Information Security

Make PDF safe again

Detecting malicious PDF files with Deep Learning

Diploma Thesis

For attainment of the academic degree of

Diplom-Ingenieur/in

submitted by

Kevin DORNER, BSc.

is201844

in the

University Course Information Security at St. Pölten University of Applied Sciences

The interior of this work has been composed in LaTeX.

Supervision

Advisor: Dipl.-Ing. Patrick Kochberger, BSc

Assistance: -

St. Pölten, June 29, 2022 _____ _____

(Signature author) (Signature advisor)

Ehrenwörtliche Erklärung

Ich versichere, dass

- ich diese Diplomarbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich sonst keiner unerlaubten Hilfe bedient habe.
- ich dieses Diplomarbeitsthema bisher weder im Inland noch im Ausland einem Begutachter/einer Begutachterin zur Beurteilung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe.
- diese Arbeit mit der vom Begutachter/von der Begutachterin beurteilten Arbeit übereinstimmt.

Der Studierende/Absolvent räumt der FH St. Pölten das Recht ein, die Diplomarbeit für Lehre- und Forschungstätigkeiten zu verwenden und damit zu werben (z.B. bei der Projektevernissage, in Publikationen, auf der Homepage), wobei der Absolvent als Urheber zu nennen ist. Jegliche kommerzielle Verwertung/Nutzung bedarf einer weiteren Vereinbarung zwischen dem Studierenden/Absolventen und der FH St. Pölten.

Ort, Datum *Unterschrift*

Acknowledgments

Firstly, I would like to express gratitude to my supervisor Dipl.-Ing. Patrick Kochberger, BSc, of the University of Applied Sciences St. Pölten for his expertise, support and guidance during the work on my thesis. Mr. Kochberger was extremely flexible and took time out of his busy schedule on short notice to give comments, valuable insights and encouragements. He gave me a little push in the right direction whenever I needed it and showed great patience with my slightly chaotic way of working.

Furthermore I would like to express my sincere thanks to my friends, Betty, Jakob, Magdalena, Oliver and Wolfgang, because, let´s be honest, without you I would not have made it through all those years of study.

Also, I want to voice my gratitude towards my family, especially my mother and sister, who have not only inspired me to chase a career in IT, but have always had my back and supported me no matter what. I can´t thank you enough for that.

Lastly I need to thank my girlfriend. I know that the past years have been really difficult and that you showed and abnormous amount of patience with me and that you had to make a lot of sacrifices for me. You always believed in me and motivated me to chase my dreams and to follow my passion. Whenever I needed a little push or felt like giving up you were there for me and helped me finish this degree. I could not have done this without you and I will forever be grateful.
I love you.

Thank you.

Citation

This work can be cited using the following bibliography entries: Listing 1,Listing 2

```
@mastersthesis { dorner2022,
    title = {Make PDF safe again - Detecting malicious PDF
     ↪  files with Deep Learning},
    author = {Kevin, Dorner},
    school = {University of Applied Sciences St. P\"{o}lten},
    type = {Diploma Thesis},
    year = {2022}
}
```

Listing 1: BibTeX Citation

```
%0 Thesis
%T Make PDF safe again - Detecting malicious PDF files with
 ↪  Deep Learning
%9 Diploma Thesis
%A Kevin DORNER
%D 2022
%I University of Applied Science St. P\"{o}lten
```

Listing 2: EndNote Citation

Kurzfassung

Malware ist ein allgegenwärtiges Problem, und während die meisten Menschen wissen, dass ausführbare Dateien gefährlich sein können, gelten andere Dateiformate als vertrauenswürdig und sicher. Eines dieser Dateiformate ist Portable Document Format (PDF). Jeden Tag werden Milliarden von PDF-Dateien erstellt damit ist PDF der de-facto Standard für den Austausch von Dokumenten weltweit. Cyberkriminelle versuchen, die große Beliebtheit von PDF auszunutzen, um Malware zu verbreiten oder Phishing-Kampagnen zu starten. Darüber hinaus haben jüngste Forschungsergebnisse gezeigt, dass selbst digital signierte PDF-Dokumente verändert werden können, ohne dass die Signatur ungültig wird.

Artificial Intelligence (AI) in Form von Neural Networks (NN) ist ein vielversprechender Ansatz, Sicherheitsanalysten bei der Identifizierung bösartiger PDF-Dateien zu unterstützen. Insbesondere Convolutional Neural Networks (CNN) haben hervorragende Ergebnisse bei Klassifizierungsaufgaben gezeigt, jedoch ist die Forschung bezüglich der Erkennung von PDF-Malware mit diesen Netzwerken sehr begrenzt. Daher werden in dieser Arbeit drei CNN mit unterschiedlichen Datensätzen trainiert, um PDF-Dateien entweder als gutartig, als Malware, oder als verdächtig zu klassifizieren.

Die Ergebnisse zeigen eine Genauigkeit von mehr als 97 Prozent bei dem Netzwerk, das trainiert wurde, um klassische PDF-Malware, sowie verdächtige Dokumente die manipuliert wurden, zu identifizieren. Darüber hinaus wurde in dieser Arbeit ein Datensatz zum Trainieren eines CNN zur Erkennung eines als 'shadow attack' bekannten Angriffs, sowie ein Tool zur Klassifizierung von PDF-Dateien durch ein NN erstellt.

Abstract

Malware is an ever-present problem, and while most people are aware that executable files can be dangerous, other file formats are deemed as trustworthy and safe. One of these file formats is PDF. Billions of PDF files are created each day and this file format is the de-facto standard for document exchange worldwide. Cybercriminals try to exploit the huge popularity of PDF to spread malware, or launch phishing campaigns. Additionally, recent research has shown that even digitally signed PDF documents can be altered without invalidating the signature.

AI in the form of NN is a promising attempt to help security analysts identifying malicious PDF files. Especially CNN have shown great results in classifying tasks, however, research regarding PDF malware detection with these networks is very limited. Therefore, in this thesis three CNN are trained on different data sets to classify PDF files as either benign, malware, or suspicious.

The results show an accuracy of more than 97 percent for the network that is trained to identify classic PDF malware, as well as suspicious documents that have been tempered with. Additionally a data set to train a CNN to detect an attack known as the "shadow attack", as well as a tool to feed PDF files to a network has been created in this work.

Contents

1. Introduction

> „*Anti-Virus "is dead"*."- Brian Dye, Symantec's senior vice president, Wall Street Journal 2014.05.04[1]

Malware is an ever-present topic and while many people today are aware of the dangers of executable files, other file formats are still viewed as harmless. One of those file formats is PDF. With more than 2.5 trillion [1] PDF files created every year, PDF is the de-facto standard for exchanging documents worldwide, and it seems only logical that attackers target something so popular and widespread. In fact, the earliest attacks [2] on the PDF file format date back to 2001, when a virus nicknamed "Peachy" was spread via PDF. Ever since, researchers, and Antivirus Software (AV) vendors alike, have tried to find ways to detect whether a PDF document is safe to open.

Traditionally, AV software makes use of signature-based detection to detect malware that is already known. The obvious downside of this approach is that malicious code has to be encountered, analyzed by a security analyist and then a signature has to be made and distributed. In times where tens of thousands of new, adapted malware software pieces are written daily, this task is becoming extremely difficult and time-consuming and by the time a threat is identified, it might have already spread and infected countless devices. This problem becomes even more evident, considering that cybercriminals have found ways to automatically generate new malware.

Other solutions, like sandbox systems are too slow to scan high amounts of files in an appropriate time span and even worse, attackers have developed methods to evade such dynamic detection systems.

Furthermore, recent research [3] has shown, that even digitally signed documents can be manipulated, without invalidating the signature, which is especially worrying, considering that PDF documents are used for contracts, invoices, classified research and more.

Therefore research has turned to Machine Learning (ML) to break the wheel and escape the vicious circle of only being able to react to known threats, while attackers already work on new ways to exploit a system.

[1] `https://web.archive.org/web/20170825175728/https://www.wsj.com/news/article_email/SB10001424052702303417104579542140235850578-lMyQjAxMTA0MDAwNTEwNDUyWj`

Especially NN have shown great potential in the field of malware classification and could prove to be an essential tool for security analysts to gain ground on cybercriminals.
Thus, this thesis was created to increase the awareness of PDF security and to evaluate one of the ML approaches to detect malicious PDF files, as well as suspiciously modified documents and validate, if this is a viable solution for the problem at hand.

1.1. Thesis Outline

This thesis is organized in several parts and provides an overview of the PDF file format as well as the weaknesses of this format. Furthermore attacks targeted at PDF and the basics of ML and NN are explained and a NN is trained and evaluated against an attack targeted at digitally signed PDF documents. Additionally a data set is created to train the NN to specifically detect this attack.

Chapter 1 introduces the topic and outlines the underestimated dangers of PDF and contains the motivation of the author to write this thesis. Limitations and contributions of the work done are included in this section as well.
Chapter 2 describes the background knowledge necessary to understand the problems with PDF and additionally covers the basics of ML that are needed to comprehend the practical part of this thesis.
Existing research on this topic and the differences to this thesis are mentioned in Chapter 3. The work of Fettaya *et al.*, on which the practical part largely builds on, is also covered in this section.
Chapter 4 explains the training and testing of the CNN and the development of the specifically designed data set, as well as the results of the experiments done.
Finally, Chapter 5 wraps up the topic and gives an outlook on future work.

1.2. Research Question

The main goal of this thesis is to train a CNN for PDF malware detection and to assess if this method is capable of detecting threats that do not include malware, like attacks on the integrity of a document. Furthermore it is evaluated, if the training process can be improved, by modifying the training samples, to identify such attacks reliably and put them into a separate class that is neither benign nor malware.

1.3. Limitations

This thesis has some limitations. Firstly, only one CNN architecture design is tested and evaluated against the data sets used, which is mainly due to the fact, that solely open-source solutions are considered.
Additionally, the data sets used to train the CNN are mostly based on data that dates back a few years and since malware evolves continually it is unclear how the results transfer to malware of today. Although the self developed data set was created with a lot of effort in terms of randomization of the attack, only form fields are implemented in the attack, which could lead to a bias of the classifier against documents with form fields.

1.4. Contribution

The main contribution of this work is a way to reliably classify PDF files into three different categories: benign, malicious and suspicious and the source code to recreate the results or use it for further research is freely available.
Furthermore a specially designed, publicly available data set to train a NN to detect a specific attack that targets the integrity of PDF documents is included.

2. Prerequisits

This chapter covers the technical background and the details of the PDF as well as potential weaknesses and attack vectors of the file format. Furthermore, basic concepts of ML and NN are explained. Additionally, this part of the thesis describes the ways of detecting malware in PDF files.

2.1. The Portable File Format

Developed in 1992 by Adobe co-founder Dr. John Warnock, PDF is one of the most used and widespread file formats across the globe today and an open standard, maintained by the International Organization for Standardization (ISO) [5]. It has become the de-facto standard concerning the distribution of printable documents. According to [6] it is the second most popular file-format found on the web. PDF is used by companies for the transfer of documents, as well as by researchers to write papers and articles, or by students for their homework.
This section aims to help understand the structure of PDF, as well as the strengths and weaknesses of this file format.

2.1.1. PDF File Structure

The ISO 32000 [8] family of ISO standards defines the core specification of PDFs. A PDF file consists of objects and is constructed of four parts: header, body, cross-reference table and trailer. Fig. 2.1 shows the initial structure of a PDF.

Header

The header contains the version number of the PDF specification [8]. Listing 3 shows an example of the PDF header.

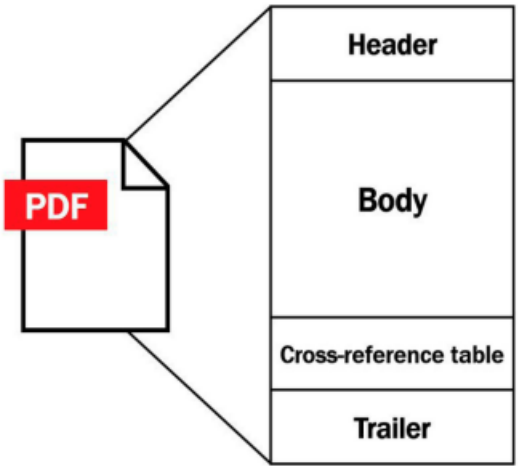

Figure 2.1.: The basic structure [7] of a PDF consists of a header section, the body where all visible content is stored, the cross-reference table, and the trailer section.

Body

The body section holds all the objects [8] and data of the document. This is the data that is shown to the user. Objects can be labelled to enable other objects to refer to them. Labelled objects are called indirect objects and each of them has a unique object identifier.

In total, eight different types of objects are supported by a PDF file:

- **Boolean objects** represent the logical values of true and false.
- **Numeric objects** contain the subtypes of integer and real. *Integer objects* represent mathematical integers, while object of type *real* represent mathematical real numbers.
- **Arrays** are one-dimensional collections of objects. Contrary to many programming languages, PDF arrays can be heterogeneous, so it can contain any combination of numbers, strings, dictionaries or any other objects, including other arrays or it can also zero elements.
- **String objects** consist of a series of zero or more bytes and are stored in a more compact way than integer objects. String objects can either be written as *literal* strings or as *hexadecimal* strings. Additionally, every string object gets encrypted, if the encryption function is used on the file.
- **Dictionary objects** contain parts of objects, their *entries*. Each entry consists of a *key*, which is a name, and a *value*, which can be any kind of object, even other dictionaries.
- **Name objects** are atomic symbols that are uniquely defined by a sequence of any characters (8-bit values) except null.

- **Stream objects** are sequences of bytes of unlimited length. Objects that are likely to be of large amounts of data, like images, are represented as streams.
- **The Null object** has a type and value that are unequal to those of any other object.

```
1            kd@lab:~$ xxd sample.pdf | head -n 1
2          00000000: 2550 4446 2d31 2e37 0d0a 25b5 b5b5 b50d
            ↪  %PDF-1.7..%.....
```

Listing 3: The "xxd" command shows a hexadecimal dump of the file and the "2550 ... b50d" corresponds to the ASCII characters of "PDF-1.7" which is the actual header [8] content. This document uses PDF version 1.7 and the dots following the version number are non-printable ASCII characters. T The % sign indicates a comment in the PDF standard and the example demonstrates that the first two lines are comments, which is true for all PDF documents.

Cross-Reference Table

Information and pointers to all objects in a PDF file can be found in the cross-reference table [8]. This allows for random access to indirect objects in the file, therefore not the whole file has to be searched to locate a specific object. Listing 4 shows an example of a cross-reference table.

Trailer

The location of the cross-reference table [8], the root object (see Section 2.1.3) and of certain special objects within the body of the file are stored in the trailer. PDF reading software start rendering a file from the bottom of the file, which is the trailer. Listing 5 shows an example of a trailer section.

2.1.2. Important Features of PDF

PDF offers a vast amount of features [8]. Besides well-known advantages, like the possibility to open PDF files on almost any device, independent of the Operating System (OS), or a good file compression algorithm and the possibility to encrypt a file or set a password, there are less-known features that are relevant to understand in this work.

```
1  xref
2  0 2437
3  0000000266 65535 f
4  0000000017 00000 n
5  0000000171 00000 n
6  0000000757 00000 n
7  0000001074 00000 n
8  0000001647 00000 n
9  0000001700 00000 n
10 0000012071 00000 n
11 0000012124 00000 n
12 0000025911 00000 n
13 ...
```

Listing 4: Example of a Cross-Reference-Table [8], inspected by opening the file with a standard text editor. The second line only containing two numbers ("0 2437") marks a subsection, where "0" corresponds to the object number and "2437" to the number of objects in this subsection. Each following 20 bytes long line represents one object, where the first 10 bytes are the object´s offset from the start of the file to the beginning of the object. This is followed by a blank space and the generation number of the object, as well as another blank space and either the letters "f" or "n", indicating if an object is free or in use.

```
1  trailer
2  <</Size 351/Root 276 0 R/Info 274 0 R
3  /ID[<2E73E198E8119EC575764BB0F1356027><E67AC8FCCE140348AA880F36
   ↪ DE64CB25>]
4  /Prev 377973>>
5  startxref
6  0
7  %%EOF
```

Listing 5: Example of a PDF Trailer [8], inspected by opening the file with a standard text editor. The "trailer" string specifies the start of the trailer section, followed by a dictionary (in between the "« "and "»" signs) that accepts key-value pairs and stores the contents of the trailer section. /Size specifies the number of entries in the cross-reference table, this case 351. This is followed by /Root, which marks the reference for the document catalog object. /Info describes the reference object of the information dictionary. /ID is an array of two string that build a file identifier. Startxref specifies the offset from the beginning of the file to the cross-reference table and the trailer end with the end-of-file tag "EOF".

Incremental Updates

PDF is designed to allow incremental updates [8]. When a file is updated, instead of rewriting the entire file, changes shall be appended to the end of the file, leaving its original contents intact. This is especially useful when changes to large documents are made, because saving takes less time.
Due to these incremental updates, a PDF file can have more than one body section, as well as multiple cross-reference sections and trailers, as Fig. 2.2 demonstrates.

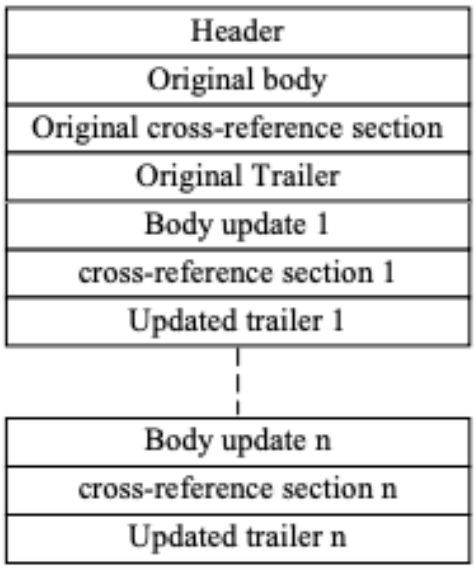

Figure 2.2.: The figure shows, that after the original body, cross-reference and trailer sections, new sections [8] are added to the file structure.

Interactive Features

PDF features that allow a user to interact with a document [8] using a mouse and/or keyboard are called interactive features. These features include:

- Preference settings to control the way a document is presented
- Interactive forms, or "AcroForms" - a collection of fields for gathering information interactively from a user.
- Actions to perform when triggered. This can be something like launching an application, or playing a sound.
- Digital signatures to authenticate the identity of a user and the validity of the contents of a document.
- Annotations for adding text notes, sounds, movies or other ancillary information.

- Other features like measurement properties or navigation facilities.

Multimedia Features

The PDF file format allows the embedding [8] and playing of multimedia content, such as images, sounds, movies, a slideshow, or 3D artwork. This feature comes along with a lot of capabilities. For example, document authors may control play-time requirements, such as which player software should be used to play a given media object. Media objects can be played in different ways, like in a floating windows or in a specific region of a page.

2.1.3. PDF Document Structure

In general, a PDF document is a hierarchy of objects [8] that are contained in the body section of a PDF file. The document´s *catalog* (see Section 2.1.3) dictionary is at the root of the hierarchy. This section explains the most important elements of this structure. Fig. 2.3 illustrates the structure of the object hierarchy.

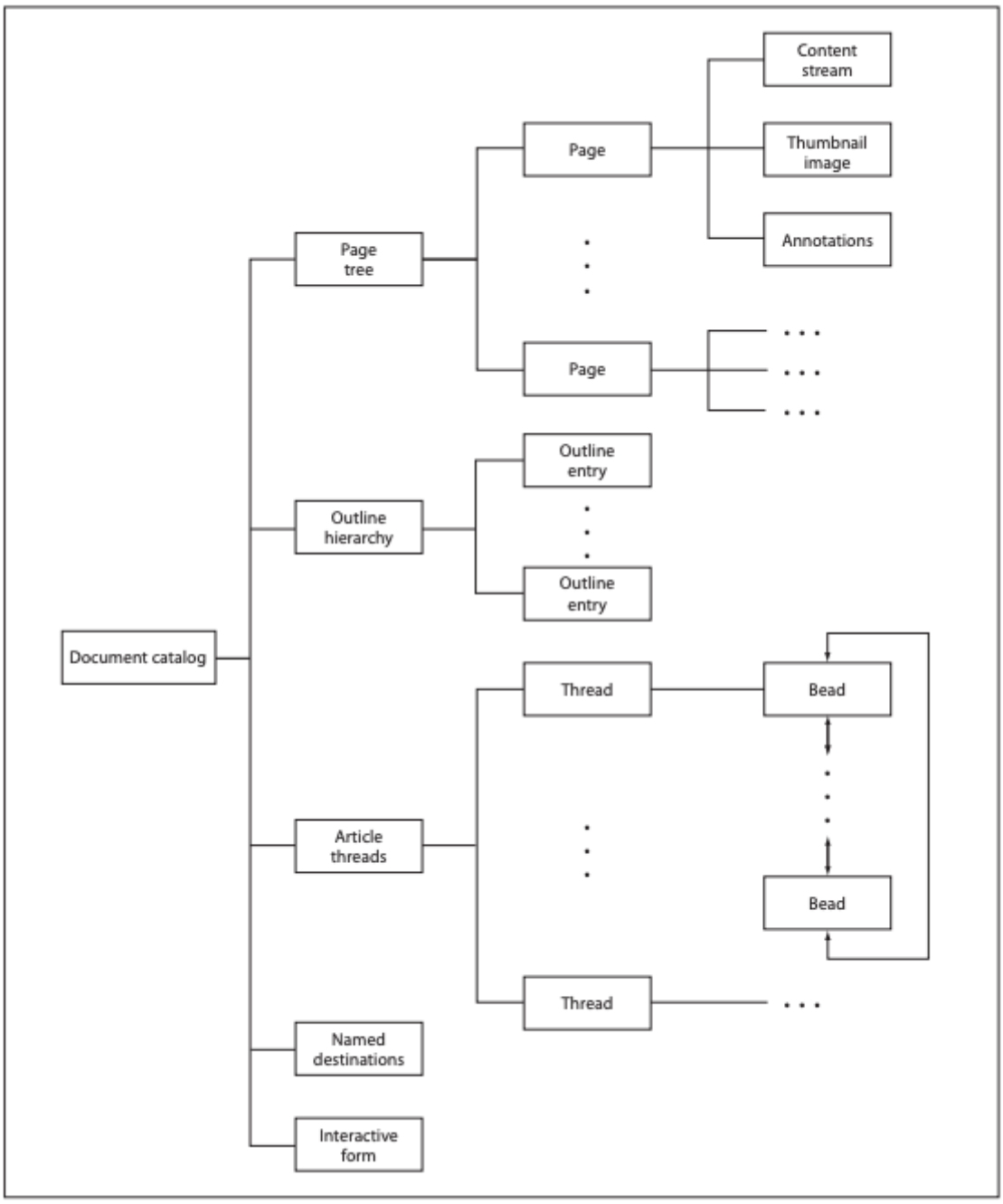

Figure 2.3.: The document catalog [8] splits into five main branches, the page tree, article threads, named destinations, interactive forms and the outline hierarchy. The page stree has two types of nodes, page tree nodes and page objects. Other important branches are the article threads branch and the ourtline hierarchy branch, which contains all the outline items that make up a visible, interactive structure for a user.

Document Catalog

The **Root** entry in the trailer of a PDF file points to the catalog dictionary [8], which is the root of a document´s hierarchy. References to other objects defining the document´s contents, outline, article threads, named destinations and other attributes are stored in the catalog. Furthermore it contains additional information about how the document shall be displayed on the screen, like if some other page than the first one shall be displayed when the document is opened.

Page Tree

The page tree [8] defines the ordering of pages in a document. This structure enables conforming document reading software to quickly open a document containing thousands of pages, using only limited memory. The tree consists of two types of nodes: page tree nodes, which are intermediate nodes, and page objects, which are leaf nodes. Each page object is a dictionary specifying the attributes of a single page of the document.

Article threads

This is an array that contains thread dictionaries. An article [8] is a sequence of physically discontiguous but logically related items, for example a news story that begins on the first page of a newsletter and then runs over onto one or more nonconsecutive interior pages. The article thread defines the sequential flow of an article. This enables a PDF viewer to present the text boxes of an article for easier reading.

Outline hierarchy

Outlines [8], sometimes called bookmarks, allow a user to navigate interactively from one part of the document to another. The outline hierarchy is tree-structured and consists of all outline items and serves as a visual table of contents to display the structure of a document. A user may interactively open and close individual items by clicking them with the mouse and when an item is opened, the immediate children of that item in the hierarchy become visible on the screen. While in most cases an outline item points to a specific destination in a document, other targets [9] like a Uniform Resource Identifier (URI), to direct the reader to a web address associated with the document, can be used.

2.2. PDF Malware

The word "malware" is a combination of "malicious" and "software" and that perfectly fits, as malware is exactly that. Unwanted software that can intentionally cause harm to a system, or fulfill other malicious acts, like stealing credentials, propagate itself, or grant unnoticed access to the infected system. Attacks carried out with the help of PDF files date back to 2001, when a virus named *Peachy* misused features of PDF to execute malicious VBScript. [10]
From 2010 to 2014 malicious PDF files represented the most dangerous infection vector in the wild, only to be then replaced by Office-based malware.
The complexity of the PDF file format enables attackers to use various approaches to conceal code injection or other attack strategies.
Furthermore, the possibility to modify the structure of the PDF file format, by for example injecting benign or malicious material in various portions of the file, make it an attractive choice for malware writers, especially because those characteristics make PDF files particularly interesting in environments where ML is used to detect malware, since it takes less efforts for attackers to create samples that bypass the ML algorithm. [11]
The most common attack vectors are briefly explained in the following paragraphs.

2.2.1. Phishing

Phishing is one of the oldest and still most common attack vectors [12] regarding PDF files. PDF documents are still perceived as safe and harmless by the majority of people, therefore it is viable to place a Uniform Resource Locator (URL), pointing to a malicious website, maybe containing a form where the user should enter some credentials, or where a malware is placed, in a PDF document and distribute that file. Since PDF documents are binary or a mixture of binary and ASCII texts, it is harder to detect the phishing attempt, compared to plain text based phishing attacks.
The main advantages of phishing using PDF documents is, that it does not rely on any existing vulnerabilities in the application used to view the document, as well as the minimal know-how needed to launch such an attack.
Furthermore this method can be used to either target as many people as possible, or for spear phishing attacks, where the goal is to get a high value target to fall victim to the phishing attempt, usually with specially crafted, unique PDF document.

2.2.2. JS Based Attacks

PDF allows the embedding of JS code [8] which is often used for forms in documents or other legitimate use cases like 3D rendering or dynamic content. Furthermore JS code may also reside in other files [10] on a local machine, or even be retrieved from remote locations with features like */URI* or */GoTo*.
However, attackers make use of this feature to trigger specific vulnerabilities, such as buffer overflows [13] in PDF viewer applications, such as Adobe Acrobat Reader or Foxit Reader. Most exploits make use of malicious JS code and since 2008 more than 25 critical Common Vulnerabilities and Exposures (CVE) IDs[1] regarding JS-based vulnerabilities have been recorded in the wild.
According to Corona *et al.*, JS-based malware [14] is typically characterized by three actions:

- **Decoding** - Since malicious code is often obfuscated, a decoding routine has to extract the exploit code and store it within objects. Listing 6 shows an example of a JS-based attack in the wild.
- **Fingerprinting** - Depending on the environment the exploit may adapt its behavior. This is done to decrease the chance of detection and to focus only on environments that are actually vulnerable to the exploit used.
- **Execution** - If the fingerprint prerequisites are met, the exploit is finally executed. It may rely on one or multiple vulnerabilities and different exploitation techniques to successfully jeopardize a PDF viewer and do further damage, like taking control of the whole OS.

2.2.3. Other Exploitable Features

In their paper *PDF Malware Detection based on Stacking Learning* Issakhani *et al.* identify other common PDF features [15] that can be utilized in various ways to deliver an attack. Among those features are:

- **Header Obfuscation** - Malicious PDF files tend to have obfuscated and/or malformed headers [16].
- **Metadata** - Attackers often hide parts of the shell code in the metadata section [17] of a PDF file, which normally contains information about the file, like its creation date.
- **Streams** - Binary data, like image files are stored in stream objects [18]. Attackers use such objects to hide malicious JS.
- **Object Streams** - Objects can be placed in a stream [19]. Objects containing malicious code can be wrapped in such a stream to conceal them.
- **Encoding Filters** - To reduce the size or protect sensitive contents of PDF streams, encoding or compression filters are used [20]. This technique is often used by attackers to hide malicious content.

[1] `https://cve.mitre.org`

```
1   var pr = null ;
2   var fnc = 'ev';
3   var sum = '';
4   app. doc . syncAnnotScan () ;
5       if ( app . plugIns . length != 0) {
6       var num = 1;
7       pr = app. doc . getAnnots ({ nPage : 0}) ;
8       sum = pr[ num ]. subject ;
9   }
10  var buf = "";
11  if ( app . plugIns . length > 3) {
12      fnc += 'a';
13      var arr = sum. split (/ -/) ;
14      for ( var i = 1; i < arr. length ; i++) {
15          buf += String . fromCharCode ("0x"+ arr [i]) ;
16      }
17      fnc += 'l';
18  }
19  if ( app . plugIns . length >= 2) { app [fnc ]( buf ); }
```

Listing 6: Example of a malicious JS decoding routine [14]. First, the encoded exploit is loaded via the *app.doc.getAnnots()* method. Afterwards, the exploit is decoded through the *String.split* and *string.fromCharCode* methods and stored within the *buf* variable. Finally, the exploit is executed through the *app ['eval']](buf)* instruction, because, if the PDF viewer has two or more plugins (*if app.plugIns.length* >=*2*, the variable *fnc* is equal to *'eval'* at the end of the computation).

- **Action Class** - This class has elements that operate on specific events, like clicking or dragging, which can be subverted to execute malware [15].
- **OpenAction/AA Object Type** - This is a sub group of Action and enables a script to get executed [2] once a file is opened. This feature is often used in combination with the execution of JS code.
- **Embedded Files** - Since it is possible to embed [8] different file formats to a PDF, attackers can use this opportunity to attach malicious files, like executables.
- **RichMedia** - The [8] file format allow to embed media files and flash objects inside them [21], a feature than can be used to attach malign media.
- **PDF Obfuscation** - Specific obfuscation forms [15], like a different representation of strings, for example hexadecimal encoding, inside the PDF are supported by the file format. This feature is used by attackers to evade signature-based detection by AV.

In 2019 Mladenov *et al.* discovered, that the cryptographic protection of digitally signed [3] PDF documents can be bypassed. This attack worked with at least 21 different PDF viewers and six online validation services. Although this specific vulnerability has since been patched by the affected vendors, Mainka *et al.* followed up with an attack [22] that allows to alter the content of a digitally signed document without invalidating the signature. At least 16 PDF viewers were vulnerable to that attack at the time of discovery, and by December 2020 there were still six unpatched PDF applications.
Since digitally signed PDFs are used for the exchange of sensitive and important information, like contracts or invoices, this breach of integrity and authenticity is alarming.

Müller *et al.* analyzed the PDF standards and identified *dangerous paths* [10], which are legitimate features that can be exploited for various malicious actions. Müller *et al.* categorized those attacks into four classes:

- Denial of Service (DoS) attacks affecting the host that processes the document
- Information disclosure attacks where personal data out of the victim´s computer is leaked
- Data manipulation on the victim´s system
- Code execution on the victim´s machine.

In total more than 200 different attack vectors were identified and 26 PDF viewers that are vulnerable to at least one attack. Furthermore the research of [10] resulted in more than ten CVEs.

2.3. PDF Malware Detection

Over the years numerous attempts have been made to reliably detect malicious PDF files. There are various ways of analyzing malware [12], ranging from static analysis to ML-based attempts or a combination of

multiple techniques. A basic overview of the existing techniques and the idea behind them is given in this section.

2.3.1. Static Analysis

Static analysis is the task of doing a complete code analysis [4] and it is very often the first step when it comes to the evaluation of a suspicious file. Often, heuristic-based rules are applied on the content of a file to find potentially malicious action. A very basic approach is to search for keywords, for example *JavaScript*. Since it is not necessary to open or run a suspicious file to do a static analysis, this method is risk-free, but can be very time-consuming if done manually, especially when the code is obfuscated.
This step can be done by a human security analyst with the help of tools, in the case of PDF for example the PDF-Tools[2], or automatically by a static analyzer software, like [23]. A major advantage of static analysis is the low cost in computation and, if done automatically, it is considerably faster than dynamic analysis [11].

2.3.2. Signature-based Detection

This is the most common and basic method [4] used by AV vendors to identify malicious files and falls into the category of static analysis. A security analyst manually inspects a suspicious file and extracts one or more patterns from the byte code, the so called signatures, and store them in a database. When a new, unknown file is analyzed, the code segments of that file are checked for matches in the database, and if a match occurs, the file is marked as malicious.
A major downside [12] of this method is, that malware authors continually improve, obfuscate or mutate the malicious code and evade the signature-based detection method. Furthermore, when zero-day vulnerabilities[3] occur, this technique is not robust.

2.3.3. Dynamic Analysis

This detection method describes the approach of behavior-based malware detection [12], using sandbox technology. The behavior exhibited from a PDF file is monitored in a controlled and safe environment and a detection decision is made, depending on the behavior observed.
Dynamic analysis monitors all the activity of the suspicious file, like access to hard disk, running processes or modifications to the registry. In contrast to static analysis [24], this method greatly increases the detection rate for attacks, as dynamic methods are immune to obfuscation and physical structure falsification. Analysis

[2] `https://blog.didierstevens.com/programs/pdf-tools/`
[3] If no mitigation or patch is available for a vulnerability, it is called a zero-day

in a sandbox, however, is only effective, if the file analyzed behaves as though it was running in a real environment. Attackers have already found solutions to detect if the file containing the malware is opened in a sandbox environment and how to avoid detection. A known anti-sandbox technique is to only exhibit benign behavior if the file is running in a sandbox environment. Further limitations are known for some sandbox tools, like only being able to deal with specific attacks. The time it takes to analyze a file, as well as the computational resources necessary are additional downsides of sandbox environments, which is why most system rely on static analysis [11].

2.3.4. Machine Learning Based Detection Techniques

Due to the fast growth of threats the work required by handwritten signature-based detection increased dramatically and therefore ML-based techniques to detect malware in PDF files have been researched and used more and more in both, academic and industrial environments [4], [25].
Research has shown that such systems could achieve high detection rates when confronted with obfuscated attacks that typically evade simple heuristics [14], [26].
The main goal of a ML-based malware detection system is to decide whether a suspicious file is malicious or benign and they operate by analyzing and classifying information won from the structure or the content of the input file. Nearly all of those approaches share the same basic architecture, also seen in Fig. 2.4:

Pre-Processing

In this part, the PDF files are parsed [11] and the data essential for detection is isolated. Depending on the approach of the developers, either specific keywords or metadata selected, or code like JS or *Actionscript* is extracted etc.
This phase is crucial, as the success of detection relies heavily on the extracted data. There are two major types of pre-processing, *static* and *dynamic*.
The basic principles explained in Section 2.3.1 are valid here as well, the suspicious PDF file is analyzed without it, or its contents being executed. Dynamic pre-processing on the other hand makes use of sandboxing technologies or other techniques to execute the file or its (JS) content in a safe way, as mentioned in Section 2.3.3.
Pre-Processing tools can either be written from scratch and tailor-made for the specific implementation, or a third-party tool is used for this step. Authors of existing ML-based detectors mostly use existing pre-processing parsers. Each parser analyzes three main elements of a PDF file: the *PDF structure*, which refers to all the elements that are not related to embedded code, like direct or indirect objects, metadata etc., the

JS, which means all the embedded JS code inside the file, and finally *embedded files*, which refers to the capability of parsers to extract or inject embedded files.
Every parser has different strengths and weaknesses in the handling of those three elements, sometimes not supporting one element at all, or only partial, like, for example, only analyzing JS code statically.

Feature Extraction

This component acts on the information extracted during the pre-processing step and converts it to a vector of numbers. A vector can, for example, represent the presence of a specific keyword or Application Programming Interface (API) calls, or occurrences of certain elements in the file. Employed feature types fall into three categories: *structural*, *JS-based* and *raw-bytes*.
Structural features target the PDF structure and the idea behind it is that malicious files are structurally different from benign ones. Files that contain malware usually consist of fewer pages or feature specific keywords more often. [27] is one of the approaches that evolves around this idea and proved to be effective. Byte-level features, where sequences of bytes taken as n-grams are taken, were used by the first attempts [28] to develop a PDF malware detector, but they were soon deemed unpractical for the problem at hand. JS-based features are related to the structure of JS code. Most of them concern API calls, information obtained from code behaviour, or lexical characteristics of the code, like the amount of specific operators in the file. The idea behind using JS-based features is that certain functions that perform dangerous operations or the presence of obfuscated code are considered malicious.

Classifier

This is the actual learning algorithm.
In a training phase the parameters are fine-tuned to reduce overfitting[4] and guarantee the highest flexibility against polymorphic variants. All PDF malware detectors are trained with supervision, meaning that it is required to label the training samples (benign or malicious) before the training starts. Depending on which feature-type for classification is chosen, an appropriate learning algorithm has to be selected. Existing research has shown that decisions trees have yielded good results and are used most of the time. Prior work has focused on trying out different combinations of the aforementioned components, for example, using a specific pre-processing module with a specific learning algorithm or feature extractor.[11]

[4] Overfitting [29] describes the inability of a network to learn effectively. This can be due to many reasons, for example a data set that is too small, or an overly complex architecture of the network.

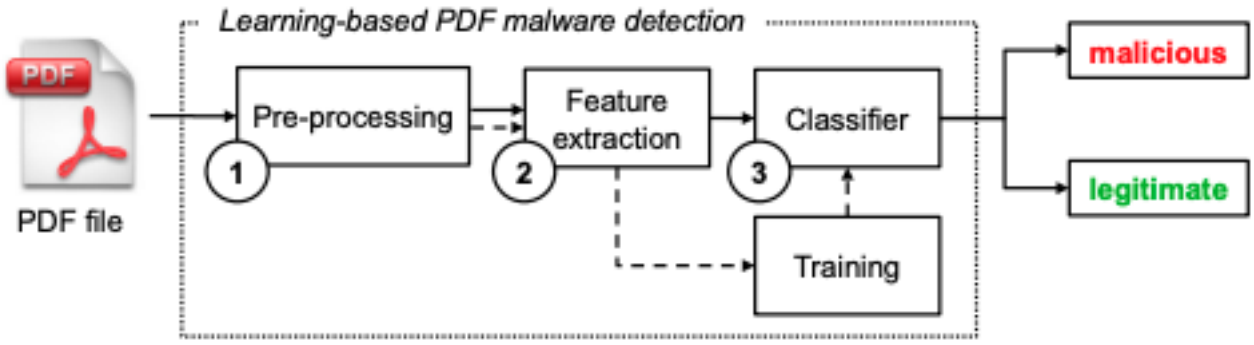

Figure 2.4.: This schematic overview of a learning-based PDF malware detection tool.[11] demonstrates, that an input file has to be pre-processed first, for example, unwanted features are thrown away, or parsing techniques extract the parts wanted for further processing. In a next step, the features are extracted. This works by converting the information obtained in the previous step to a vector of numbers, where a vector represents, for example, a specific keyword. Finally, the data is fed to the classifier, that trains with the data and makes predictions about the input data.

2.4. Machine Learning Background

This section gives an overview of the most important ideas and concepts of ML. First, keywords like AI and ML are explained and the section then focuses on the basics of CNN and how such a network is trained.

2.4.1. Artificial Intelligence

Many definitions and explanations of AI exist, with one of the most famous one coming from Alan Turing [30].

To answer the question if a machine could think, Turing suggested a game, called "the imitation game". Three players take part in the game, one assuming the role of an "interrogator, while the others play a male and a female. The goal of the interrogator is, to determine which player is the male, and which one is the female, without being able to see them, just by asking questions. One of the players tries to assist the interrogator, while the other one has the objective to trick the interrogator into making the wrong decision. Turing then concludes with the thought, that if a machine can take the place of either the male or female player, and the results of the games played do not change, it can be assumed that the machine is able to think.

2.4.2. Machine Learning

Ever since the invention of computers the question arose, if it was possible to build computer system that automatically improve with experience. One of the most basic definitions of ML has been phrased by Tom Mitchell [31], who described it as a computer program that improves its performance at some task through experience. Mitchell [31] followed this up with a more precise definition:“ A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at task in T, as measured by P, improves with experience E.”
An example would be a program that plays chess and improves its performance, measured in the ability to win chess games, through experience by playing games against itself. To get a proper learning problem, that can be solved with ML, the three features, Task T, Performance measure P and Training experience E have to be identified.

2.4.3. Neural Networks

In 1995 Hassoun described NN [32] as parallel computational models, with varying degrees of complexity, comprised of densely interconnected adaptive processing units. Furthermore those networks are fine-grained parallel implementations of nonlinear static or dynamic systems, and one of the key features is the adaptive nature, allowing the networks to “learn by example”, instead of traditional “programming”.
NN are mainly used for tasks like pattern recognition and to simulate the activity of the human brain. Various types and architectures [33], [34] of NN exist, and for the task of malware classification [4], [15] CNN have shown promising results. CNN use convolution instead of general matrix multiplication in at least one of the layers [35].

Basic Concepts

Inspired by the human brain, NN consist of many connected processors, called neurons [36]. The learning process of NN though, is different from that of humans. Given the task of recognizing hand written digits, NN learn by being trained on training examples. These examples are fed to the network as input images, and the output result is then validated. The network develops a system to learn from those examples and infers rules for recognizing the handwritten digits. The accuracy is improved by increasing the size of the training data set.
Each input file is encoded by a special layer [37], the input layer, to get the desired values that can be worked with by the neurons of the network. The decision of a network on how to classify any given input is based

on the weights and the bias of each neuron within the network. Weight can be described as the importance of an input to a neuron in the decision making process, while the bias determines how easy it is to get the neuron fired (triggered). A neurons output can either be “1”, or “0”, depending on whether the sum of the inputs multiplied by the weight, added to the bias is higher or lower than 0. Consequentially, a higher bias means that the neuron is more likely to output a “1”, whereas a negative bias will rather lead to an output of “0”.
It is necessary to update the weights and biases of the neurons to make sure that the network actually learns and evolves. This can be achieved with the help of special learning algorithms that enable an automatic learning process without the programmer having to interact, ultimately leading to the network being capable of solving complex problems. If a neuron could only have input and output values of “1” and “0”, changes to the weights could cause a dramatic change in the output of other neurons, leading to maybe correctly classifying one example that was misclassified earlier, but now misclassifying a lot of other examples that were correctly identified earlier. Therefore, so called sigmoid neurons are used for NN, because their input and output can not only take on binary values, but anything in between and their output can take on different values too. This way small changes in their weights and biases only cause minimal changes in their output, making it possible for the network to learn.

A so called *loss function* [37] is used to monitor and improve the learning process. This function measures the (in)correctness of the output provided by the network, for example how many images were classified wrongly. The best case scenario is an outcome of as close to "0" as possible, meaning that almost no error in classification happened. A high loss on the other hand suggests that the accuracy of the network is low and adjustments to the weights and bias are necessary. The process of calculating the optimal values for the weights and bias is called gradient descent.
First, weights are set to a random value and little by little adjusted, usually by lowering their values, hence descending, until the loss function reaches its minimum. The step size taken in descending is called the learning rate, and in general is set to a very low amount, as to not miss the minimum with a descent too large. The process of updating the weights of neurons is called backpropagation.
The errors of neurons [38] of the hidden layers are calculated by using the error value from the output layer and then passing them backwards to adjust their weights. This whole process is usually done many times, known as epochs, and in general, the more epochs a network is trained, the better the accuracy.

2.4.4. Convolutional Neural Networks

CNNs have been used primarily in various image classification tasks [39], but recent research has shown, that they can be used for malware detection and classification as well.

This section aims to give an understanding of the architecture of a CNN and the way it operates. For ease of understanding, the architecture of a CNN for the task of image classification is explained. The basic principles are the same for a malware classifying CNN.

The first layer [37] is always the input layer, consisting of the input neurons, where the encoding of any given input happens. The encoding needs to happen, because otherwise the next layers can´t use the input. The number of neurons present in the input layer is determined by the size of the input. For example, a network designed to classify black and white images with a size of 28 by 28 pixels would have an input layer consisting of 784 neurons (28x28x1). The last layer is always the output layer, consisting of the output neurons. There are as many output neurons in this layer as there are target classes, for example two, if the network needs to classify into the classes of "benign" and "malicious". In between the input and output layers are so called “hidden” layers, where the neurons in these layers are neither inputs nor outputs. Fig. 2.5 shows a typical CNN architecture of a CNN Note that there are many different architecture types and the following layers might not all exist in every CNN:

Convolutional Layer

These layers are the most basic but most important layers [29] in a CNN. They act as filters and a CNN may contain multiple of these layers. Those filters slide over the layer for the given input data and are used for feature extraction, for example finding color, edges, gradient orientation etc. of an input image. The neurons of the network thus learn to fire when those features are detected. To achieve this, a set amount of filters with a fixed size (kernel size) move across the pixel-matrix of the input image from left to right, line by line, with a fixed step-size. The first Convolutional layer usually consists of 16 or 32 filters and the convoluted output of every filter is a matrix. These matrices are then usually fed to a second Convolutional layer with the same architecture, followed by a (Max)-Pooling layer.

(-Max)-Pooling Layer

A pooling layer [40] is used to produce a summary statistic of its input and to reduce the spatial dimensions of the feature map, but without losing important structural information. To achieve this downsampling, a *max pooling* layer reports only the strongest values of each input feature while the rest is thrown away. An

average pooling layer on the contrary passes the average values on to the next layer. Through convolution and pooling, the size of the input continually shrinks and the amount of filters to recognize high-level signals rises. After the last Max-Pooling layer, one or more Fully-connected layers follow.

Fully-connected/Dense Layer

These layers [40], [41] are used as the final layers of CNNs for classification tasks. Every neuron is connected to every input and output. To feed the result matrix of the previous Convolutional and Max-Pooling layers to a Fully-connected/Dense layer, the output has to be flattened first. A fully-connected layer/dense layer has the task to ultimately classify the input into a corresponding class. Many implementations of CNNs us the softmax rule for the final decision making.

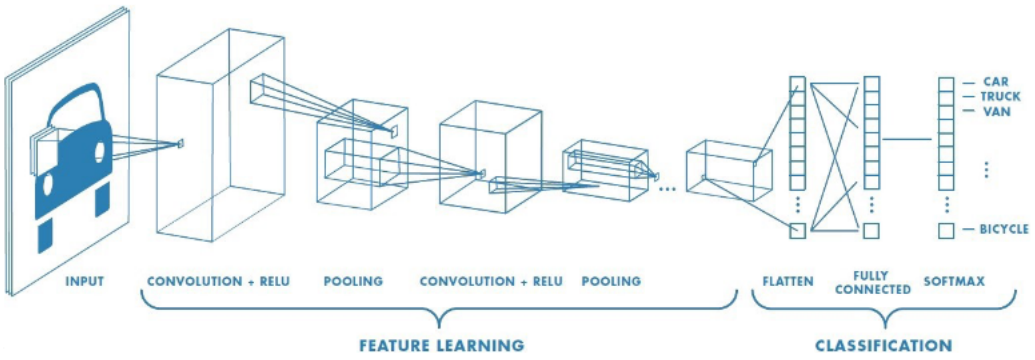

Figure 2.5.: The first layer of this sample architecture [42] is the input layer, where an image is broken down and only the most important features are passed on to the next layers. This is followed by multiple Convolutional layers and Pooling layers, where the input is broken down even further and the neurons are trained to recognize the important features. Finally, the input is flattened and passed on to a Fully-connected layer, where with the help of the softmax function, which is used for multi-class classification problems, a prediction is made.

Activation Functions

Activation functions [43] decide whether a neuron is fired or not. Therefore, it determines the output of a CNN and influences the accuracy and efficiency. The most common activation functions used in CNN are explained briefly.

- **Sigmoid function** The output of this function will always be between 0 and 1. It is not as computationally efficient as other activation functions and is only useful for binary classification tasks.

- **Softmax** This activation function is useful for multi-class classification tasks, as the function calculates the probabilities of each target class across all possible target classes.
- **Rectified Linear Unit (ReLU)** This is a very fast and computationally efficient activation function and often used in hidden layers in Deep Learning.

2.4.5. Basics of CNN Training

To understand everything done in Chapter 4 a few key terms are briefly explained. ML is usually divided into supervised [44] and unsupervised learning, with the difference being, that supervised learning requires human interaction to correctly label the data fed to the CNN, while in unsupervised learning the network tries to find hidden structures in unlabeled data. Supervised learning is used for classification tasks, meaning that the CNN has to predict the target class, e.g. "benign" or "malware".
To train a CNN a **data set** has to be obtained, or crafted manually. A data set consists of files that are used to train and test the CNN and to achieve high accuracy and confidence in the predictions, the quality and quantity of the input data set is crucial. The data set is usually divided into two parts, the training data samples, which are used during training of the CNN and the test samples, that is data that is unknown to the network prior to testing, which are taken to evaluate the accuracy of the network.
When a CNN is trained, the training data set is divided into so called **batches** [45]. The size of these batches can be fixed or adaptive, and is influenced by various factors, like the hardware that is used. After working through a batch, the predictions for every sample in the batch are compared to the expected output and the model is updated to improve accuracy.
Usually a CNN trains for a set amount of rounds, called **epochs**, where each means working through the complete training data set. The number of epochs is mostly anywhere in between 5 and 1000, depending on the model trained and the time available. When the error rate, or loss is at a minimum, training is stopped, as no more improvement can be expected.

3. Related Work

The topic of PDF malware analysis and detection has been around since the beginning of the 21th century, when a virus called "Peachy" [2] was first discovered. Although PDF is used worldwide by billions of people, most research about malware detection has focused on other file formats.
However, over the last decade, with the upcoming of ML, there has been an increase in awareness about the threats of malicious PDF documents and approaches on how to detect such documents have been developed. In this section, relevant work that has been done in the field of PDF automated malware detection and evasion techniques is introduced.

3.1. Automated Malware Detection

Although several tools[1] exist to help security analysts manually dissect a PDF file and determine whether it contains malware or not, manual analysis is time-consuming and not efficient enough considering that billions of PDF documents [1] are created every day.
Therefore, automating this process was necessary and approaches have been made in different directions to solve this problem. Carmony *et al.* describe three different categories of methods of automated PDF malware detectors [46], namely signature-based detectors, metadata and structural features based detectors, and JS based ones.

3.1.1. Signature-Based Detectors

Commercial AV vendors rely heavily on this method to detect malicious PDF files. Shafiq *et al.* proposed a detector that uses statistical anomaly using n-grams [28] and managed to achieve a True Positive (TP) detection rate of more than 80 percent.
In 2013, Lu *et al.* presented "MPScan" [47], a dedicated PDF scanner that can de-obfuscate JS code and then performs a multilevel analysis on the extracted JS strings to detect malware and successfully detected more

[1] https://blog.didierstevens.com/programs/pdf-tools/

than 90 percent of malicious PDF documents. One step of the analysis process of "PDF Scrutinizer" [48] relies on static heuristics to determine whether a file is benign or not.

3.1.2. Metadata and Structrual Features-Based Detectors

This detection method focuses on the name objects [49] in a PDF file, and the approaches often incorporate static analysis tools for pre-processing as well. Maiorca *et al.* propose a detector that extracts metadata [50] and then utilizes random forests for detection, but it was soon discovered, that this approach failed against polymorphic attacks.
Furthermore, [26], [27], [51] demonstrate other approaches in this category.

3.1.3. DL-based Approaches

Since the main focus of this thesis lies on PDF malware detection with a CNN, research in this area is mentioned in more detail here, although, strictly speaking, these approaches fall under either of the categories mentioned earlier. Most of the research in the field of malware detection using DL has been done on other file formats, mainly executable files.
David *et al.* achieved an accuracy of more than 98% with a NN that automatically generates signatures [52] based on the behavior of an executable file. Malware classification through a CNN has been done with great success in [53]–[55].
Fettaya *et al.* were the first researchers to specifically target PDF malware detection with a CNN [4]. The approach used in that paper is the foundation of the practical part of this thesis, as the source code is publicly available. The first 200kB of a file are used as input vectors and the CNN searches for patterns to do the classification. Originally, the CNN achieved a detection rate of more than 90% while maintaining a very low False Positive (FP) rate. Additionally, Fettaya *et al.* claim that even new, undetected malware can be classified correctly with a high accuracy. Other approaches targeting PDF malware detection with a CNN include [15], [56]. The data set developed by Issakhani *et al.* is, among others, used in this thesis.

3.2. The Shadow Attack

Mladenov *et al.* discovered, that digitally signed documents can be altered [3] **after** the signature is applied, **without** invalidating the signature.
Three different attack variations are demonstrated, with the most powerful, dubbed "Hide and Replace", being able to change the entire document. Research on the possibility to detect this kind of malicious

documents with a CNN has, to the best knowledge of the author of this thesis, not been done at the time of writing.

In the "Hide and Replace" attack, the manipulated PDF document contains a second, different document with different content. This second document is hidden and after the victim digitally signs the document, the attackers append a new "Xref table" and "Trailer" section to the file.

Finally, the attackers make the hidden content visible without invalidating the digital signature. Fig. 3.1 demonstrates how this attack can look like.

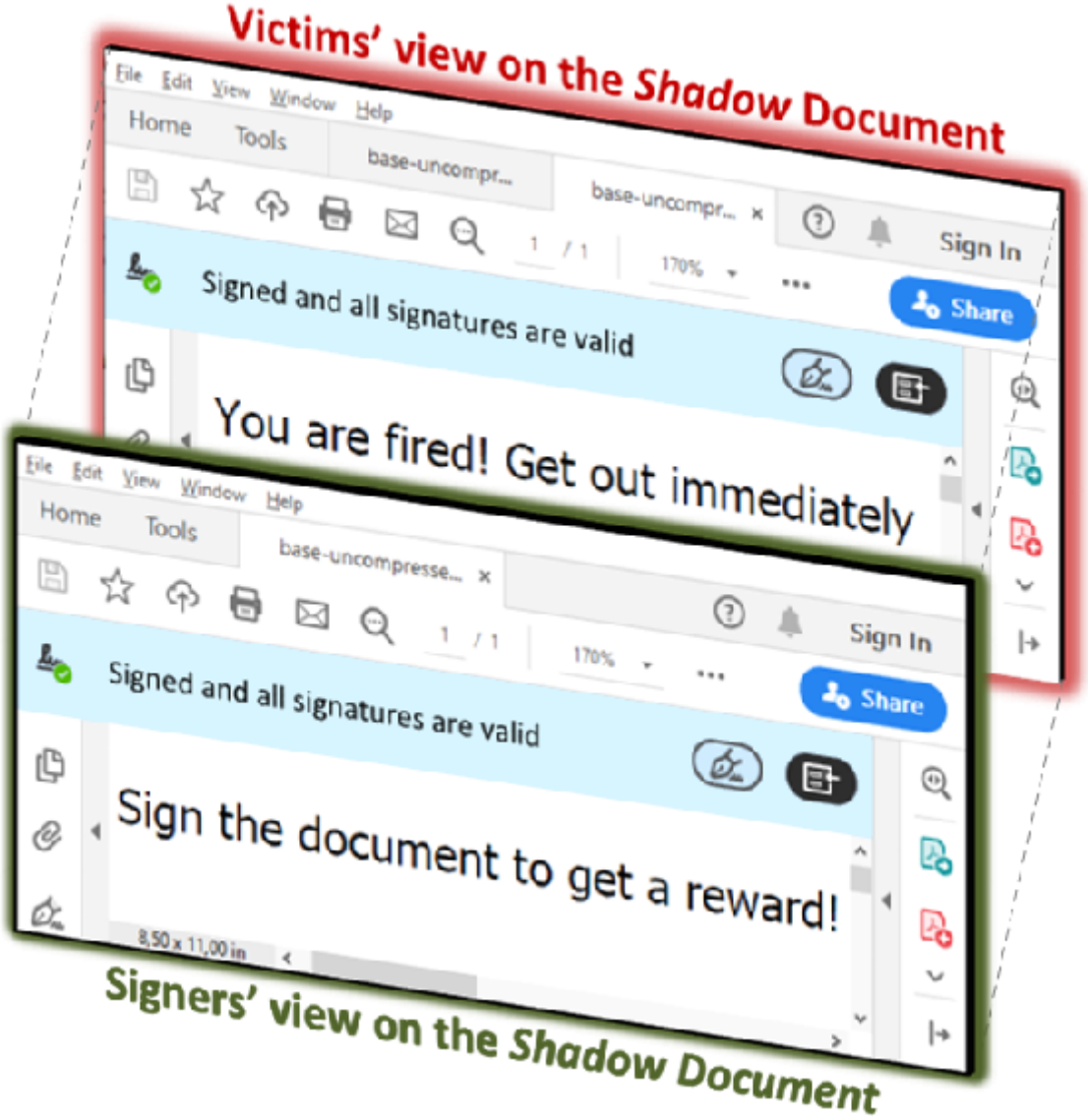

Figure 3.1.: The example shows, that the person that digitally signs the document has a different view [3] of the document. After the digital signature process, the attackers swap the prepared content without invalidating the digital signature.

4. Approach

This chapter describes the exact setup of the experiments done, as well as the results. The complete source code and the selfmade data set are available at the authors github[1] repository.

4.1. Training and Testing of the CNN

This section describes everything that is done in order to train and test the NN, covering the lab setup, as well as the essential parts of the scripts used.

4.1.1. Lab Setup

A Virtual Machine (VM) is used to train the CNN and for all test and validation tasks. Ubuntu is chosen as the OS for all experiments and due to the limitations of the virtualization software, all ML tasks are done by the Central Processing Unit (CPU).
Table 4.1 shows the specification of the host system and the VM.

System	**Host System**	**VM**
OS	Ubuntu 18.04.6 LTS	Ubuntu 22.04 LTS
Kernel	Linux 4.15.0-175 generic	Linux 5.15.0-33-generic
CPU	Intel Xeon CPU E5-2630 v4 @ 2.20GHz	Intel Xeon CPU E5-2630 v4 @ 2.20GHz
CPU cores	40	32
RAM	256 GB	128 GB

Table 4.1.: Lab Setup

[1] `https://github.com/neighbourhoodnerd/saferPDF`

```
1  FROM python:3.10.4-buster
2
3  WORKDIR /usr/src/app
4
5  COPY requirements.txt ./
6
7  RUN apt update
8  RUN apt install libpng-dev libopenblas-dev freetype* -y
9  RUN pip install --no-cache-dir -r requirements.txt
10 RUN mkdir logs
11 RUN id
12
13 RUN whoami
14
15 COPY . .
16
17 ENTRYPOINT [ "python", "./train.py" ]
```

Listing 7: The Dockerfile shows that python version 3.10.4 is used and that a few commands are run, like the installation of the requirements specified in "requirements.txt". Finally the entrypoint is set to "./train.py", which is the training script.

4.1.2. Preparing the Container Environment

All training and testing is done in a Docker[2] container. Listing 7 shows the Dockerfile used and Listing 8 lists the content of the "requirement.txt" file.

The bash script in Listing 9 is used to build the container.

4.1.3. Architecture of the CNN

In [4], Fettaya *et al.* proposed three different architectures for a CNN. The authors claim, that multiple convolutional layers in model "C" enable the CNN to create complex signatures from different parts of the

[2] https://docker.com

```
kd@lab:~/DeePdf$ cat requirements.txt
torch
numpy
pandas
tqdm
matplotlib
scikit-learn
```

Listing 8: The requirements.txt contains all the packages that are needed to train and test the NN.

```
        #!/bin/bash

        sudo docker build -t deepdf .
```

Listing 9: The container is built with the "docker build" command. This command is in a bash script for ease of use.

input file and combine them together. Therefore this architecture is chosen for the experiments in this thesis. The CNN consists of three convolutional layers, each followed by a max-pooling layer and finally a fully-connected layer and a linear layer.

Fig. 4.1 shows the exact architecture of the CNN, while Listing 14 contains the source code of the model.

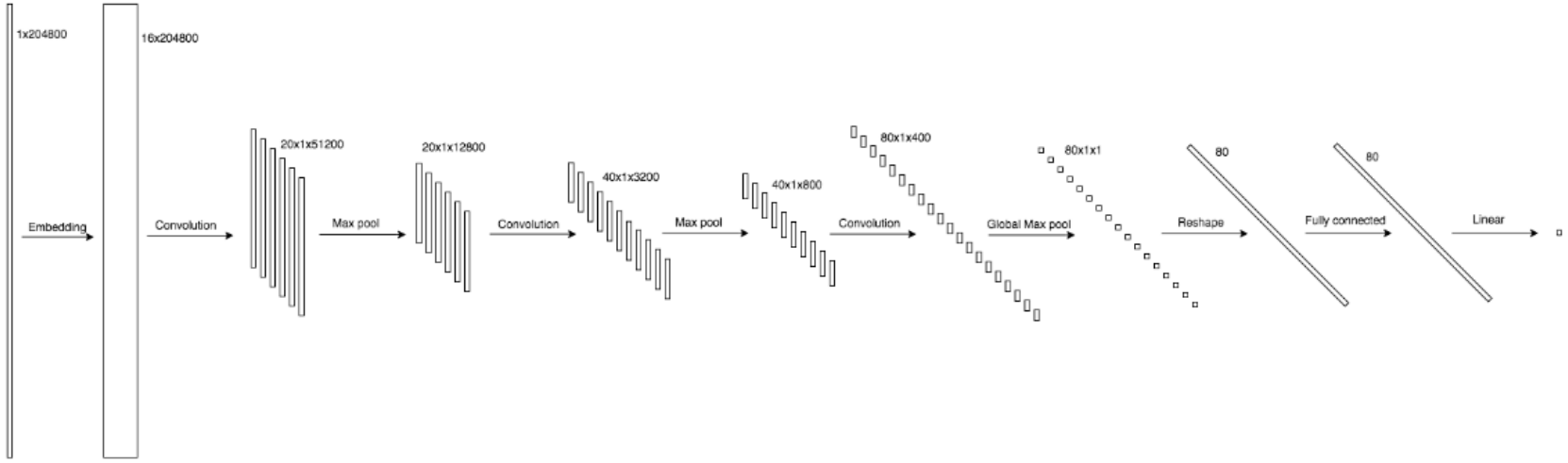

Figure 4.1.: Architecture of the CNN [4]. The network consists of three convolutional layers, each followed by a max-pooling layer. A fully-connected layer and a linear layer are present as well. Each convolutional layer differs from the previous one in either number of kernels, window size or stride.

4.1.4. Data Sets

Experiments are done with different data sets. In total, three data sets are used as the basis training and testing material for the experiments. Table 4.2 shows an overview of the data sets used in this thesis.
A CNN is trained for each data set and validated against the same data set. In total, three models are trained:

- **Model "Contagio"** - This model is trained on the "contagio" data set only.
- **Model "Evasive"** - This model is trained on the "evasive" data set only.
- **Model "Shadow"** - This model is trained on the selfmade shadow attack data set only.

Data Set	Contagio	Evasive	Combined
Benign Samples	9095	9093	9182
Malicious Samples	11102	13101	13212
Shadow Attack Samples	0	0	5983

Table 4.2.: Three different data sets are used in this thesis. "Contagio" data set is a widely used dump of PDF files, containing more than 11000 malicious samples. The "Evasive" data set is a modification of the "Contagio" data set, focusing on malware samples that are more likely to be undetected. The "Combined" data set is a self-made data set that contains more than 5900 samples manipulated with the "shadow attack".

Contagio Data Set

This data set [57] containing more than 20.000 files is used in most research work regarding PDF malware detection, as it is freely available and already labelled. Since it is used so often, it allows for comparability between different approaches and is a main reason why it is considered in this work.
After removing everything without the PDF extension and deduplicating, the data set consists of 9095 benign and 11102 malicious files.

Evasive Data Set

Issakhani *et al.* developed a data set that is based on the contagio data set and samples gathered from VirusTotal[3] and made it publicly available [58]. The authors chose samples that tend to evade common significant features [15], thus making them harder to detect.

[3] `https://virustotal.com`

After downloading, deduplicating and preparing the files as training input for the CNN, the data sets consists of 9093 benign files and 13101 malicious files.

4.1.5. Data Preparation

Prior to feeding the data to the CNN, each file of every data is renamed to the SHA256 hash sum of the corresponding file. This allows for easy deduplication and greater clarity.
Afterwards a Comma-separted values (CSV) file is created, where each line holds the information about one file in the data set. The CSV file contains the name of the file, which is the hash sum and the correct label, which is either "0" for benign, "1" for malicious, or "2" for suspicious and is later used as input for the training and testing scripts.
Since the the models "contagio" and "evasive" are only able to do binary classification, suspicious files are labeled as malicious when these models are used for testing.
Listing 10 shows an example of a CSV file.

4.1.6. Training

Every model is trained for a total of ten epochs on the corresponding data set and batch size is set to 128. Table 4.4 displays the total time necessary to train each model.
Training is done with the source code provided by [4], with slight adjustments to the script being made, mostly due to outdated packages or to modify the verbose output of the script.
Listing 11 shows how a training is started, while Listing 12 displays an excerpt of a logfile during the training process.

```
1  kd@lab:~/DeePdf$ cat combined_dataset.csv | tail -10
2  28169,ff9bc5eb3e4bdfaaef1f42c1f0bf3bdf9bce1089f81f878d8e3e44d96
   ↪ dd6bc21,2
3  28170,ffa52f84d6ba3607dd22ddafc2802983d8e650801dc7049dee5e7839b
   ↪ 21c3b5c,2
4  28171,ffae5a8fdc631509ab694870afe921c50afd4d7d6af4ea2fd2dbe3732
   ↪ 1839a1a,2
5  28172,ffb5895c3fe4a1b6fffcb19fab3f23051f5538c6481a62c07c8a10656
   ↪ 7c27382,2
6  28173,ffc383bb1422ac39fa9d10083f558f5eabf15429f285d5edc28857237
   ↪ 8867bab,2
7  28174,ffe73a2d085f726922948487d140a43be631cf9d9b5b754ab47492b94
   ↪ 03c6b37,2
8  28175,ffeb78eb14b589d0a62bbfbc6425ab34571006121a806e703e568cc5b
   ↪ bc360c2,2
9  28176,ffec7bc022358b0b8e1cdacbb6845e2b70fb528838eaa264981bb5ddc
   ↪ 653850b,2
10 28177,fff641385100b076c0ba66e2ee185458230d67f64e9e33a6943ec70c0
   ↪ 5ded087,2
11 28178,fff87a067ccb94008583f7737224abf1596eaca0faa739025072a53ee
   ↪ c5a2f56,2
```

Listing 10: Every line corresponds to one file of the data set. The first number is the index number of that file, followed by the SHA256 hash value of that file, which is also the name of the file. Finally, the verdict, which is the label for that file, ends the line. In this case, the verdict for each file is "2", indicating that those are files containing the shadow attack.

```
kd@lab:~/DeePdf$ sudo bash run.sh ModelC training_contagio.csv
↪ /data --name model_contagio
```

Listing 11: The training is started via bash script which does folder mappings to the container and sets the entry point to the actual training script. The first parameter passed to the script is the base model that is to be used for training. In this case, verweis auf appendix modelc, "ModelC" is used. This is followed by the path to the CSV file that contains the information of the data set. The next argument is the location where the actual PDF samples are stored. The optional parameter "name" is passed to the script to name the model and log file accordingly.

```
DEBUG:root:Training id: model_contagio
DEBUG:root:Nb epochs: 10
DEBUG:root:Used device cpu
DEBUG:root:Train Test split is done with contagio mode
DEBUG:root:Train size 12092, Valid size 1621,  Test size
↪ 6484
DEBUG:root:Train df sha256: 2c493008a8d351c9b7005fbdec43517
↪ 7ca4e49cc638c0141f8731012a05a3976
DEBUG:root:ModelC (
  (embed): Embedding(257, 16), weights=((257, 16),),
  ↪ parameters=4112
  (conv_1): Conv1d(16, 128, kernel_size=(16,),
  ↪ stride=(16,)), weights=((128, 16, 16), (128,)),
  ↪ parameters=32896
  (pooling): MaxPool1d(kernel_size=4, stride=4, padding=0,
  ↪ dilation=1, ceil_mode=False), weights=(), parameters=0
  (batch_norm1): BatchNorm1d(128, eps=1e-05, momentum=0.1,
  ↪ affine=True, track_running_stats=True),
  ↪ weights=((128,), (128,)), parameters=256
```

```
12          (conv_2): Conv1d(128, 128, kernel_size=(16,),
            ↪  stride=(16,)), weights=((128, 128, 16), (128,)),
            ↪  parameters=262272
13          (relu2): ReLU(), weights=(), parameters=0
14          (pooling2): MaxPool1d(kernel_size=4, stride=4, padding=0,
            ↪  dilation=1, ceil_mode=False), weights=(), parameters=0
15          (batch_norm2): BatchNorm1d(128, eps=1e-05, momentum=0.1,
            ↪  affine=True, track_running_stats=True),
            ↪  weights=((128,), (128,)), parameters=256
16          (conv_3): Conv1d(128, 128, kernel_size=(4,),
            ↪  stride=(2,)), weights=((128, 128, 4), (128,)),
            ↪  parameters=65664
17          (relu3): ReLU(), weights=(), parameters=0
18          (pooling3): MaxPool1d(kernel_size=24, stride=24,
            ↪  padding=0, dilation=1, ceil_mode=False), weights=(),
            ↪  parameters=0
19          (fc_1): Linear(in_features=128, out_features=128,
            ↪  bias=True), weights=((128, 128), (128,)),
            ↪  parameters=16512
20          (batch_norm3): BatchNorm1d(128, eps=1e-05, momentum=0.1,
            ↪  affine=True, track_running_stats=True),
            ↪  weights=((128,), (128,)), parameters=256
21          (fc_2): Linear(in_features=128, out_features=1,
            ↪  bias=True), weights=((1, 128), (1,)), parameters=129
22          (sigmoid): Sigmoid(), weights=(), parameters=0
23        )
24        DEBUG:root:step,tr_loss, tr_acc, val_loss, val_acc, time
25
26        DEBUG:root:Started timing
27        DEBUG:root:Step:0 | Loss:0.892018 | Acc:0.2656 | Time:98.76
28        DEBUG:root:Step:15 | Loss:0.240854 | Acc:0.9260 | Time:6.64
29        DEBUG:root:Step:30 | Loss:0.120697 | Acc:0.9641 | Time:6.93
```

```
30          ....
31          ....
```

Listing 12: The logfile output shows, that the epochs are set to 10 and that the CPU is used. The data set is then split into a training set, a validation set and a test set. The training set is used to train the CNN, and one epoch lasts as long as it takes to feed all samples in the training data set to the network. After each epoch, the accuracy of the model is validated with a validation data set, and finally, after the set number of epochs reached, the model is evaluated with the test training set. Before the actual training starts, the output shows the exact architecture of the model. Finally, the progress during the training can be seen, as every 15 steps the current accuracy and loss value are written to the file.

4.1.7. Tool for Validation and Testing

Although the source code of [4] includes validation of the accuracy within the training process, a separate script is written to be able to manually feed files to predict to the model and to add additional functionality. The features of the self-developed tool include:

- Feed any number of labelled files to the classifier
- Multi-Class classification if the model allows it
- Transformation of the input data to the necessary format
- Construction of a data set out of labelled PDF files
- Randomization of test and validation data
- Results and Statistics in a logfile

4.1.8. Validation of Training Accuracy

To validate the accuracy, each model is tasked with classifying 1000 random samples of the corresponding data set. The script then returns the values for TP, FP, True Negative (TN) and False Negative (FN), as well as the total amounts of predicted files. This process is repeated ten times for every trained model.

4.2. The Shadow Attack

Mainka *et al.* demonstrated, that it is possible to craft PDF documents [22] that can be changed **after** being digitally signed. A more detailed explanation of this exploit technique is given in Section 4.2.

The authors developed three different attacks, with "Hide and Replace" being considered the strongest method. This method is used in this thesis and it is evaluated if a CNN can be trained to detect documents that are manipulated this way. To achieve this, a data set has to be created first and the network is trained on those samples. Afterwards, the validation process explained and used in Section 4.1.8 is executed. Fig. 4.2 shows an example of a document prepared with the shadow attack and Fig. 4.3 demonstrates, that the manipulated file is not considered harmful by any AV.

4.2.1. Creation of Data Set

As, to the best of the authors knowledge, no comprehensive data set of "shadowed" documents is available, a new one is created. First, the source code of the original attack [59] is adjusted to create more than five thousand modified PDF documents. For a greater variety in the data set, the "base" files of the attack are benign files of the contagio data set. Every chosen benign file is modified to contain hidden content of random size, while displaying other, harmless random content. A script to automatically repeat this process is used and more than 5900 shadowed PDF files are created. These newly created files are combined with the "evasive" data set and form a combined data set.

4.2.2. Testing Models against Data Set

To check if the models trained on the "contagio" and "evasive" data sets already detect the shadow attack with a satisfying accuracy, the process explained in Section 4.1.8 is executed. As those networks are trained for binary classification only, all malicious "shadow" files are put into the class "malicious". Table 4.3 shows the outcome of the test, indicating that the CNN is not able to reliably classify PDF documents manipulated with this kind of attack as malicious. The "Contagio" model has a detection rate of less than one percent, while the "Evasive" model correctly identifies about two thirds of the files as malicious, hinting that the improved data set that this model was trained on has similarities with the shadow attack.

4.2.3. Adaption of the Architecture and Re-Training the CNN

As the shadow attack is not malware per definition, further outlined by the fact that not a single AV product flags manipulated files as malicious (see Fig. 4.3), the CNN is trained to distinguish between three different categories: Benign, Malware and Suspicious. The latter category is used for PDF files that have potentially been tempered with like in the shadow attack, by hiding elements or certain content. The architecture of the CNN has to be adapted to achieve this. The loss and activation functions are changed to enable multi-class

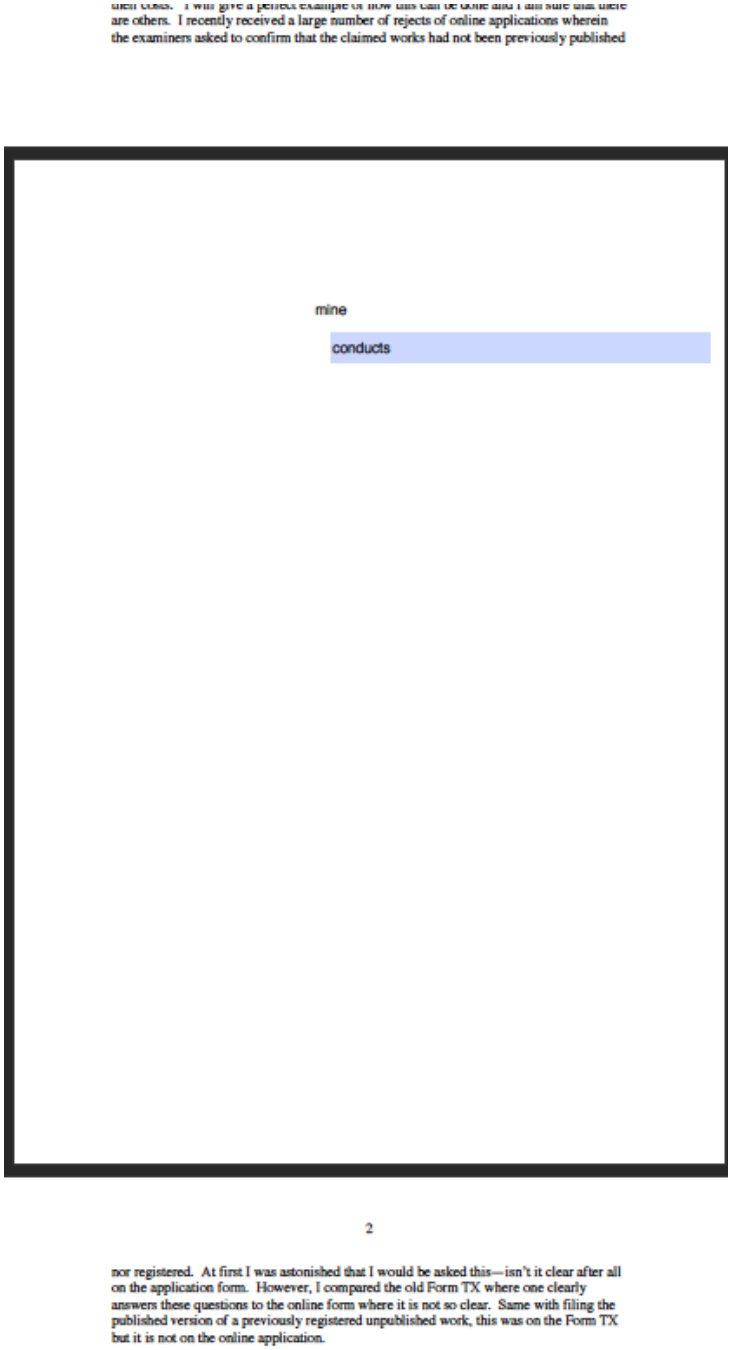

Figure 4.2.: An extra page is inserted into a benign PDF file. This extra site contains form fields including random text, in this case "mine" and "conducts". This is how the victim will see the document. After the document is digitally signed and returned to the attacker, the hidden content is made visible.

classification. Listing 14 shows the complete, modified code of the architecture of the multi-classification model.

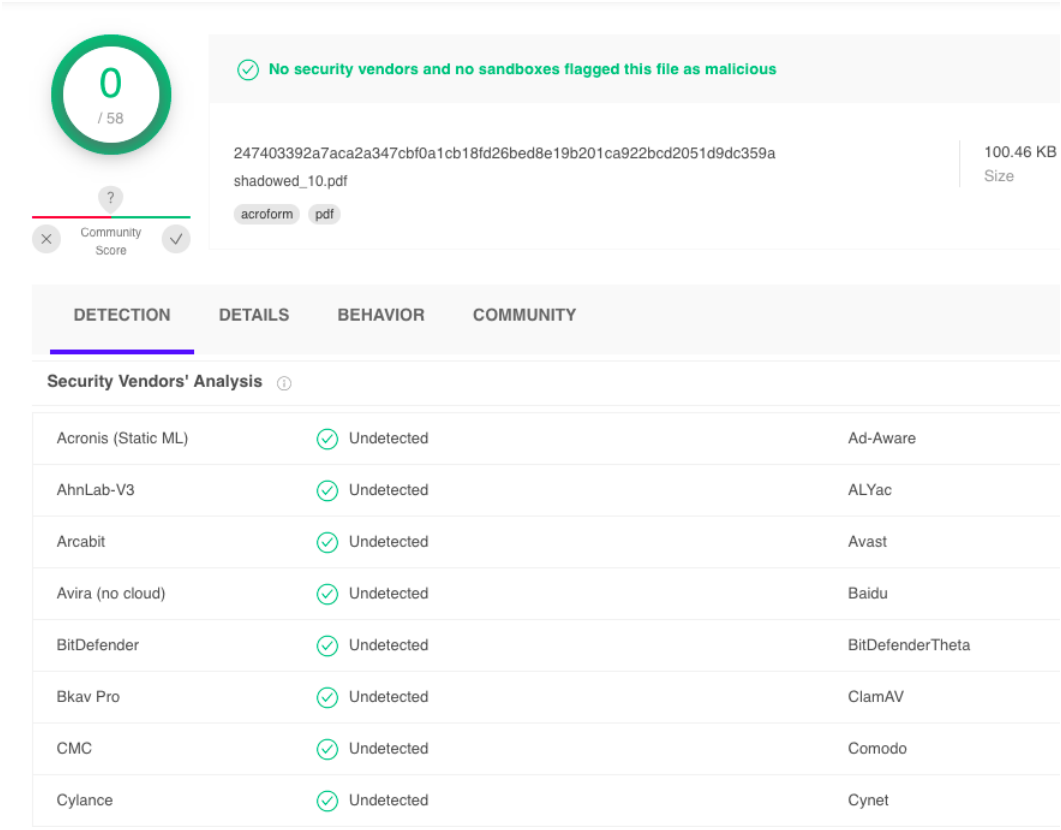

Figure 4.3.: A manipulated document created for the data set (Section 4.2.1) is uploaded to VirusTotal and scanned by 58 AV products. The file is not detected as malicious by a single vendor.

Model Name	**Contagio**	**Evasive**
Accuracy in %	0,82	68,04
FP in %	0	0
FN in %	99,13	31,96

Table 4.3.: Models "Contagio" and "Evasive" are tested with 1000 PDF documents that are all modified with the shadow attack. All files are labeled as malicious, which is why there can not be any FP results. The test is repeated ten times for both models and the average values are shown in the table.

While loss stagnates during training after about three epochs, training accuracy keeps improving slightly, which could be due to the change of the loss function. Training this model takes significantly longer, potentially caused by the larger data set and the change in architecture. After training is completed, the CNN is validated the same way described in Section 4.1.8.

4.3. Results

This section contains all the results of the tests done in this thesis.

Model Name	**Contagio**	**Evasive**	**Shadow**
Epochs Trained	10	10	10
Batch Size	128	128	128
Total Training Time (minutes)	161	177	320
Accuracy on Contagio Data Set - in %	99,76	99,87	99,36
FP rate on Contagio Data Set - in %	0,02	0,05	0,47
FN rate on Contagio Data Set - in %	0,22	0,08	0,17
Accuracy on Evasive Data Set - in %	96,17	99,56	99,34
FP rate on Evasive Data Set - in %	0,02	0,06	0,36
FN rate on Evasive Data Set - in %	3,81	0,38	0,30
Accuracy on Combined Data Set - in %	81,67	94,77	97,86
FP rate on Combined Data Set - in %	0,00	0,05	0,49
FN rate on Combined Data Set - in %	18,33	5,18	1,65

Table 4.4.: Results of Validation Tests

Table 4.4 shows, that the "Shadow" model that has the ability to classify into three different categories performs best on the "Combined" data set. This model has a significantly lower FN rate on the "Combined" data set than the other two models trained, which is due to being trained on this data set, thus obviously having learned to identify files manipulated with the shadow attack. Interestingly, model "Shadow" even outperforms model "Contagio" on the "Contagio" data set in terms of FN, indicating that this model is oversensitive to malicious files.

This is further highlighted by the fact, that model "Shadow" has a higher FP rate across all data sets.

The results clearly show, that the data set developed in this thesis leads to a significant improvement in the prediction accuracy when a model is faced with files that have been manipulated with the "shadow attack". The model trained on the "Combined" data set managed to achieve and overall accuracy of more than 97% on this data set, manily due to the fact that the other models can not detect the "shadow attack" properly, as highlighted in Table 4.3. Therefore it is safe to assume that the quality of the data set is of great significance when it comes to training a NN that is used for malware prediction and that a model can be trained to detect a specific attack.

5. Conclusion

This final chapter concludes the thesis and outlines future work.
PDF malware is a real threat and a potential point of entry for attackers. The liberties this file format grants can be used to insert malware into a PDF document or to launch other, hidden forms of attacks, like shadow attacks.
As billions of PDF documents are created each day, it is impossible for security analysts to find every malicious document before it is spread in the world wide web. Furthermore, many AV products use signature-based detection techniques that are useless when faced with unknown threats and many companies can´t afford or implement other detection technologies, like a commercial sandboxing system. Therefore researchers have turned to ML and make of DL to automatically detect malware.
This thesis has shown that CNN can be trained to not only detect classic PDF malware, but to also identify suspicious files that have been modified with the shadow attack. This attempt could prove fruitful in the fight against PDF malware and aid in the protection of billions of users.

5.1. Future Work

AI and especially the research field of ML offer huge potential in the fight against malware. This thesis merely scratched the surface and focused on PDF documents only.
Extending the scope to other file formats and different attack vectors is of great importance for future work.

5.1.1. Build a Web-Based PDF Classifier

An idea to build on this work is to build a web-based PDF classifier that people can use to upload files and let the classifier predict whether the file is benign, malware or suspicious. By implementing a feedback functionality, the classifier could be updated and retrained based on the feedback.
This could help especially small companies or private users to be safer when dealing with potentially dangerous files.

5.1.2. Analyzing the Impact of Adversarial Attacks

Adversarial attacks [7], [60] are a major problem for NN used for classification tasks. To build a robust PDF malware detection system, the effects of adversarial examples need to be evaluated.

A. Appendix

A.1. Appendix: Model Architecture of Model "Contagio"

```
import torch.nn as nn
import torch
import numpy as np

INPUT_LENGTH = 1024*200
INPUT_HEIGHT = 257

def compute_output_size(input_length, window, stride):
    return int((input_length - window)/stride)+1

class ModelC(nn.Module):
    def __init__(self,input_height=INPUT_HEIGHT,
    ↪  input_length=INPUT_LENGTH):
        super().__init__()
        embedding_size = 16
        self.embed = nn.Embedding(input_height, embedding_size)
        self.conv_1 = nn.Conv1d(embedding_size, 128, 16, 16)
        self.pooling = nn.MaxPool1d(4)
        output_size = compute_output_size(input_length, 16, 16) // 4
        self.batch_norm1 = nn.BatchNorm1d(128)
        self.conv_2 = nn.Conv1d(128, 128, 16, 16)
        self.relu2 = nn.ReLU()
        self.pooling2 = nn.MaxPool1d(4)
```

```
        output_size = compute_output_size(output_size, 16, 16) // 4
        self.batch_norm2 = nn.BatchNorm1d(128)
        self.conv_3 = nn.Conv1d(128, 128, 4, 2)
        self.relu3 = nn.ReLU()
        output_size = compute_output_size(output_size, 4, 2)
        self.pooling3 = nn.MaxPool1d(output_size)
        self.fc_1 = nn.Linear(128,128)
        self.batch_norm3 = nn.BatchNorm1d(128)
        #self.dropout = nn.Dropout(.25)
        self.fc_2 = nn.Linear(128,1)
        self.sigmoid = nn.Sigmoid()

    def forward(self,x):
        x = self.embed(x)
        x = torch.transpose(x, 1, 2)
        x = self.pooling(nn.functional.relu(self.conv_1(x)))
        x = self.batch_norm1(x)
        x = self.pooling2(nn.functional.relu(self.conv_2(x)))
        x = self.batch_norm2(x)
        x = nn.functional.relu(self.conv_3(x))
        x = self.pooling3(x)
        x = x.view(-1,128)
        x = nn.functional.relu(self.fc_1(x))
        x = self.batch_norm3(x)
        #x = self.dropout(x)
        x = self.fc_2(x)
        x = self.sigmoid(x)
        return x
```

Listing 13: Architecture of Model "Contagio"

A.2. Appendix: Model Architecture of Model "Shadow"

```
import torch.nn as nn
import torch
import numpy as np

INPUT_LENGTH = 1024*200
INPUT_HEIGHT = 257

def compute_output_size(input_length, window, stride):
    return int((input_length - window)/stride)+1

class ModelC(nn.Module):
    def __init__(self,input_height=INPUT_HEIGHT,
     ↪  input_length=INPUT_LENGTH):
        super().__init__()
        embedding_size = 16
        self.embed = nn.Embedding(input_height, embedding_size)
        self.conv_1 = nn.Conv1d(embedding_size, 128, 16, 16)
        self.pooling = nn.MaxPool1d(4)
        output_size = compute_output_size(input_length, 16, 16) // 4
        self.batch_norm1 = nn.BatchNorm1d(128)
        self.conv_2 = nn.Conv1d(128, 128, 16, 16)
        self.relu2 = nn.ReLU()
        self.pooling2 = nn.MaxPool1d(4)
        output_size = compute_output_size(output_size, 16, 16) // 4
        self.batch_norm2 = nn.BatchNorm1d(128)
        self.conv_3 = nn.Conv1d(128, 128, 4, 2)
        self.relu3 = nn.ReLU()
        output_size = compute_output_size(output_size, 4, 2)
        self.pooling3 = nn.MaxPool1d(output_size)
        self.fc_1 = nn.Linear(128,128)
```

```
30          self.batch_norm3 = nn.BatchNorm1d(128)
31          self.dropout = nn.Dropout(.25)
32          self.fc_2 = nn.Linear(128,3)
33          self.softmax = nn.Softmax(dim = 1)
34
35      def forward(self,x):
36          x = self.embed(x)
37          x = torch.transpose(x, 1, 2)
38          x = self.pooling(nn.functional.relu(self.conv_1(x)))
39          x = self.batch_norm1(x)
40          x = self.pooling2(nn.functional.relu(self.conv_2(x)))
41          x = self.batch_norm2(x)
42          x = nn.functional.relu(self.conv_3(x))
43          x = self.pooling3(x)
44          x = x.view(-1,128)
45          x = nn.functional.relu(self.fc_1(x))
46          x = self.batch_norm3(x)
47          x = self.dropout(x)
48          x = self.fc_2(x)
49          x = self.softmax(x)
50          return x
```

Listing 14: Architecture of Model "Shadow". The difference is the use of "Dropout" as well as the "Softmax" function, which is necessary for multi-class classification.

List of Figures

List of Listings

List of Tables

Bibliography

[1] Phil Ydens, *Keynote technical conference 2015*, 2015. [Online]. Available: https://www.youtube.com/watch?v=5Axw6OGPYHw (visited on 05/14/2022).

[2] Miles Munson and Jesse Cross, "Deep PDF parsing to extract features for detecting embedded malware.," Office of Scientific and Technical Information (OSTI), Tech. Rep., Sep. 2011. DOI: 10.2172/1030303.

[3] Vladislav Mladenov, Christian Mainka, Karsten Meyer zu Selhausen, Martin Grothe, and Jörg Schwenk, "1 trillion dollar refund," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ACM, Ed., Nov. 2019. DOI: 10.1145/3319535.3339812.

[4] Raphael Fettaya and Yishay Mansour, *Detecting malicious pdf using cnn*, arXiv, Ed., 2020. DOI: 10.48550/ARXIV.2007.12729.

[5] Adobe Inc., *What does pdf mean?* 2022. [Online]. Available: https://www.adobe.com/acrobat/about-adobe-pdf.html (visited on 05/10/2022).

[6] Mark Watson, *Statistics of common crawl monthly archives*, 2022. [Online]. Available: https://commoncrawl.github.io/cc-crawl-statistics/plots/mimetypes (visited on 05/10/2022).

[7] Ho Bae, Younghan Lee, Yohan Kim, Uiwon Hwang, Sungroh Yoon, and Yunheung Paek, "Learn2evade: Learning-based generative model for evading pdf malware classifiers," *IEEE Transactions on Artificial Intelligence*, vol. 2, no. 4, pp. 299–313, 2021. DOI: 10.1109/TAI.2021.3103139.

[8] ISO Central Secretary, "Document management — portable document format — part 2: Pdf 2.0," en, International Organization for Standardization, Geneva, CH, Standard ISO 32000-2:2020, 2020. [Online]. Available: https://www.iso.org/standard/75839.html.

[9] Leonard Rosenthol, *Developing with PDF: Dive Into the Portable Document Format*, Inc." " O'Reilly Media, Ed. 2013, ISBN: 9781449327910.

[10] Jens Müller, Dominik Noss, Christian Mainka, Vladislav Mladenov, and Jörg Schwenk, "Processing dangerous paths – on security and privacy of the portable document format," in *Proceedings 2021 Network and Distributed System Security Symposium*, Internet Society, Ed., 2021. DOI: 10.14722/ndss.2021.23109.

[11] Davide Maiorca, Battista Biggio, and Giorgio Giacinto, "Towards adversarial malware detection," *ACM Computing Surveys*, vol. 52, no. 4, Association for Computing Machinery (ACM), Ed., pp. 1–36, Jul. 2020. DOI: 10.1145/3332184.

[12] Jason Zhang, *Mlpdf: An effective machine learning based approach for pdf malware detection*, arXiv, Ed., 2018. DOI: 10.48550/ARXIV.1808.06991.

[13] Sami Rautiainen, "A look at portable document format vulnerabilities," *Information Security Technical Report*, vol. 14, no. 1, pp. 30–33, Feb. 2009. DOI: 10.1016/j.istr.2009.04.001.

[14] Igino Corona, Davide Maiorca, Davide Ariu, and Giorgio Giacinto, "Lux0r: Detection of malicious pdf-embedded javascript code through discriminant analysis of api references," in *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop - AISec '14*, ACM Press, Ed., 2014. DOI: 10.1145/2666652.2666657.

[15] Maryam Issakhani, Princy Victor, Ali Tekeoglu, and Arash Lashkari, "PDF malware detection based on stacking learning," in *Proceedings of the 8th International Conference on Information Systems Security and Privacy*, SCITEPRESS - Science and Technology Publications, Eds., 2022. DOI: 10.5220/0010908400003120.

[16] Yuanzhang Li, Yaxiao Wang, Ye Wang, Lishan Ke, and Yu-an Tan, "A feature-vector generative adversarial network for evading PDF malware classifiers," *Information Sciences*, vol. 523, Elsevier BV, Ed., pp. 38–48, Jun. 2020. DOI: 10.1016/j.ins.2020.02.075.

[17] Daiping Liu, Haining Wang, and Angelos Stavrou, "Detecting malicious javascript in PDF through document instrumentation," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, IEEE, Ed., Jun. 2014. DOI: 10.1109/dsn.2014.92.

[18] Young-Seob Jeong, Jiyoung Woo, and Ah Reum Kang, "Malware detection on byte streams of PDF files using convolutional neural networks," *Security and Communication Networks*, vol. 2019, Hindawi Limited, Ed., pp. 1–9, Apr. 2019. DOI: 10.1155/2019/8485365.

[19] Zacharias Tzermias, Giorgos Sykiotakis, Michalis Polychronakis, and Evangelos P. Markatos, "Combining static and dynamic analysis for the detection of malicious documents," in *Proceedings of the Fourth European Workshop on System Security - EUROSEC '11*, ACM Press, Ed., 2011. DOI: 10.1145/1972551.1972555.

[20] Ronald Brandis and Luke Steller, "Threat modelling adobe pdf," DEFENCE SCIENCE and TECHNOLOGY ORGANISATION EDINBURGH (AUSTRALIA) COMMAND . . ., Tech. Rep., 2012.

[21] Yuning Cui, Yi Sun, Jun Luo, Yonghui Huang, Yuxuan Zhou, and Xuelei Li, "MMPD: A novel malicious PDF file detector for mobile robots," *IEEE Sensors Journal*, Institute of Electrical and Electronics Engineers (IEEE), Eds., pp. 1–1, 2020. DOI: 10.1109/jsen.2020.3029083.

[22] Christian Mainka, Vladislav Mladenov, and Simon Rohlmann, "Shadow attacks: Hiding and replacing content in signed PDFs," in *Proceedings 2021 Network and Distributed System Security Symposium*, Internet Society, Ed., 2021. DOI: 10.14722/ndss.2021.24117.

[23] Pavel Laskov and Nedim Šrndić, "Static detection of malicious JavaScript-bearing PDF documents," in *Proceedings of the 27th Annual Computer Security Applications Conference on - ACSAC '11*, ACM Press, Ed., 2011. DOI: 10.1145/2076732.2076785.

[24] Caglar Ulucenk, Vijay Varadharajan, Venkat Balakrishnan, and Udaya Tupakula, "Techniques for analysing PDF malware," in *2011 18th Asia-Pacific Software Engineering Conference*, IEEE, Ed., Dec. 2011. DOI: 10.1109/apsec.2011.41.

[25] Kaspersky Enterprise Cybersecurity, *Machine learning for malware detection*, 2018.

[26] Charles Smutz and Angelos Stavrou, "Malicious pdf detection using metadata and structural features," in *Proceedings of the 28th Annual Computer Security Applications Conference*, Association for Computing Machinery, Ed., ser. ACSAC '12, Orlando, Florida, USA, 2012, pp. 239–248, ISBN: 9781450313124. DOI: 10.1145/2420950.2420987.

[27] Nedim Šrndić and Pavel Laskov, "Hidost: A static machine-learning-based detector of malicious files," *EURASIP Journal on Information Security*, vol. 2016, no. 1, Springer Science and Business Media LLC, Eds., Sep. 2016. DOI: 10.1186/s13635-016-0045-0.

[28] M. Zubair Shafiq, Syed Ali Khayam, and Muddassar Farooq, "Embedded malware detection using markov n-grams," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer Berlin Heidelberg, Ed., 2008, pp. 88–107. DOI: 10.1007/978-3-540-70542-0_5.

[29] Keiron O'Shea and Ryan Nash, *An introduction to convolutional neural networks*, arXiv, Ed., 2015. DOI: 10.48550/ARXIV.1511.08458.

[30] Alan M Turing, "Computing machinery and intelligence," in *Parsing the turing test*, Springer, Ed., 2009, pp. 23–65.

[31] M Tom, "Mitchell: Machine learning," *1997 Burr Ridge*, vol. 45, no. 37, IL McGraw Hill, Ed., pp. 870–877, 1997.

[32] Muhamad H Hassoun, *Fundamentals of artificial neural networks*, MIT Press, Ed., ser. A Bradford book. May 1995.

[33] Ejaz Ahmed, Michael Jones, and Tim K. Marks, "An improved deep learning architecture for person re-identification," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Ed., Jun. 2015. DOI: 10.1109/cvpr.2015.7299016.

[34] Dong Yi, Zhen Lei, and Stan Z. Li, *Deep metric learning for practical person re-identification*, arXiv, Ed., 2014. DOI: 10.48550/ARXIV.1407.4979.

[35] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning*, MIT press, Ed. 2016.

[36] Jürgen Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, Elsevier BV, Ed., pp. 85–117, Jan. 2015. DOI: 10.1016/j.neunet.2014.09.003.

[37] M.A. Nielsen, *Neural Networks and Deep Learning*, Determination Press, Ed. 2015. [Online]. Available: https://books.google.at/books?id=STDBswEACAAJ.

[38] Dan C. Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *IJCAI*, 2011. DOI: 10.5591/978-1-57735-516-8/IJCAI11-210.

[39] Andrew McDole, Mahmoud Abdelsalam, Maanak Gupta, and Sudip Mittal, "Analyzing cnn based behavioural malware detection techniques on cloud iaas," *arXiv*, 2020. DOI: 10.48550/ARXIV.2002.06383.

[40] Jonas Teuwen and Nikita Moriakov, "Convolutional neural networks," in *Handbook of Medical Image Computing and Computer Assisted Intervention*, Elsevier, Ed., 2020, pp. 481–501. DOI: 10.1016/b978-0-12-816176-0.00025-9.

[41] C.-C. Jay Kuo, "Understanding convolutional neural networks with a mathematical model," *Journal of Visual Communication and Image Representation*, vol. 41, Elsevier BV, Ed., pp. 406–413, Nov. 2016. DOI: 10.1016/j.jvcir.2016.11.003.

[42] Sumit Saha, *A comprehensive guide to convolutional neural networks - the eli5 way*, Towards Data Science, Ed., Dec. 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (visited on 05/09/2022).

[43] Aurélien Géron, *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Unsupervised learning techniques*. Jun. 2022, ISBN: 9781492032649.

[44] Fabio Nelli, "Machine learning with scikit-learn," in *Python Data Analytics*, Apress, Ed., 2015, pp. 237–264. DOI: 10.1007/978-1-4842-0958-5_8.

[45] Aditya Devarakonda, Maxim Naumov, and Michael Garland, *Adabatch: Adaptive batch sizes for training deep neural networks*, 2017. DOI: 10.48550/ARXIV.1712.02029.

[46] Curtis Carmony, Mu Zhang, Xunchao Hu, Abhishek Vasisht Bhaskar, and Heng Yin, "Extract me if you can: Abusing PDF parsers in malware detectors," in *Proceedings 2016 Network and Distributed System Security Symposium*, Internet Society, Ed., 2016. DOI: 10.14722/ndss.2016.23483.

[47] Xun Lu, Jianwei Zhuge, Ruoyu Wang, Yinzhi Cao, and Yan Chen, "De-obfuscation and detection of malicious PDF files with high accuracy," in *2013 46th Hawaii International Conference on System Sciences*, IEEE, Ed., Jan. 2013. DOI: 10.1109/hicss.2013.166.

[48] Florian Schmitt, Jan Gassen, and Elmar Gerhards-Padilla, "PDF scrutinizer: Detecting JavaScript-based attacks in PDF documents," in *2012 Tenth Annual International Conference on Privacy, Security and Trust*, IEEE, Ed., Jul. 2012. DOI: 10.1109/pst.2012.6297926.

[49] Priyansh Singh, Shashikala Tapaswi, and Sanchit Gupta, "Malware detection in PDF and office documents: A survey," *Information Security Journal: A Global Perspective*, vol. 29, no. 3, pp. 134–153, Feb. 2020. DOI: 10.1080/19393555.2020.1723747.

[50] Davide Maiorca, Giorgio Giacinto, and Igino Corona, "A pattern recognition system for malicious PDF files detection," in *Machine Learning and Data Mining in Pattern Recognition*, Springer Berlin Heidelberg, Ed., 2012, pp. 510–524. DOI: 10.1007/978-3-642-31537-4_40.

[51] Davide Maiorca, Davide Ariu, Igino Corona, and Giorgio Giacinto, "A structural and content-based approach for a precise and robust detection of malicious PDF files," in *Proceedings of the 1st International Conference on Information Systems Security and Privacy*, SCITEPRESS - Science and Technology Publications, Eds., 2015. DOI: 10.5220/0005264400270036.

[52] Omid E. David and Nathan S. Netanyahu, "DeepSign: Deep learning for automatic malware signature generation and classification," in *2015 International Joint Conference on Neural Networks (IJCNN)*, IEEE, Ed., Jul. 2015. DOI: 10.1109/ijcnn.2015.7280815.

[53] Mahmoud Kalash, Mrigank Rochan, Noman Mohammed, Neil D. B. Bruce, Yang Wang, and Farkhund Iqbal, "Malware classification with deep convolutional neural networks," in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, IEEE, Ed., Feb. 2018. DOI: 10.1109/ntms.2018.8328749.

[54] Danish Vasan, Mamoun Alazab, Sobia Wassan, Hamad Naeem, Babak Safaei, and Qin Zheng, "IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture," *Computer Networks*, vol. 171, Elsevier BV, Ed., p. 107 138, Apr. 2020. DOI: 10.1016/j.comnet.2020.107138.

[55] Wai Weng Lo, Xu Yang, and Yapeng Wang, "An xception convolutional neural network for malware classification with transfer learning," in *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, IEEE, Ed., Jun. 2019. DOI: 10.1109/ntms.2019.8763852.

[56] Kang He, Yuefei Zhu, Yubo He, Long Liu, Bin Lu, and Wei Lin, "Detection of malicious PDF files using a two-stage machine learning algorithm," *Chinese Journal of Electronics*, vol. 29, no. 6, Institution of Engineering and Technology (IET), Eds., pp. 1165–1177, Nov. 2020. DOI: 10.1049/cje.2020.10.002.

[57] Mila Parkour, *16,800 clean and 11,960 malicious files for signature testing and research.* 2015. [Online]. Available: http://contagiodump.blogspot.com/2013/03/16800-clean-and-11960-malicious-files.html (visited on 05/11/2022).

[58] Arash Habibi Lashkari, *Evasive malicious pdf (cic-evasive-pdfmal2022) data set*, 2021. [Online]. Available: http://ahlashkari.com/Datasets-Evasive-PDFMalware.asp (visited on 05/11/2022).

[59] Ruhr University Bochum - Chair for Network and Data Security, *Pdf-attacker*, 2020. [Online]. Available: https://github.com/RUB-NDS/pdf-attacker.

[60] Ah Kang, Young-Seob Jeong, Se Kim, and Jiyoung Woo, "Malicious PDF detection model against adversarial attack built from benign PDF containing JavaScript," *Applied Sciences*, vol. 9, no. 22, p. 4764, Nov. 2019. DOI: 10.3390/app9224764.