table 6.1. These applications are chosen in a way that for each app, at least one other application with a similar purpose is contained in the dataset. The collected network traffic logs consist of 22 902 sessions originating from the chosen applications, with a total of 1 197 495 monitored network packets. Although all applications were executed for roughly the same amount of time, the amount

Package Identifier	Sessions	Packets
org.telegram.messenger	7648	57429
com.whatsapp	5142	40473
com.twitter.android	2625	169226
at.willhaben	2285	131992
com.spotify.music	1034	62195
com.melodis.midomiMusicIdentifier.freemium	897	36082
org.thoughtcrime.securesms	751	10563
air.com.hypah.io.slither	607	133253
tv.twitch.android.app	497	395696
com.robtopx.geometryjumplite	453	39639
com.valvesoftware.android.steam.community	426	76201
at.oebb.ts	374	28619
com.Slack	163	16127

Table 6.1: A summary of the first dataset with application identifiers as label.

of traffic recorded in this timeframe differs. This difference in sessions and packets can be seen from the summary as well. The disparity can result from the implementation design and the network libraries used in each application. For example, applications like *tv.twitch.android.app* and *com.spotify.music* produce a large number of packets in relation to their established sessions during streaming. Since both apps stream multimedia content, this amount of packets results presumably from the large size of transmitted media. During the evaluation process, neither equalising the number of packets nor the number of sessions changed the outcome significantly. For this reason, we decided not to reduce the available amount of traffic. Interestingly, only 163 sessions of the messenger *com.Slack* are contained in the traffic log, although the app was used in the same extent as other messaging applications. Upon closer inspection, we found out that *com.Slack* seems to use persistent WebSockets. As these are event-driven in both sides, the share of polling calls is probably reduced, leading to fewer transmitted packets and established sessions. We performed the necessary preprocessing of this dataset on a cluster, taking advantage of the parallelisation capabilities. On an Intel Xeon E5-2699 v4, the process took about two hours using the 88 available cores.

Categories	Sessions	Packets
messenger	13704	124592
shop	3085	236812
socialmedia	2625	169226
streaming	2428	493973
game	1060	172892

Table 6.2: The resulting dataset after grouping the applications into categories.

6.1.2 Category Dataset

This dataset is a modification of the previously described application dataset. Since the application dataset consists of single sessions documents for each app, the categorisation module discussed in Section 5.3 is applied. This module substitutes the application names with the related category name. In this case, the module maps all applications to five categories, according to the purpose of each app. For this reason, *com.Slack*, *org.thoughtcrime.securesms*, *com.whatsapp* and *org.telegram.messenger* are mapped to the category *messenger*, whereas, the music streaming app *com.spotify.music*, the music detection service *com.melodis.midomiMusicIdentifier.freemium* and the video streaming application *tv.twitch.android.app* are belonging to the class *streaming*. Although the app provides other features as well, the main objective of *com.valvesoftware.android.steam.community* is the selling of video games. Thus, this application is considered as part of the group *shop*, similar to the national railroad ticket sale app *at.oebb.ts*. The labels of session documents related to *com.twitter.android* are replaced with *socialmedia*, while *com.robtopx.geometryjumplite* and *air.com.hypah.io.slither* represent the category *game*. Table 6.2 summarises the dataset. As a result of the mapping between categories and application names, the total number of sessions, as well as the number of packets in the dataset remained the same. Since this dataset is based on the already preprocessed application dataset, the afterwards applied categorisation takes only a couple of minutes on a single workstation.

6.1.3 Generating Artificial Data

Due to the high effort of manual traffic gathering, we extended the previously described datasets with artificial packets. With these datasets, the generalisation capabilities of the chosen models and their tendency to overfit can be evaluated even though the number

Categories	Sessions	Packets
org.telegram.messenger	9886	73237
com.whatsapp	6720	52990
com.twitter.android	3440	217659
at.willhaben	2994	175646
com.spotify.music	1328	76877
com.melodis.midomiMusicIdentifier.freemium	1167	47149
org.thoughtcrime.securesms	979	13204
air.com.hypah.io.slither	804	178387
tv.twitch.android.app	624	512565
com.robtopx.geometryjumplite	590	51739
com.valvesoftware.android.steam.community	562	102523
at.oebb.ts	479	35807
com.Slack	211	21130

Table 6.3: The summary of the extended application dataset.

Categories	Sessions	Packets
messenger	19173	173552
shop	4324	334959
socialmedia	3656	233527
streaming	3428	765253
game	1481	237791

Table 6.4: The category dataset after extension with additional sessions.

of samples is restricted. We apply the *variation* script of the preprocessing module on the application and category dataset. This script increases the variety of features in the respective dataset by copying a random share of each class and altering the value in a given range. We choose to use the default value and vary 30 per cent of each session. With this amount, the application dataset is enlarged to 29 784 sessions containing 1 558 913 packets, while the altered category dataset consists of 32 062 sessions with 1 745 082 packets. Throughout this work, we refer to the first dataset as extended application dataset and the latter as extended category dataset. Table 6.3 shows the distribution of sessions in the former, whereas Table 6.4 summarises the extended category dataset. Similar to creating the category dataset, the extension process only takes a few minutes on a workstation.

Configuration	1	2	3	4
Vector Size	300	300	**30**	300
Training Iterations	20	**50**	20	20
Inverse Regularisation	10^4	10^4	10^4	$\mathbf{10^6}$

Table 6.5: The altered configuration parameters of the first approach.

6.2 Model Performance

All approaches described in Chapter 5 are evaluated using the four datasets with different parameter configurations. In the beginning, we initialise these configurations with a default parameter set and measure the performance on each test dataset. To observe the influence of different parameters on the classification results, we alter the respective values between multiple executions. The resulting precision, recall and F_1 scores are compared between each configuration and parameter set. For every approach, we choose four parameter configurations that emphasise clearly differentiable aspects of the model.

6.2.1 Approach 1 - Doc2Vec

In this approach, we evaluate the performance of Doc2Vec with PV-DM and PV-DBoW on the previously described dataset. To compare the results, we create four parameter configurations, with the first set representing the default values. For the parameters of the first set, we set the size of the *window* to 8. This number corresponds to the length of a sentence in a document. We choose a *min count* of 0 and a *vector size* of 300. The training on the dataset is performed for 20 iterations. For the second one, we increase the number of training iterations to 50, whereas we decrease the *vector size* to 30 for the third configuration. In the fourth set, we change the *inverse regularisation* parameter of the logistic regression classifier from 10^4 to 10^6. The altered parameters of the configuration are shown in Table 6.5.

For all experiments, the performance of Doc2Vec with a logistic regression classifier is quite low. As shown in Figure 6.1, neither precision nor recall exceeds 50 per cent, regardless of the chosen parameters. This could be an indicator that either Doc2Vec or the logistic regression classifier is not suitable for this type of data or that there are not enough data samples for each class. As the classifier achieved only marginally better performance on the less fragmented category dataset than on the other datasets, the former seems to be more likely. Using PV-DM with default parameters, the average

precision and recall on the application dataset reach its minimum. On the extended application dataset, slightly better results are achieved. In contrast to the application datasets, the category and extended category dataset perform clearly better with a precision of 0.39, respectively 0.44. Increasing the training iterations in the second configuration improves the performance on all datasets except the category dataset. On the application and extended application dataset, the precision is enhanced by six per cent, while the extended category dataset increases by five, representing the best performance of this approach. The precision on the category dataset instead decreases by seven per cent, from 0.39 to 0.32. For the first three datasets, no significant recall improvement can be seen, whereas the value for the category dataset slightly decreases. It seems that the classifier benefits especially on high fragmented datasets from increased training iterations. Only the application dataset draws advantage from decreasing the vector size in the third configuration. The precision improves to 35 per cent with a recall of 0.44, while the classification performance on all other datasets declines. On the category dataset, the third configuration yields with 37 per cent respectively 0.47 a nearly equal precision and recall as the first parameter set. The precision for the extended application dataset drops to 20 per cent while the recall is reduced to 0.33, the minimum values of this model. For the precision on the extended category dataset, a decrease to 0.36 with a slightly enhanced recall of 0.46 can be seen. It could be possible that the larger vector size includes features related to another context in each training vector. The decrease seems to reduce this environmental noise, resulting in a better training process in the case of the application and extended application dataset. The increase of the inverse regularisation parameter of the logistic regression classifier results in an improvement of precision on the application dataset, as it increases to 33 per cent with 0.4 recall. On all other datasets, this configuration yields comparable values to the default parameter set.

The category dataset especially benefits from using PV-DBoW, as the precision in the first configuration achieves 44 per cent with a recall value of 0.45. All other datasets obtain lower results, with precisions between 0.32 and 0.35 for the application and extended category dataset and 0.24 for the extended application dataset. Except for the extended application dataset, all recall values are in a range between 0.39 and 0.47, while the former achieves values from 0.33 to 0.4. The classifier accomplishes better results on the application and category dataset than using PV-DM with the same parameter set. On this datasets, the model seems to benefit from the lack of dependence on the order of surrounding words in PV-DBoW. Interestingly, when increasing the performed amount of training in the second configuration, the precision on all datasets

Configuration	1	2	3	4
CNN Filter Number	10	10	10	**8**
DNN Number of Neurons	10	10	10	**15**
Embedding Dimension	25	25	**40**	25
Training Epochs	10	**30**	**5**	10

Table 6.6: The parameter configurations for the convolution and dense neural network.

decreases. The category dataset performs best with a precision of 0.32, whereas the value on the application dataset decreases to 0.28. On both extended datasets, a precision of 0.23 is achieved. As the recall drops as well, this decrease most probably results from focussing on single features during the training process. If these features are not contained in the test dataset, the model cannot accurately predict the corresponding class. Decreasing the vector size in the third configuration results in a slight performance boost of the application and extended application dataset by four respectively one per cent. As before using PV-DM, datasets with fewer packets per class seem to benefit from the reduce in surrounding context information with these parameters. On the category dataset, the precision decreases marginally to a value of 0.42, whereas the drop to 0.26 on the application dataset is even higher. The increase of the inverse regularisation parameter of the logistic regression classifier in the last configuration does not considerably change the performance on the application and category dataset. On the extended application dataset, a small improvement to 0.31 can be seen, whereas the precision of the extended category dataset decreases by five per cent to 0.3. Similar to the model using PV-DM, it seems that the inverse regularisation parameter does not influence the prediction performance of the model significantly.

In Figure 6.1, the resulting F_1 scores of the performed Doc2Vec evaluation is shown. The fluctuation in these results does not clearly indicate a pattern. By comparing the performance on all datasets, this approach seems to be more appropriate to classify the category dataset. Although the F_1 scores of this dataset are only marginally better than those of the application dataset, they remain more stable. The category dataset evidently outperforms both artificially extended datasets, as the F_1 score of only two configurations exceeds the worst result of the former.

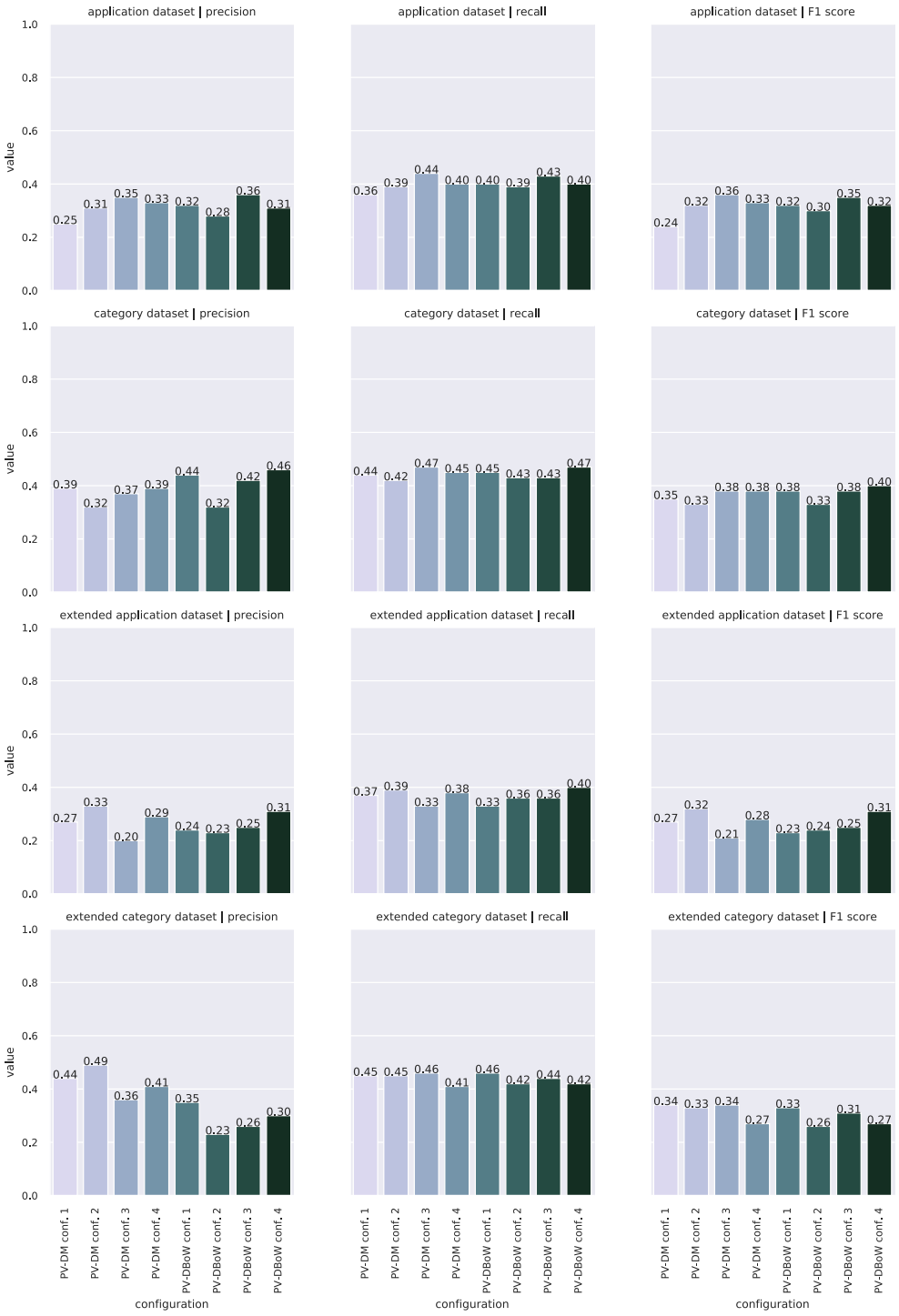


Figure 6.1: Precision, recall and F_1 score of the different Doc2Vec configurations.

6.2.2 Approach 2 - Dedicated Classification Network

The default parameters for the evaluation of the second approach depend on the network layer structure. For the convolution network, we choose a *filter number* of ten and a *kernel* size of three for the first convolution layer. Although the second uses the same *kernel size* as the first one, the *number of filters* is reduced to five. For the dense neural network, the *number of neurons* in the dense hidden layer is set to ten. In the embedding layer of both layer structures, we choose an *output dimension* of 25 and train the network for ten epochs. Based on these values, we create three additional sets of parameter configurations. In the second configuration set, we extend the training phase by increasing the number of epochs to 30, in order to adapt the model better to the training data. The third configuration reduces the performed training to five epochs but instead extends the *embedding layer dimensionality* to 40. More sparse vectors in the embedding layer could be better distinguishable, which should allow reducing the training phase while maintaining a stable classification prediction. In the last configuration, we modify the parameters of the convolution and dense hidden layer. For the former, the *number of filters* is reduced to eight, focussing more on single features and increasing the importance of each. The *number of neurons* for the latter is increased to 15, which should result in more fine-grained class decisions due to the wider variety available. All four configurations use batch training with 300 elements per batch and a third of the available training data as a validation set. The differences between the parameter sets are displayed in Table 6.6.

In Figure 6.2 can be seen, that the precision on the category dataset and extended category dataset remains quite stable at about 96 per cent for all tested configurations. The only exception is the first parameter configuration in combination with the convolution neural network classifier. In this case, the precision is reduced to 77 per cent. All results are supported by similar high recall value, the only outlier being the first configuration with the convolution neural network, which achieves a value of 0.84. From these results can be seen that the category and extended category dataset perform equally well on both layer structures and do not strongly depend on the chosen parameters.

The application-based datasets seem more problematic, as the convolution network with the first and second parameter configuration accomplishes precision values of about 0.73, supported by recall values of 0.8. The extended training phase of the second parameter configuration does not seem to have an impact on the application dataset. Unlike the application dataset, the performance on the extended application dataset benefits

from additional training. Comparing the ten training epochs to 30 specified in the second configuration, precision and recall values increase by seven per cent, from 0.77 to 0.84 for the former and from 0.82 to 0.89 for the latter. As the extended application dataset contains more diverse features than the application dataset, a higher amount of training could be needed to detect classes reliably. Extending the embedding layer dimensionality while lowering the amount of training in the third parameter set results in a decreased classification performance on the application dataset. The precision drops to 38 per cent with a recall of 0.5. In this case, the extended application dataset outperforms the base dataset clearly. Achieving 80 per cent precision and a recall of 0.86, the performance of this parameter set is within comparable ranges to the first and second configuration. It seems that packets from the application dataset can be mapped onto smaller embedding vectors than the larger number of packets in the extended dataset. In this case, the additional reduction of the amount of training results in a decreased performance. Reducing the number of filters in the last configuration, the precision of the application dataset drops to 0.35 with a recall of 0.49, being the worst performance of all neural network configurations and datasets. The extended application dataset is affected in a similar way, as the classification precision is 0.42, supported by a recall value of 0.55. Increasing the importance of single features leads to focus on characteristics that are only present in the training set, resulting in a less generalised model.

While using the dense classification network, the application dataset achieves a precision value of 0.61 and a recall value of 0.52. On the extended application dataset, the precision is comparable to the category datasets with 0.92, supported by a recall of 0.9. Again, the performance difference between the application and extended application dataset is likely caused by assigning too much weight to features which are not present in the test set. Since a comparable parameter configuration was used in the convolution neural net, this decrease in performance could result from the missing self-optimization of the convolution filters. With this layer architecture, the application dataset seems to draw advantage from an extended training phase, as precision and recall increases to 0.93, respectively 0.92. Interestingly, the performance on the extended application dataset drops to a precision of 0.5 and recall of 0.62. A possible explanation would be overfitting on the training data, which is supported by the results using the third configuration. Since the amount of training is reduced in this configuration, the precision increases to 0.9 and recall to 0.92. As to be expected, the application dataset does not perform well using the third configuration. The precision value drops to 0.38 with 0.54 recall. Both the application and extended application dataset benefit from additional nodes in the dense hidden layer. For the former, the performance improves to a precision value of

Configuration	1	2	3	4
Number of Neurons	10	10	10	**15**
Doc2Vec Training	20	**50**	**30**	20
Dense Classifier Training	10	**30**	**5**	10

Table 6.7: The parameter configurations for Doc2Vec combined with the dense neural network classifier.

0.81, whereas the latter achieves 0.86. The recall value for both datasets exceeds 0.86. In contrast to the previous configurations of the dense classifier, increasing the number of neurons results in nearly equal performance on the application and extended application dataset. Precision and recall are only marginally lower as the values for the category datasets. For this reason, this dense network configuration seems to be suitable to apply on all four datasets. Further extending the number of dense layer nodes did not increase the classification performance.

The recall values match the corresponding precision values of all network classifier experiments quite well, resulting in a similar distribution for the F_1 score, shown in Figure 6.2. Generally speaking, the dense network classifier provides the best results using 15 neurons in the dense hidden layer. In this configuration, it is suited for all datasets. Although the convolution classifier is slightly less precise on application datasets, it is less dependent on parameters while still resulting in more than 70 per cent precision and recall on all datasets. Nearly all problematic configurations of this classifier are most likely caused by overfitting on features in the training set. By reevaluating the importance of all feature values, it could be possible to increase the performance of these configurations.

During additional experiments, we could observe the impact of the performed substitution of static IP addresses with their subnet ranges. While using static IP addresses, the loss diverges during the performed training. The precision drops by about ten per cent on all datasets, disregarding the specified parameters.

6.2.3 Approach 3 - Doc2Vec with External Classifier

Since this approach combines Doc2Vec with a dense network classification, we reuse the parameter configurations of the previous approaches for comparability. In the first configuration, we use a Doc2Vec *window size* of 8, a *min count* of 0 and a *vector size* of

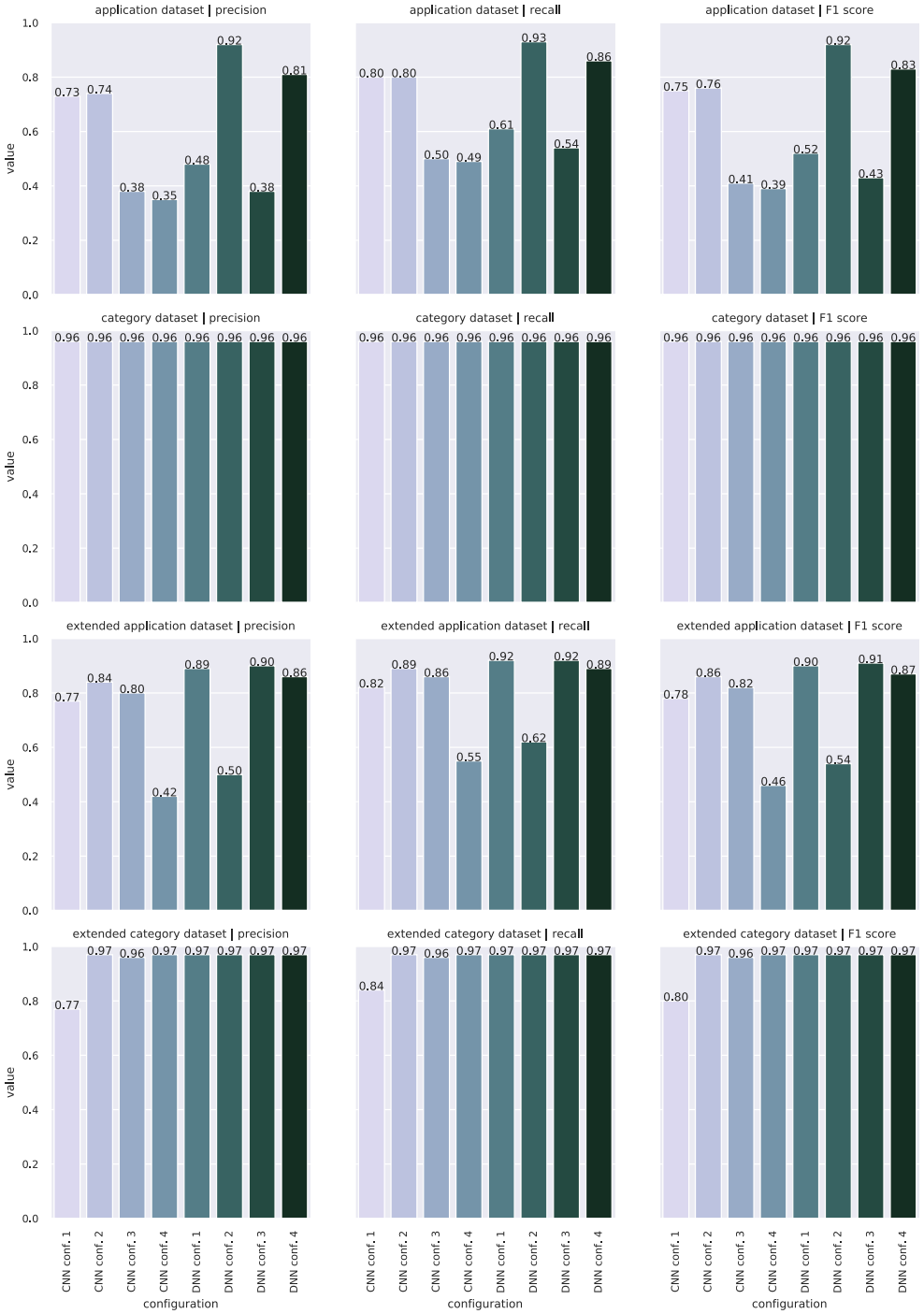


Figure 6.2: Precision, recall and F_1 score of the different neural network configurations.

300. In the dense layer structure, we set the *number of neurons* in the hidden layer to 10. We train Doc2Vec for 20 iterations and the dense classifier for ten epochs. Based on these values, we compare the results of three additional parameter configurations, altering values of Doc2Vec and the dense network classifier at the same time. The second configuration extends the amount of training by setting the Doc2Vec training iterations to 50 and the epochs of the dense network to 30. In the third configuration, the training of the dense network is reduced to five epochs, whereas Doc2Vec is trained for 30 iterations. Changing the *number of neurons* in the dense hidden layer to 15 represents the last parameter configuration. Similar to the second approach, a batch size of 300 and a randomly chosen validation set of 30 per cent is used during the training of the dense classifier. The parameter configurations can be seen in Table 6.7.

The change of the classifier boosts the prediction performance slightly compared to the first Doc2Vec approach with a logistic regression classifier. The increase in precision and recall are shown in Figure 6.3. Using the first parameter set with Doc2Vec in PV-DM configuration, the precision improves on the application dataset by nearly 20 and on the category dataset by 17 per cent, referring to the first approach. Similarly, the recall values increase to 0.52 and 0.58. In contrast, the extended datasets do not benefit in the same amount. For the extended application dataset, the precision rises by ten per cent to 0.37 and the recall by seven per cent to 0.44. The extended category dataset achieves a nearly equal precision with a minor improvement by two per cent, supported by a better recall of 0.51. The prediction performance of all datasets draws advantage from an extended training phase for Doc2Vec and the dense network. On the application dataset, the precision increases to 0.5 with a recall of 0.57, whereas the category dataset achieves 0.6, supported by a recall of 0.61. The precision on the extended application dataset enhances to 0.41, while an increase to 0.53 is observable on the extended category dataset. On both datasets, the recall value improves by five per cent to 0.49 for the extended application and 0.56 for the extended category dataset. In further experiments, we could improve the classification performance on all datasets. By increasing the Doc2Vec iterations to 75 and the dense classifier training epochs to 50, we boosted the performance by about ten per cent. Conversely, the precision and recall did not enhance by extending the amount of training any further. The number of packets per class does not seem to have a large impact on the performance, as the F_1 score difference between application and category dataset is only seven per cent.

As expected, reducing the amount of training does not yield better results. Only the category and extended category dataset achieve precision and recall values comparable

to the first configuration. The precision on the application dataset, on the other hand, drops to 0.39 with a recall of 0.47, whereas the extended application dataset results in a precision value of 0.29. This value represents the lowest performance of this approach. Increasing the neurons instead improves the prediction performance on all datasets in comparison to default parameters. The application dataset yields a precision of 0.48 with a recall or 0.54, though the category dataset outperforms the former with a precision of 56 per cent, supported by a slightly better recall of 0.58. On the extended application dataset, the increase of neurons seems to have the same effect as extending the amount of training in the second parameter set, as it results in the same precision and recall values. The precision on the extended category dataset instead is enhanced to a value of 0.51 with 0.54 recall. This increase in performance is similar to increasing the number of neurons for the dense classifier of the second approach.

Testing these configurations using PV-DBoW, precision and recall improvements of more than ten per cent compared to the first Doc2Vec approach can be seen. On the application dataset, a classification precision of 0.47 with a recall value of 0.53 is achieved, while the category dataset yields marginally better results with a precision value of 0.51 and a recall of 0.55. The precision on the extended application dataset rises to 0.48 and a recall value of 0.46, whereas the extended category dataset achieves 52 per cent precision, supported by a recall of 0.55. Increasing the amount of training in the second parameter set slightly improves the classification results on all datasets except for the extended categories. A marginal decline can be observed on this dataset. On the application and extended application dataset, the precision enhances by four respectively one per cent. The most substantial improvement can be seen on the category dataset, where the precision increases to 0.62 per cent with a recall of 0.64. The contrary effect is observable by reducing the amount of training. By using the third parameter configuration, the precision on the application dataset decreases to 0.36 and the recall to 0.44, whereas on the category dataset, the lowest precision value of 0.5 is achieved. The extended application dataset precision drops to 0.32, although supported by a recall of 0.42. Similarly, the precision and recall of the extended category dataset decrease to 0.45, respectively 0.49. With the last parameter set, the classification performance increases again. The application dataset achieves a precision value of 0.53 and a recall of 0.58, the highest values on this dataset using this approach. The extended application dataset benefits in the same way, as the precision is increased to 0.42, the maximum value on this dataset. Furthermore, the precision on the category dataset is on par with the extended training configuration, although the recall is slightly worse. On the extended category dataset, the classifier achieves a precision of 0.51 with 0.55 recall. As

before using PV-DM, this increase is similar to the performance boost observed in the dense neural network with configuration four in the second approach.

From the resulting F_1 scores shown in Figure 6.3 can be seen, that the dense network classifier yields more stable results on all parameter sets and reduces the fluctuation compared to the logistic regression classifier. No significant difference between Doc2Vec using PV-DM or PV-DBoW can be observed. Generally, the classification performance is drastically below the convolution and dense neural network of the second approach. The most consistent classification performance is achieved on category-based datasets. Using PV-DBoW on the category dataset, the highest performance values are obtained, although PV-DM results in a lower fluctuation between each configuration.

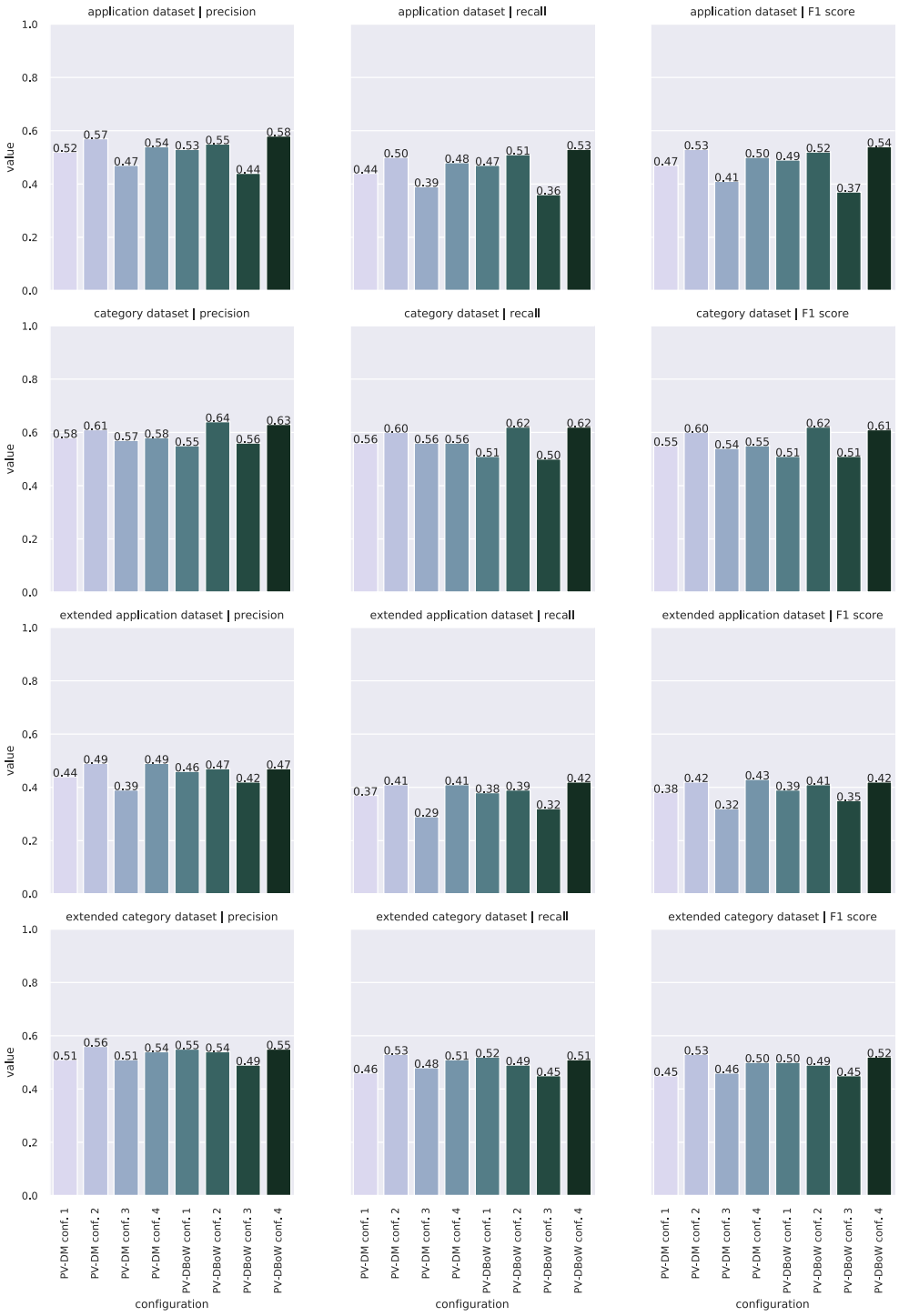


Figure 6.3: Precision, recall and F_1 score of Doc2Vec with logistic regression classifier.

6.3 Limitations

From the results can be seen, that approaches based on Doc2Vec do not achieve high precision and recall values. As a possible reason, Doc2Vec could be inappropriate for this structure of traffic data, since the change in used parameters did not alter the results drastically. A solution is to apply other machine learning approaches to this data. This explanation is supported by the better performance of the neural network classifier on the application and especially category-based datasets.

Most encountered problematic configurations of the neural networks likely result from overfitting on the training data. Neither a dropout layer nor the use of a global max pooling layer has proven to be an effective countermeasure, leaving only the option to increase the dataset quality. As overfitting depends on the chosen network features from the dataset, the importance of each feature has to be reevaluated. New features could be added or existing ones removed to better the results. Changing the used features would require to adapt the network parameters as well, potentially leading to completely different models and approaches.

6.4 Summary

The classification neural networks of the second approach are clearly superior to Doc2Vec-based approaches on the evaluated datasets. Of both layer structures, the dense neural network slightly outperforms the convolution network on the application dataset and the extended application dataset, while both networks are on par regarding their F_1 scores on the category and extended category dataset. The achieved performance seems to depend mainly on the dataset quality rather than the chosen configuration. For example, on the category dataset, the same high precision and recall values are achieved regardless of the configuration or what type of neural network was used.

The two Doc2Vec-based approaches do not achieve comparable levels of classification performance with respect to the neural networks. An F_1 score of 60 per cent is exceeded in only three configurations, while in all other cases the performance remains significantly below this threshold. Furthermore, these scores show that the substitution of the logistic regression classifier with a dense network increases the performance by at least ten per cent. The only exception is the third configuration using PV-DBoW on the application dataset, which improves by two per cent. As the performance differ-

ence between application-based and category-based datasets is marginal, the number of packets per class does not seem to be an important factor. Even balancing the unequal amount of packets per category did not increase the performance. We cannot observe a significant difference in prediction performance between PV-DM and PV-DBoW, regardless of the classifier used. This could be an indicator that the surrounding network packet context does not provide any additional information or that using Doc2Vec is inappropriate for this data.

Both Doc2Vec-based approaches achieve their best F_1 scores on the category dataset, while the results on the application and extended application dataset fluctuate more strongly. A similar pattern can be seen in the scores of the dense and convolution neural network as well. This could indicate that either the amount of sessions per application is not sufficient or, more likely, that the network packets are not distinctive enough to be used in these models. Furthermore, it can be seen that the parameter sets have only limited influence on the classification results of well-performing models. On the majority of the dataset, the F_1 score remains in a small range within each approach regardless of the parameter configuration.

The used network features seem to be essential in this combination, as removing single ones during our experiments resulted in a significant amount of misclassifications. For example, when removing IP addresses or ports, the F_1 score for the second approach decreases to a maximum of 0.58, while Doc2Vec with dense network classifier achieves a maximum of 0.39. The minimum score for the former lies at 0.24, where the latter yields 0.25. Although the first approach is not as strongly affected as others, the F_1 score still drops below 0.33, with 0.16 per cent as a lower bound.

The data points of the best-performing combinations of model and dataset were visualised for further analysis using UMAP in Tensorboard. The resulting points are coloured based on the related label and shown in Figure 6.4. As the data points were recorded directly in front of the dense classification layer, the Doc2Vec approach with logistic regression classifier could not be visualised in this way. Since UMAP projects high-dimensional data onto three dimensions, visualised points close to each other are not necessarily neighbours in the original space. For this reason, graphical proximity serves only as a first indicator. Instead, Tensorboard provides the possibility to list surrounding points based on the distance to the current selection. This measurement serves as the foundation of our analysis.

Figure 6.4a shows the visualisation of the convolution neural network on the category

dataset using the fourth parameter configuration. It can be seen that multiple small clusters are formed, of which the majority of points belongs to the same category. In the centre of the space, the data points create a sparsely distributed point cloud, which in itself contains regions of point concentrations of single categories.

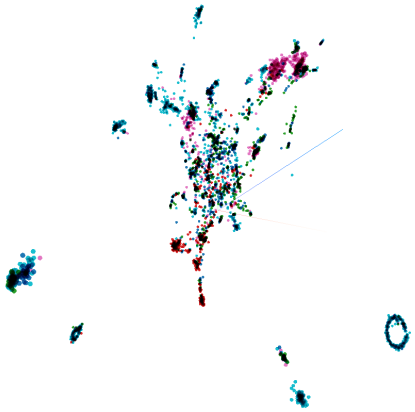
The dense neural network on the extended category dataset in the second parameter configuration is shown in Figure 6.4b. A similar point cloud of data points is observable in the centre of the visualisation, though groups of categories are formed in arm-like structures of the cloud. The upper right arm mainly consists of data points of the category shop, whereas the majority of the messenger category can be found at the bottom of the cloud. In the top left arm, most of the data points belong to the category socialmedia, while the category game dominates the centre. Surrounding the central point cloud, multiple smaller clusters and a ring-like structure are formed by data points of the streaming category.

In Figure 6.4c, the data points resulting from Doc2Vec using PV-DM with the fourth parameter configuration on the category dataset are displayed. Clusters of higher point density can be seen, although the categories in these clusters cannot be distinguished as clearly as in the previous plots. Each cluster consists of data points from multiple categories, where the concentration of single category data points increases near the cluster borders. In the cluster centres, points of other categories are nearby, indicating probable misclassifications of the model. A high density of data points belonging to the category streaming is observable in the ring-like structure. The sparse point cloud instead consists of all categories, although smaller regions inside are dominated by single categories.

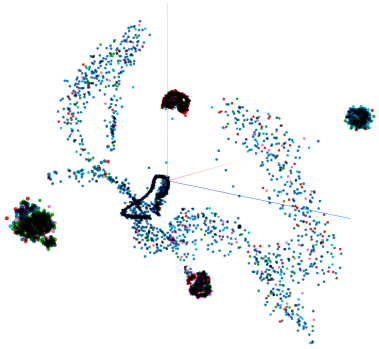
Figure 6.4d shows Doc2Vec in the second parameter configuration using PV-DBoW on the category dataset. In this plot, well-distinguishable concentrations of data points can be seen, as points of the categories game, messenger and shop form their own clusters. The group in the upper left corner of the plot is a mixture of the category shop and socialmedia, where points of both categories are in close proximity. Although the entwined ring structure is built by a majority of data points belonging to the streaming category, a significant number of points from the messenger and game category can be seen inside as well.

Based on the collected results, it seems that Doc2Vec is not suited for network traffic classification on these datasets and that higher classification performance can be achieved using a tailored dense or convolution neural network.

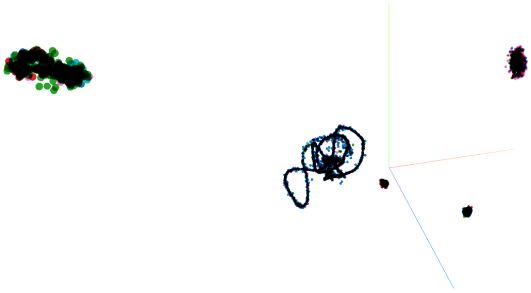
(a) Convolution neural network in fourth configuration on category dataset



(b) Dense neural network in second configuration on extended category dataset



(c) Doc2Vec with dense classifier in fourth configuration on category dataset



(d) Doc2Vec with dense classifier in second configuration on category dataset

Figure 6.4: Tensorboard UMAP visualisation of model data points.

Chapter 7

Conclusion

Identifying applications based on their network traffic can be useful in a wide range of scenarios. For example, fingerprinting a device could be possible if a sufficiently large amount of installed applications on the device can be identified. Identification techniques can be applied in large networks to detect, whether applications containing malware are present on user devices. By analysing the applications used in a network, the service quality of network connections can be optimised to favour the most frequently used apps. With the increasing use of transport encryption in smartphone applications, traffic analysis becomes more difficult as the inspection of network packet content is prevented. But even with traffic encryption in place, traffic metadata remains unencrypted. Although metadata features like hostnames can be useful as the first indicator to determine applications or services, the result tends to be more accurate if multiple features are combined. However, identifying meaningful features is a complex task, as the importance of each feature strongly depends on the type of data and application. Feature combinations often form patterns hidden in metadata. Those patterns are often characteristic for a service or app, serving as another indicator. Patterns can span over vast amounts of metadata. For this reason, the detection of patterns often requires to process large quantities of network data.

In this thesis, we proposed a way to identify smartphone applications by analysing the related network traffic metadata. The solution leaves any transport encryption used in the network connection intact. Our method focuses on the analysis of metadata gathered from network packets, as packet information is available in plain text. Since relying on small quantities of metadata often leads to a high number of erroneous identifications. For this reason, we applied supervised machine learning models to a large amount

of network metadata, as these models benefit from large quantities of training data. Such models provide the ability to detect subtle patterns in network traffic metadata. We showed that metadata share similarities with the structure of natural language text documents. Network protocols define structural rules equivalent to grammar. Similar to words, the possible placement of a packet feature is dictated by the protocol, while the value itself could vary within a given range. The features itself are part of network packets, which we, therefore, assume equivalent to sentences. Similar to sentences forming text documents with a given context, packets can be grouped into a TCP session. For each session, we consider the package name of the source application as the context of a session. The session context is represented by the class label of each network packet in the dataset. The described similarity enabled us to rely on well-established models from the field of natural language processing, which we presented in three methods. The first relied on Doc2Vec, a supervised machine learning model tailored to infer the context from text documents. Doc2Vec transformed the words from the documents into a vector space, taking the context information of sentences and documents into account. The resulting vector space served as input for a logistic regression classifier, which predicted the document context in the form of the application class labels. For the second approach, we used word embeddings in front of a convolution and a dense neural network to process text documents and detect the corresponding labels. In the third approach, we combined Doc2Vec for the word transformation with a dense neural network classifier for the prediction.

We implemented a prototype of a network traffic-based application detector, which is able to learn patterns contained in labelled metadata features in order to predict the application class. Our implementation includes all proposed approaches, allowing us to customise different model parameters. We explained the process of collecting connection metadata along with application name labels on Android smartphones and provided a preprocessing module for creating structured text documents to train the models. The preprocessing module can also be applied to unseen traffic without labels, in order to predict the most similar application class labels using the developed classification module with a trained model.

Based on real user interaction, we collected network traffic data from multiple apps on an Android smartphone. We transformed the network sessions in the gathered traffic into text documents. From this text documents, we created four evaluation datasets. The documents of the first are labelled with the corresponding application name. The second dataset consists of the same documents grouped into categories related to the

purpose of the app. Both datasets were extended artificially by varying a certain amount of features in order to introduce noise, resulting in the other two datasets. We compared the performance of our approaches on these datasets and evaluated the influence of various model parameters. Based on our experiments, we draw the following conclusions:

- The convolution and dense neural network outperform the Doc2Vec with logistic regression and dense neural network classifier in every scenario. On the category and extended category dataset, the F_1 score remained at about 97 per cent, regardless of the chosen parameters. The classification performance seems to depend mainly on the selected features.
- Doc2Vec with logistic regression classifier achieved F_1 scores between 21 and 40 per cent. Between the category and application dataset, a difference of about 10 per cent could be observed. As both extended datasets performed worse than the base datasets, it seems that the logistic regression classifier is inappropriate for this type of data. Although the change of classifier to a dense neural network increased precision and recall values slightly, the difference to the convolution and dense neural network is significant. It seems that Doc2Vec is not suitable for processing network traffic in this form.
- The network parameters have an only marginal influence on the achieved precision and recall, though exceptions can be observed using dedicated neural networks and Doc2Vec with dense classifier on the application dataset.
- Both Doc2Vec approaches did not display significant differences between PV-DM and PV-DBoW. This result indicates that either the context of the surrounding word environment did not provide any valuable further information or that Doc2Vec is inappropriate for the selected network packet features.

Bibliography

[1] Aiolli, F. et al. "Mind your wallet's privacy: identifying Bitcoin wallet apps and user's actions through network traffic analysis". In: *Symposium on Applied Computing – SAC 2019*. ACM, 2019, pp. 1484–1491. ISBN: 978-1-4503-5933-7.

[2] Alan, H. F. and Kaur, J. "Can Android Applications Be Identified Using Only TCP/IP Headers of Their Launch Time Traffic?" In: *Security and Privacy in Wireless and Mobile Networks – WISEC 2016*. ACM, 2016, pp. 61–66. ISBN: 978-1-4503-4270-4.

[3] Arora, A., Garg, S., and Peddoju, S. K. "Malware Detection Using Network Traffic Analysis in Android Based Mobile Devices". In: *Eighth International Conference on Next Generation Mobile Apps, Services and Technologies, NGMAST 2014, Oxford, United Kingdom, September 10-12, 2014*. IEEE, 2014, pp. 66–71. ISBN: 978-1-4799-5073-7.

[4] Banerjee, I. et al. "Comparative effectiveness of convolutional neural network (CNN) and recurrent neural network (RNN) architectures for radiology text report classification". In: *Artificial Intelligence in Medicine* 97 (2019), pp. 79–88.

[5] Chaddad, L. et al. "App traffic mutation: Toward defending against mobile statistical traffic analysis". In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops, INFOCOM Workshops 2018, Honolulu, HI, USA, April 15-19, 2018*. IEEE, 2018, pp. 27–32. ISBN: 978-1-5386-5979-3.

[6] Conti, M. et al. "The Dark Side(-Channel) of Mobile Devices: A Survey on Network Traffic Analysis". In: *IEEE Communications Surveys and Tutorials* 20 (2018), pp. 2658–2713.

[7] Dereli, N. and Saraclar, M. "Convolutional Neural Networks for Financial Text Regression". In: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28 - August 2, 2019, Volume 2: Student Research Workshop*. Association for Computational Linguistics, 2019, pp. 331–337. ISBN: 978-1-950737-47-5.

[8] Dong, S. and Li, R. “Traffic identification method based on multiple probabilistic neural network model”. In: *Neural Computing and Applications* 31 (2019), pp. 473–487.

[9] Hajjar, A., Khalife, J., and Diaz-Verdejo, J. E. “Network traffic application identification based on message size analysis”. In: *J. Network and Computer Applications* 58 (2015), pp. 130–143.

[10] He, G. et al. “Mobile app identification for encrypted network flows by traffic correlation”. In: *IJDSN* 14 (2018).

[11] Ichino, M., Maeda, H., and Yoshiura, H. “Score Level Fusion for Network Traffic Application Identification”. In: *IEICE Transactions* 99-B (2016), pp. 1341–1352.

[12] Jain, A. V. “Network Traffic Identification with Convolutional Neural Networks”. In: *Conference on Dependable, Autonomic and Secure Computing – DASC 2018*. IEEE Computer Society, 2018, pp. 1001–1007. ISBN: 978-1-5386-7518-2.

[13] Kim, J. and Sim, A. “A New Approach to Multivariate Network Traffic Analysis”. In: *J. Comput. Sci. Technol.* 34 (2019), pp. 388–402.

[14] Kim, J. et al. “Multivariate network traffic analysis using clustered patterns”. In: *Computing* 101 (2019), pp. 339–361.

[15] Le, Q. V. and Mikolov, T. “Distributed Representations of Sentences and Documents”. In: *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*. Vol. 32. JMLR Workshop and Conference Proceedings. JMLR.org, 2014, pp. 1188–1196.

[16] Li, D., Zhu, Y., and Lin, W. “Traffic Identification of Mobile Apps Based on Variational Autoencoder Network”. In: *13th International Conference on Computational Intelligence and Security, CIS 2017, Hong Kong, China, December 15-18, 2017*. IEEE Computer Society, 2017, pp. 287–291. ISBN: 978-1-5386-4822-3.

[17] Liu, Z., Zhou, W., and Li, H. “Scene text detection with fully convolutional neural networks”. In: *Multimedia Tools Appl.* 78 (2019), pp. 18205–18227.

[18] Lotfollahi, M. et al. “Deep Packet: A Novel Approach For Encrypted Traffic Classification Using Deep Learning”. In: *CoRR* abs/1709.02656 (2017).

[19] Maaten, L. v. d. and Hinton, G. “Visualizing data using t-SNE”. In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[20] Martin, M. L. et al. “Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things”. In: *IEEE Access* 5 (2017), pp. 18042–18050.

[21] McInnes, L. and Healy, J. “UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction”. In: *CoRR* abs/1802.03426 (2018).

[22] Meidan, Y. et al. “ProfilIoT: a machine learning approach for IoT device identification based on network traffic analysis”. In: *Symposium on Applied Computing – SAC 2017*. ACM, 2017, pp. 506–509. ISBN: 978-1-4503-4486-9.

[23] Mikolov, T. et al. “Efficient Estimation of Word Representations in Vector Space”. In: *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*. 2013.

[24] Mizumura, N. et al. “Smartphone Application Usage Prediction Using Cellular Network Traffic”. In: *2018 IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2018, Athens, Greece, March 19-23, 2018*. IEEE Computer Society, 2018, pp. 753–758. ISBN: 978-1-5386-3227-7.

[25] Nazari, Z., Noferesti, M., and Jalili, R. “DSCA: an inline and adaptive application identification approach in encrypted network traffic”. In: *Proceedings of the 3rd International Conference on Cryptography, Security and Privacy, ICCSP 2019, Kuala Lumpur, Malaysia, January 19-21, 2019*. ACM, 2019, pp. 39–43. ISBN: 978-1-4503-6618-2.

[26] Nguyen, H. T. et al. “Text-independent writer identification using convolutional neural network”. In: *Pattern Recognition Letters* 121 (2019), pp. 104–112.

[27] Nguyen, T. D. et al. “DÏoT: A Crowdsourced Self-learning Approach for Detecting Compromised IoT Devices”. In: *CoRR* abs/1804.07474 (2018).

[28] Parwez, M. A., Abulaish, M., and Jahiruddin. “Multi-Label Classification of Microblogging Texts Using Convolution Neural Network”. In: *IEEE Access* 7 (2019), pp. 68678–68691.

[29] Pluskal, J., Lichtner, O., and Rysavý, O. “Traffic Classification and Application Identification in Network Forensics”. In: *Advances in Digital Forensics XIV - 14th IFIP WG 11.9 International Conference, New Delhi, India, January 3-5, 2018, Revised Selected Papers*. Vol. 532. IFIP Advances in Information and Communication Technology. Springer, 2018, pp. 161–181. ISBN: 978-3-319-99276-1.

[30] Saltaformaggio, B. et al. “Eavesdropping on Fine-Grained User Activities Within Smartphone Apps Over Encrypted Network Traffic”. In: *Workshop on Offensive Technologies – WOOT 2016*. USENIX Association, 2016.

[31] Sawabe, A., Iwai, T., and Satoda, K. “Identification of Smartphone Applications by Encrypted Traffic Analysis”. In: *Consumer Communications & Networking Conference – CCNC 2019*. IEEE, 2019, pp. 1–2. ISBN: 978-1-5386-5553-5.

[32] Shahid, M. R. et al. “IoT Devices Recognition Through Network Traffic Analysis”. In: *IEEE International Conference on Big Data, Big Data 2018, Seattle, WA, USA, December 10-13, 2018*. IEEE, 2018, pp. 5187–5192. ISBN: 978-1-5386-5035-6.

[33] Sivan, N., Bitton, R., and Shabtai, A. “Analysis of Location Data Leakage in the Internet Traffic of Android-based Mobile Devices”. In: *CoRR* abs/1812.04829 (2018).

[34] Swarnkar, M. et al. “AppHunter: Mobile Application Traffic Classification”. In: *IEEE International Conference on Advanced Networks and Telecommunications Systems, ANTS 2018, Indore, India, December 16-19, 2018*. IEEE, 2018, pp. 1–6. ISBN: 978-1-5386-8134-3.

[35] Taylor, V. F. et al. “AppScanner: Automatic Fingerprinting of Smartphone Apps from Encrypted Network Traffic”. In: *IEEE European Symposium on Security and Privacy – EURO S&P 2016*. IEEE, 2016, pp. 439–454. ISBN: 978-1-5090-1751-5.

[36] Taylor, V. F. et al. “Robust Smartphone App Identification via Encrypted Network Traffic Analysis”. In: *IEEE Trans. Information Forensics and Security* 13 (2018), pp. 63–78.

[37] Wang, Q. et al. “I know what you did on your smartphone: Inferring app usage over encrypted data traffic”. In: *Communications and Network Security – CNS 2015*. IEEE, 2015, pp. 433–441. ISBN: 978-1-4673-7876-5.

[38] Wright, C. V., Monrose, F., and Masson, G. M. “On Inferring Application Protocol Behaviors in Encrypted Network Traffic”. In: *Journal of Machine Learning Research* 7 (2006), pp. 2745–2769.

[39] Yamansavascilar, B. et al. “Application identification via network traffic classification”. In: *2017 International Conference on Computing, Networking and Communications, ICNC 2017, Silicon Valley, CA, USA, January 26-29, 2017*. IEEE Computer Society, 2017, pp. 843–848. ISBN: 978-1-5090-4588-4.

[40] Yao, L., Mao, C., and Luo, Y. “Graph Convolutional Networks for Text Classification”. In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019, pp. 7370–7377. ISBN: 978-1-57735-809-1.

[41] Zaman, M. et al. “Malware detection in Android by network traffic analysis”. In: *International Conference on Networking Systems and Security, NSysS 2015, Dhaka, Bangladesh, January 5-7, 2015*. IEEE, 2015, pp. 1–5. ISBN: 978-1-4799-8126-7.

[42] Zheng, T. et al. “Detection of medical text semantic similarity based on convolutional neural network”. In: *BMC Med. Inf. & Decision Making* 19 (2019), 156:1–156:11.