

Technische Hochschule Ingolstadt  
Fakultät Informatik  
Bachelor Künstliche Intelligenz

# **KI-basiert versus klassisch - File Carving in der digitalen Fahrzeugforensik**

## **Bachelorarbeit**

**Vor- und Zuname**

Lea Achter

**ausgegeben am**

21. November 2023

**abgegeben am**

12. Februar 2024

**Erstprüfer**

Prof. Dr.-Ing. Hans-Joachim Hof

**Zweitprüfer**

Prof. Dr. Michael Jarschel

## **Abstract**

Durch die voranschreitende Entwicklung der Technologie im Bereich der Automobilindustrie können immer mehr digitale Spuren in Fahrzeugen festgestellt werden. Das führt zu einem immer wichtiger werdenden Gebiet, der digitalen Fahrzeugforensik. Dieses beschäftigt sich mit dem Auslesen der Fahrzeugspeicher, die meist proprietäre Dateitypen der Automobilhersteller enthalten. Somit ist der Aufbau dieser Dateien meist unbekannt, was einen Unterschied zur digitalen Forensik darstellt. Hier können durch sogenannte File Carver Dateitypen, anhand bekannter Byte Sequenzen, wie Header oder Footer, erkannt werden. Unbekannte proprietäre Dateien, wie die der Automobilindustrie können somit meist nicht gefunden werden.

Das Ziel dieser Arbeit ist es zu untersuchen, in wieweit klassische File Carver die spezifischen Dateitypen der Automobilbranche erkennen, und ob KI-basierte Ansätze hier möglicherweise einen Vorteil bieten können. Hierzu wird ein synthetischer Datensatz erstellt, um eine Basis mit relevanten Dateitypen zu schaffen.

Die Tests der Softwares auf dem erstellten Datensatz zeigen, dass sich der File Carver Autopsy am Besten für eine Untersuchung in der digitalen Fahrzeugforensik eignet. Jedoch lassen die erzielten Ergebnisse der KI-basierten Methoden auf ein deutliches Entwicklungspotential schließen.

## **Danksagung**

An dieser Stelle möchte ich mich bei denjenigen bedanken, die mich während der Anfertigung dieser Bachelorarbeit unterstützt haben. Zunächst möchte ich mich bei Herrn Prof. Dr.-Ing. Hans-Joachim Hof, der meine Arbeit begutachtet hat, für das hilfreiche Feedback und die Korrektur meiner Arbeit bedanken. Ebenfalls möchte ich mich bei meinem Betreuer Dr. Kevin Gomez Buquerin für die fachliche Expertise, und die schnelle und freundliche Beantwortung meiner Fragen bedanken.

# Inhaltsverzeichnis

<b>Abkürzungsverzeichnis</b>	<b>ix</b>
<b>1. Einleitung</b>	<b>1</b>
<b>2. Grundlagen</b>	<b>3</b>
2.1. Digitale Fahrzeugforensik . . . . .	3
2.2. File Carver . . . . .	3
2.2.1. Klassische File Carver . . . . .	4
2.2.1.1. Foremost . . . . .	5
2.2.1.2. Scalpel . . . . .	6
2.2.1.3. Forensik Toolkit Imager . . . . .	7
2.2.1.4. Autopsy . . . . .	8
2.2.1.5. Bulk_extractor . . . . .	8
2.2.2. KI-basierte File Carver . . . . .	9
2.2.2.1. Support Vector Machine . . . . .	9
2.2.2.2. CarveML . . . . .	11
2.2.2.3. Scedan . . . . .	14
2.3. Dateitypen . . . . .	15
2.3.1. Comma Separated Values (CSV) . . . . .	15
2.3.2. Excel Binary File Format (XLS) . . . . .	16
2.3.3. GPS Exchange Format (GPX) . . . . .	16
2.3.4. Crash Data Retrieval (CDR) . . . . .	17
2.3.5. CAN Database Format (DBC) . . . . .	18
<b>3. Implementierung</b>	<b>19</b>
3.1. Synthetischer Datensatz . . . . .	19
3.1.1. Vorbereitung . . . . .	19
3.1.2. Erstellung des Datensatzes . . . . .	20
3.1.3. Ausgleichen der Dateien . . . . .	21
3.1.4. Erstellung der SquashFS Datei . . . . .	23

3.2. Docker . . . . .	24
3.2.1. Aufbau von Docker und Vergleich zur Virtual Machine . . . . .	24
3.2.2. Dockerfile . . . . .	25
3.3. KI File Carver . . . . .	26
<b>4. Testdesign und -durchführung</b>	<b>29</b>
4.1. Evaluationsmetriken und Akzeptanzkriterien . . . . .	29
4.2. Erster Ansatz . . . . .	32
4.3. Anpassung der Software . . . . .	34
<b>5. Evaluation und Diskussion</b>	<b>37</b>
5.1. Evaluation der File Carver . . . . .	37
5.2. Evaluation der KI-basierten Ansätze . . . . .	39
5.3. Diskussion der Ergebnisse . . . . .	42
<b>6. Zusammenfassung und zukünftige Arbeit</b>	<b>44</b>
<b>A. Anhang</b>	<b>i</b>
<b>Appendices</b>	<b>i</b>
<b>Literatur</b>	<b>vii</b>

# Abbildungsverzeichnis

2.1. Skizzenhafter Aufbau einer Datei mit Header und Footer [5, S. 214] . . . . .	4
2.2. Arbeitsablauf von Foremost 0.69 [9] . . . . .	6
2.3. Arbeitsablauf von Scalpel [9] . . . . .	7
2.4. Lineare Hard-Margin SVM [15, S. 46] . . . . .	10
2.5. Byte Frequenzen von 0 - 256 für jeden Dateityp in Prozent . . . . .	12
2.6. Boxplot der Shannon Entropie für die verschiedenen Dateitypen . . . . .	13
2.7. Dateitypen, die mit Scedan erkannt werden können [18] . . . . .	14
2.8. Darstellung der ersten zwei Zeilen einer CSV Datei in Text und Hexadezimal . . .	15
2.9. Darstellung des Header einer XLS Datei in Text und Hexadezimal . . . . .	16
2.10. Darstellung des Header einer GPX Datei in Text und Hexadezimal . . . . .	17
2.11. Darstellung des Anfangs einer CDR Datei in Text und Hexadezimal . . . . .	17
2.12. Darstellung des Anfangs einer DBC Datei in Text und Hexadezimal . . . . .	18
3.1. Darstellung der ersten zwei DBC Dateien, gespeichert in einem Dataframe . . . .	22
3.2. Vergleich einer Virtuellen Maschine mit Docker [36] . . . . .	25
3.3. Struktur der Ordner zum Training von Scedan . . . . .	28
4.1. Confusionsmatrix einer binären Klassifikation [39] . . . . .	31
5.1. Confusionsmatrix von Autopsy . . . . .	38
5.2. Scatterplot der verschiedenen Dateitypen . . . . .	40
5.3. Confusionsmatrix von CarveML . . . . .	40
5.4. Confusionsmatrix von Scedan . . . . .	41
A.1. Arbeitsablauf von Bulk_extractor [13] . . . . .	i

# Tabellenverzeichnis

2.1. Accuracy der Modelle aus CarveML [16] . . . . .	13
3.1. Vergleich der Anzahl der Dateitypen im Originalen Datensatz mit dem im synthetisch erstellen Datensatz . . . . .	23
3.2. Accuracy der Modelle mit CarveML trainiert auf 1% der Daten . . . . .	27
4.1. Ergebnisse des ersten Ansatzes der File Carver . . . . .	34
5.1. Vergleich der Metriken der klassischen File Carver . . . . .	37
5.2. Vergleich der Akzeptanzkriterien der File Carver . . . . .	39
5.3. Vergleich der Akzeptanzkriterien der KI-basierten File Carver . . . . .	41
5.4. Vergleich der Metriken von CarveML und Scedan . . . . .	42

# Codeverzeichnis

3.1. Schreiben von Shelf Objekt in eine Text Datei . . . . .	28
A.1. Dockerfile für File Carver . . . . .	ii
A.2. Dockerfile für Scedan . . . . .	v



# Abkürzungsverzeichnis

<b>BGC</b>	Bifragmentet Gap Carving
<b>CAN</b>	Controller Area Network
<b>CBF</b>	Compound File
<b>CDR</b>	Bosch Crash Data Retrieval
<b>CSV</b>	Comma Separated Values
<b>DB</b>	Decision Boundary
<b>DBC</b>	CAN Database Format
<b>ECU</b>	Electronic Control Units
<b>EDR</b>	Event Data Recorder
<b>FC</b>	File Carver
<b>FF</b>	Fahrzeugforensik
<b>FN</b>	False Negative
<b>FP</b>	False Positive
<b>FTK</b>	Forensik Tool Kit
<b>GPS</b>	Global Positioning System
<b>GPX</b>	GPS Exchange Format
<b>KI</b>	Künstliche Intelligenz
<b>LDA</b>	Linear Discriminant Analysis
<b>MCC</b>	Matthews Correlations Coefficient
<b>MIME</b>	Multipurpose Internet Mail Extensions
<b>ML</b>	Machine Learning
<b>NLP</b>	Natural Language Processing
<b>OLE</b>	Object Linking and Embedding
<b>Sceadan</b>	Systematic Classification Engine for Advanced Data ANalysis
<b>SVM</b>	Support Vector Machine
<b>TN</b>	True Negative
<b>TP</b>	True Positive
<b>TSK</b>	The Sleuth Kit
<b>VM</b>	Virtual Machine
<b>WSL</b>	Windows-Subsystem für Linux

**XLS**    Excel Binary File Format  
**XML**    Extensible Markup Language

# 1. Einleitung

Schon in 2022 stellte das Deutsche Forschungszentrum für künstliche Intelligenz in [1] ein neues Projekt vor, Carve-DL. Das Ziel war es, mit Methoden aus der Künstliche Intelligenz (KI), dem sogenannten Deep Learning, versteckte oder gelöschte Dateien auf Speichermedien wiederherzustellen. Dieser Prozess nennt sich *Carving*. In der digitalen Forensik gibt es bereits eine Vielzahl von Software, die auch *File Carver* genannt werden. Jedoch bringen diese Nachteile mit sich. Beispielsweise ist es schwer, Dateifragmente in unstrukturierten Teilen des Datenspeichers zu finden. Auch korrupte Dateien können schwer erkannt werden. Zudem sammeln sich immer mehr Daten an, sodass in Ermittlungen teilweise enorme Datenmengen untersucht werden müssen. Hier sind die klassischen File Carver meist zu langsam. Auch in modernen Fahrzeugen werden immer mehr Daten gespeichert, was zu einem immer wichtiger werdenden Gebiet in der digitalen Forensik führt, der digitalen Fahrzeugforensik. Bei einer forensischen Untersuchung des Tesla Autopiloten konnte von Gomez Buquerin und Hof [2] ein SquashFS Dateiensystem gefunden werden. Jedoch war es bei fast sieben Prozent der Dateien nicht möglich einen Dateityp zu bestimmen. Es wurde vermutet, dass es sich um beschädigte Dateien handelt. Da klassische File Carver mit bekannten Byte Sequenzen, zur Erkennung von Start und Ende einer Datei, arbeiten, könnte es von Vorteil sein, sich bei unbekannten Dateitypen oder beschädigten Dateien, die Struktur dieser zur Erkennung zu verwenden. Die Idee Dateifragmente anhand ihrer Eigenschaften, wie der Byte Verteilung oder der Entropie, zu untersuchen ist schon länger im Einsatz. Das sogenannte Smart Carving versucht, anhand dieser Eigenschaften stark fragmentierte Dateien zu rekonstruieren [3].

In dieser Bachelorarbeit wird untersucht, in wieweit klassische File Carver die spezifischen Dateitypen der Automobilbranche erkennen, und ob KI-basierte Ansätze hier möglicherweise einen Vorteil bieten können. Hierfür wird ein Datensatz aus nicht fragmentierten Dateien erstellt, um auf diesem die File Carver zu testen. Nach der Einleitung werden zunächst die Grundlagen in Kapitel 2 vorgestellt. Es wird dabei auf die digitale Fahrzeugforensik eingegangen, und die verwendeten File Carver und Dateitypen vorgestellt. Um die Software vergleichen zu können, wird in Kapitel 3 ein synthetischer Datensatz erarbeitet. Da die ausgewählte Software unter verschiedenen Betriebssystemen läuft, werden Docker Images erstellt. Danach werden die KI-basierten File Carver trainiert. Die Software wird in Kapitel 4 getestet. In Kapitel 5, werden beide

Methoden, mit Bezug auf die Verwendung in der digitalen Fahrzeugforensik, evaluiert. Am Ende, in Kapitel 6, folgt eine Zusammenfassung dieser Bachelorarbeit und mögliche weiterführende Arbeiten werden aufgezeigt.

## 2. Grundlagen

Bevor die verschiedenen File Carver implementiert und getestet werden, soll dieses Kapitel zunächst die wichtigsten Grundlagen dieser Arbeit definieren. Hierbei wird zunächst auf die digitale Fahrzeugforensik eingegangen. Danach werden die verwendeten klassischen und KI-basierten File Carver kurz vorgestellt, und im letzten Teil wird auf die für den Datensatz ausgewählten Dateitypen eingegangen.

### 2.1. Digitale Fahrzeugforensik

Die digitale Fahrzeugforensik (FF) befasst sich mit den Daten, die in modernen Fahrzeugen vorgefunden werden können. Diese können aus vielen verschiedenen Datenquellen stammen, wie beispielsweise den Electronic Control Units (ECU) oder auch dem Event Data Recorder (EDR). Besonders für strafrechtliche Ermittlungen nach einem Unfall oder einer Straftat sind diese Daten wichtig. Hierbei ist es notwendig, diese nachvollziehbar aus den Datenquellen zu beziehen. Dafür müssen zunächst maßgebende Daten, Komponenten und Werkzeuge der Untersuchungen definiert werden. Um dies zu erreichen, führten Gomez Buquerin et al. [4] Experteninterviews durch. In diesen wurde am häufigsten das Unix `dd` Kommando erwähnt. Es dient zum bit-genauen Kopieren von Festplatten und Dateien. Ebenso wurden die Bosch Crash Data Retrieval (CDR) Produkte und Berla iVe<sup>1</sup> genannt. Aus diesen Untersuchungen resultieren Speicherauszüge der Fahrzeuge, sogenannte Memory Dumps. Sie enthalten meist unbekannte und proprietäre Dateitypen der Fahrzeughersteller. Um in diesen für Ermittlungen geeignete Daten zu finden, können verschiedene Methoden angewendet werden, unter anderem werden sogenannte File Carver (FC) verwendet.

### 2.2. File Carver

In der digitalen Fahrzeugforensik werden FC benötigt, um Fahrzeugdumps zu untersuchen. Jedoch gibt es viele verschiedene Softwareprogramme, die unterschiedliche Anwendungsgebiete haben.

---

<sup>1</sup><https://firewire-revolution.de/produkt-kategorie/alle/it-forensik-hardware/berla/>, zuletzt aufgerufen am 06.02.2024

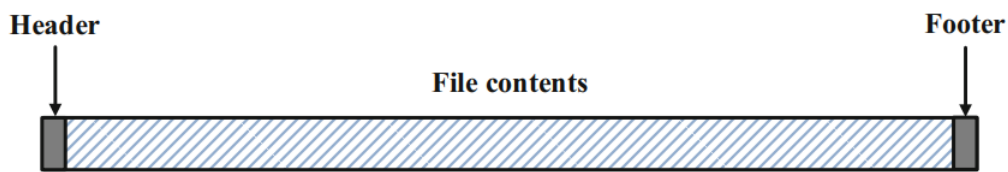


Abbildung 2.1.: Skizzenhafter Aufbau einer Datei mit Header und Footer [5, S. 214]

Der folgende Abschnitt dient dazu, die Auswahl der FC zu erläutern und den Aufbau dieser näher zu beschreiben.

### 2.2.1. Klassische File Carver

FC sind Software Tools, die sich mit der Rekonstruktion von Daten befassen ohne die Metadaten des Dateisystems zu benötigen. Sie arbeiten mit den Informationen, die in den Datenblöcken auf der Festplatte gespeichert sind. Hierbei gibt es verschiedene Herangehensweisen, wie diese Dateien rekonstruiert werden können. [5]

Ein Ansatz ist das Header/Footer Carving. Es wird auch dateistrukturbasiertes Carving genannt. Hierbei werden die verschiedenen Strukturen der Header, bzw. der Footer von Dateien verwendet. Header und Footer sind Bytesequenzen am Anfang und Ende eines Dokuments, die für einen Dateitypen spezifisch sind. In **Abbildung 2.1 wird der allgemeine Aufbau** einer Datei skizziert. Um die Datei nun zu rekonstruieren, werden die Daten nach bekannten Bytesequenzen für Dateiheader durchsucht. Sobald einer gefunden wurde, wird weiter nach dem zugehörigen Footer gesucht. Alles zwischen diesen beiden Bereichen, kann nun einer Datei zugeordnet werden. Diese Methode ist allerdings nur dann zielführend, wenn die Dateien zusammenhängend gespeichert werden. Das kann dazu führen, dass Dateien, die fragmentiert gespeichert wurden, nicht gefunden werden können. Ein weiteres Problem ist, dass öfter in den Prozess eingegriffen werden muss, da die Header- und Footersequenzen manuell bestimmt werden müssen. Ebenso muss meist bekannt sein, nach welchen Dateitypen gesucht wird. Hierfür werden Experten benötigt. Des Weiteren kann es passieren, dass die Datei keine spezifischen Header bzw. Footer Sequenzen besitzt, oder diese beschädigt wurden. Diese Dateien können nicht erkannt werden. [5][3]

Ein weiterer Ansatz ist das Bifragmented Gap Carving (BGC). Mit diesem ist es möglich Dateien, die während des Speicherprozesses in zwei Fragmente geteilt worden sind, wiederherzustellen. In realen Fällen wurden Dateien zu 97% zusammenhängend oder bifragmentiert gefunden. Im BGC werden, wie im vorherigen Ansatz auch, zunächst die Header und Footer der Datei gesucht. Danach wird die Größe der Lücke zwischen den beiden Fragmenten ausgewählt und alle möglichen Kombinationen überprüft. Falls das zu keinem Ergebnis führt, wird eine größere Lücke

gewählt und der Prozess wird wiederholt. In dieser Arbeit werden jedoch zur Vereinfachung nur zusammenhängende Dateien betrachtet. [5]

Es gibt viele verschiedene Open Source FC. In [6][7][8] werden die Bekanntesten verglichen, die Besten sollen in dieser Arbeit betrachtet werden. Zusätzlich dazu wurde noch eine Internetrecherche durchgeführt, um die FC zu finden, die am häufigsten empfohlen wurden. In [7] wurden die FC mit Multimediadateien wie MP4 oder MOV Dateien getestet. Hierbei ergab sich, dass *Autopsy*<sup>2</sup>, *PhotoRec*<sup>3</sup> und *Defraser*<sup>4</sup> gefolgt von *Foremost*<sup>5</sup>, die besten Ergebnisse erzielt haben. Da Defraser jedoch ein FC ist, der nur benutzt wird um Multimediadateien zu carven, wurde dieser nicht weiter betrachtet. Laurenson [6] und Courrejou [8] testeten die FC an Datensätzen, die eine größere Auswahl verschiedener Dateitypen enthalten. Beide kamen zu dem Ergebnis, dass PhotoRec die besten Ergebnisse auf den Datensätzen liefert. Gefolgt von Foremost und *Scalpel*<sup>6</sup>. Ebenfalls konnte das *Forensik Tool Kit (FTK)*<sup>7</sup> gute Ergebnisse erzielen. Eine Internetsuche nach empfohlenen FC ergab, dass die Beliebtesten Foremost, Scalpel, *Bulk\_extractor*<sup>8</sup> und testdisc (PhotoRec) sind. Das für den Datensatz verwendete SquashFS-Dateisystem kann unter Windows nicht als Laufwerk gemounted werden, weshalb PhotoRec nicht verwendet werden kann. Jedoch ist in Autopsy ein PhotoRec Modul vorhanden, das für die Erkennung genutzt wurde. Somit ergeben sich für diese Arbeit die nachfolgend vorgestellten FC.

#### 2.2.1.1. Foremost

Foremost<sup>5</sup> ist ein FC, der von Agenten des United States Air Force Office of Special Investigations 2001 entwickelt wurde. Die Software ist kostenlos und unter unix-basierten Betriebssystemen verfügbar. Sie arbeitet mit dem Header/Footer Carving. Somit kann Foremost Dateien wiederherstellen, deren Header, Footer und maximale Dateigröße in der *foremost.conf* Datei vordefiniert wurden. Für einen Teil dieser Dateitypen gibt es eine vordefinierte Funktion. Es kann jedoch der Kommentar vor dem gesuchten Dateityp auch in der Konfigurationsdatei gelöscht werden. Ein Vorteil dieses FC ist, dass er individuell an den Use Case angepasst werden kann. Die Software durchsucht zunächst die Image Datei oder das Laufwerk nach den definierten Headern, dabei können Dateitypen, nach denen nicht gesucht wird, mit „#“ auskommentiert werden. Wenn einer dieser Header gefunden wurde, wird weiter nach dem zugehörigem Footer gesucht. [5]

---

<sup>2</sup><https://github.com/sleuthkit/autopsy>, zuletzt aufgerufen am 25.01.2024

<sup>3</sup><https://www.cgsecurity.org/wiki/PhotoRec>, zuletzt aufgerufen am 06.02.2024

<sup>4</sup><https://www.forensicinstitute.nl/products-and-services/forensic-products/defraser>, zuletzt aufgerufen am 06.02.2024

<sup>5</sup><https://foremost.sourceforge.net/>, zuletzt aufgerufen am 26.01.2024

<sup>6</sup><https://github.com/sleuthkit/scalpel>, zuletzt aufgerufen am 26.01.2024

<sup>7</sup><https://www.exterro.com/digital-forensics-software/ftk-imager>, zuletzt aufgerufen am 25.01.2024

<sup>8</sup>[https://github.com/simsong/bulk\\_extractor](https://github.com/simsong/bulk_extractor), zuletzt aufgerufen am 25.01.2024

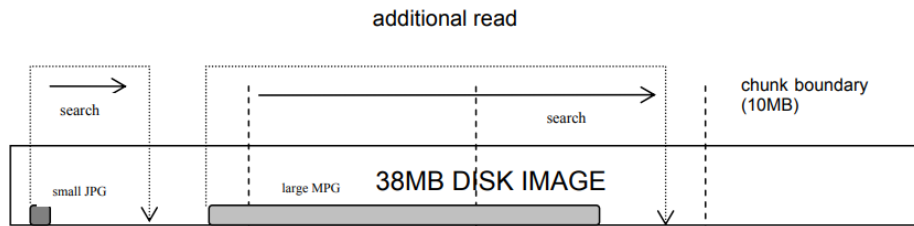


Abbildung 2.2.: Arbeitsablauf von Foremost 0.69 [9]

In Abbildung 2.2 ist ein **beispielhafter** Arbeitsablauf von Foremost 0.69 dargestellt. Dabei wird der zu durchsuchende Part zunächst in Chunks eingeteilt. Danach verläuft die Suche in jedem Chunk hintereinander. Wenn ein Header gefunden wird, wird der Bereich, der zugehörig in der Konfigurationsdatei als maximale Dateigröße angegeben ist, nach einem Footer durchsucht. Falls die Datei über mehrere vordefinierte Chunks geht, muss ein Buffer aufgebaut werden, der die Informationen zwischenspeichert. [9]

#### 2.2.1.2. Scalpel

Scalpel wurde 2005 erstmals von Golden und Vassil [9] vorgestellt. Die Open Source FC Software wurde bis 2013 regelmäßig geupdated, besitzt aber keine offizielle veröffentlichte Version. Der Source Code ist auf dem GitHub von Sleuthkit<sup>9</sup> verfügbar. Scalpel basiert ursprünglich auf Foremost 0.69, was vorallem durch die Konfigurationsdatei *scalpel.conf* auffällt. Diese ist analog zu der, die von Foremost verwendet wird. Allerdings müssen alle gesuchten Dateitypen erst durch das Entfernen des „#“ definiert werden. Es sind keine vordefinierten Funktionen enthalten. Das Entfernen aller Kommentare kann schnell zu vielen falschen Ergebnissen führen.

Die Software wurde entwickelt, um die negativen Punkte, die Foremost mit sich bringt, zu verbessern. Dabei sollte die Software wenig Ressourcen benötigen, schnell Ergebnisse liefern, ohne dabei an Genauigkeit zu verlieren, und auf verteilten forensischen Systemen laufen. [9]

Ein beispielhafter Arbeitsprozess der Software ist in Abbildung 2.3 dargestellt. Zunächst liest Scalpel die Konfigurationsdatei ein. Diese enthält die Header und Footer jeder Datei und kann auch die maximale Dateigröße des Dateitypen definieren. Danach geht Scalpel zweimal über die Daten. Beim ersten Mal wird das zu durchsuchende Speicherabbild in einzelnen Chunks eingeteilt. Die Größe kann dabei vom Benutzer angepasst werden und ist ursprünglich 10 MB. Jeder dieser Chunks wird erst nach Header- und dann nach passenden Footersequenzen durchsucht. Diese werden mit der zugehörigen Position in einer Datenbank gespeichert. Der erste Durchlauf ergibt

<sup>9</sup><https://github.com/sleuthkit/scalpel>, zuletzt aufgerufen am 26.01.2024



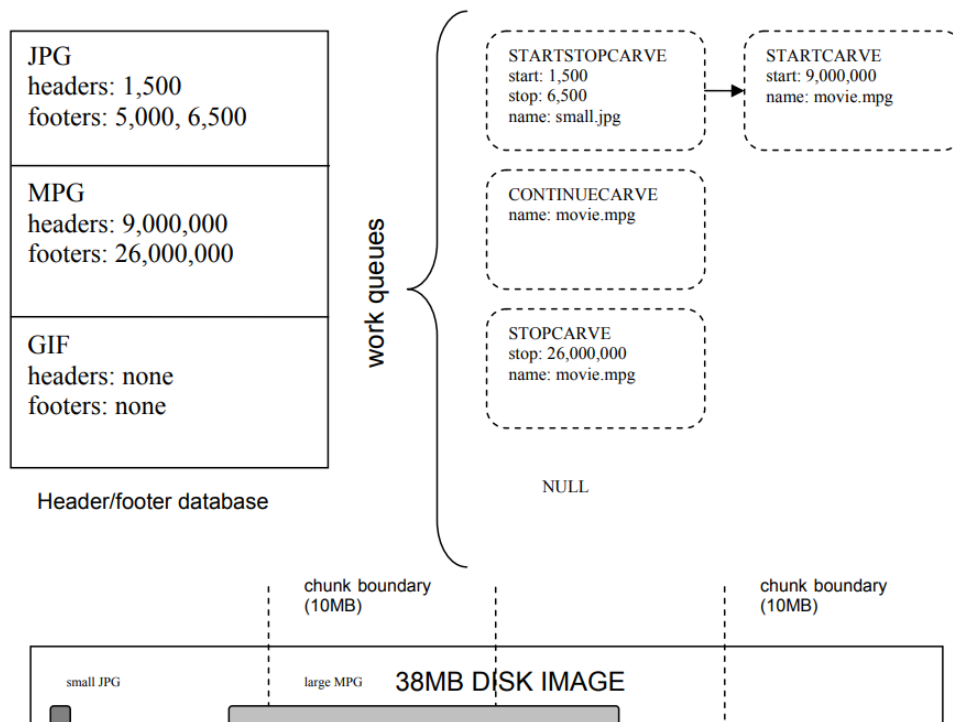


Abbildung 2.3.: Arbeitsablauf von Scalpel [9]

somit alle gefundenen Header und Footer mit ihren Positionen. Im zweiten Durchlauf werden nun verschiedene *work queues* erstellt, je nachdem ob sich die Datei komplett in den eingeteilten Chunks befindet oder sich über mehrere Chunks erstreckt. Diese work queues kopieren die gefundenen Dateidaten in eine neu erstellte Datei. [9]

### 2.2.1.3. Forensik Toolkit Imager

Der FTK Imager ist eine kostenlose Software der AccessData Group<sup>10</sup>, die dafür verwendet werden kann, einen schnellen Überblick über die Daten zu bekommen. Sie ist unter Windows verfügbar. Die kostenpflichtige Software FTK kann auch unter Linuxsystemen verwendet werden. Mit FTK Imager ist es möglich perfekte Kopien, auch forensisches Image genannt, von Daten zu erstellen. Des Weiteren können Images in die Software eingebunden und die Inhalte somit dargestellt werden. Um den Inhalt darzustellen wird der Windows Explorer verwendet. Ebenfalls können auch bereits gelöschte Dateien, die noch nicht überschrieben wurden, dargestellt werden. [10]

<sup>10</sup><https://www.exterro.com/digital-forensics-software/ftk-imager>, zuletzt aufgerufen am 25.01.2024

#### 2.2.1.4. Autopsy

Autopsy<sup>11</sup> ist eine Open Source Software, die eine grafische Oberfläche für The Sleuth Kit (TSK) und weitere forensische Softwares bietet. Sie wird regelmäßig geupdatet und ist unter Linux, Windows und MacOSX verfügbar. Jedoch ist sie nur unter Windows komplett getestet. Um einen Datenträger zu untersuchen, wird zunächst ein Case erstellt. Zu diesem kann dann ein Speicherabbild hinzugefügt werden. Hierbei werden mit TSK die Dateien in dem Dateiensystem nummeriert und in einer Datenbank gespeichert. Die verschiedenen Module können nun verwendet werden, um entweder einzelne Dateien oder den Datenträger zu untersuchen. Die Ergebnisse werden im *Ergebnis-Tree* dargestellt. Ebenfalls ist die Möglichkeit gegeben eigene Module anzubinden. Dies erfolgt durch das `org.sleuthkit.autopsy.ingest` Package. Am Ende kann ein Report ausgegeben werden. [11][12]

#### 2.2.1.5. Bulk\_extractor

Die Software `bulk_extractor` wurde von Simson Garfinkel entwickelt und 2013 veröffentlicht [13]. Der Code ist Open Source und auf GitHub<sup>12</sup> verfügbar. Die Software wird regelmäßig geupdated und seit 2021 erfolgen regelmäßige Neuveröffentlichungen. Sie ist auf Linux, Windows und MacOS verfügbar.

Der Zweck von `Bulk_extractor` ist es, Massendaten (engl. *bulk data*) schneller, bei der Erstuntersuchung der Daten, einzuordnen und somit bestimmte Datenträger priorisieren zu können. Hierbei handelt es sich um einen komplementären Prozess zum dateiestrukturbasierten Ansatz. Die Daten werden, ohne die Metadaten des Dateiensystems zu verwenden, untersucht. So können Informationen gefunden werden, die nicht in den Dateien liegen. In der Massendatenanalyse (engl. *bulk data analysis*) können auch defekte oder überschriebene Dateien gefunden und ausgelesen werden. Ein weiterer Vorteil ist, dass das Verarbeiten der Daten, im Gegensatz zu beispielsweise FTK, parallelisiert werden und somit erheblich schneller sein kann. [13]

`Bulk_extractor` arbeitet mit verschiedenen Scannern, die in **Abbildung A.1** dargestellt werden. Hierbei wird in zwei verschiedene Scannerarten unterschieden. Diese sind die Basic Scanner und die Rekursiven Scanner. Die Basic Scanner, wie der *email scanner*, analysieren die Daten einmal und schreiben die Informationen in eine Datei. Mit den Rekursiven Scannern, wie dem *zip scanner*, können die Daten erst decodiert und dann nochmal analysiert werden. Durch die API ist es auch möglich eigene Scanner für proprietäre Dateitypen zu verwenden. [13]

---

<sup>11</sup><https://github.com/sleuthkit/autopsy>, zuletzt aufgerufen am 25.01.2024

<sup>12</sup>[https://github.com/simsong/bulk\\_extractor](https://github.com/simsong/bulk_extractor), zuletzt aufgerufen am 25.01.2024

### 2.2.2. KI-basierte File Carver

Nachdem im vorherigen Kapitel die klassischen FC herausgearbeitet wurden, sollen in diesem Kapitel die KI-basierten FC vorgestellt werden. Da FC die Dateitypen mithilfe der Header und Footer Sequenzen bestimmen, kann das bei proprietären Dateitypen, die meist in der Automobilbranche verwendet werden, zu Problemen führen, da diese Sequenzen nicht vorhanden oder bekannt sind. Einen Vorteil könnte hier die KI bieten. In dieser Arbeit wird ausschließlich auf Machine Learning (ML) Ansätze eingegangen. Das ML ist ein Teilgebiet der KI, es befasst sich damit, eine Zielvariable  $y$ , auch Klasse genannt, mithilfe einer Menge an Variablen  $X$  vorherzusagen. In diesem Fall ist die Zielvariable der Dokumententyp, der vorhergesagt werden soll. Da die Modelle anhand der Struktur der Bytes eines Dokuments lernen und weniger Bezug auf beispielsweise Header oder Footer nehmen, kann in diesem Kontext möglicherweise eine Verbesserung der derzeitigen FC erreicht werden.

In dem Bereich des ML befasst sich das Natural Language Processing (NLP) mit der Verarbeitung von natürlicher Sprache, also beispielsweise mit der Klassifikation von Texten. Diese Erkenntnisse können auch auf die Klassifikation von Dateitypen übertragen werden. In [14] werden verschiedene Ansätze zur Dateityperkennung miteinander verglichen. Es zeigt sich, dass Support Vector Machines bessere Ergebnisse erzielen, als Neuronale Netze. Die in dieser Arbeit verwendeten KI-basierten FC nutzen alle die Support Vector Machine (SVM) um die Dateitypen zu erkennen.

Die folgenden KI-basierten FC wurden ausgewählt, da **der** Code Open Source ist und diese somit, ohne größere Veränderungen, gut vergleichbar zu der ausgewählten klassischen Software ist.

#### 2.2.2.1. Support Vector Machine

Zunächst soll die SVM definiert werden. Eine SVM ist ein ML Modell, welches eine binäre Klassifikation durchführt, das heißt einen Datensatz in zwei Klassen einteilt. Hierbei wird mithilfe einer Hyperebene, der Decision Boundary (DB), versucht, die Klassen linear zu trennen. In Abbildung 2.4 ist beispielhaft eine SVM in  $\mathbb{R}^2$  dargestellt. Um die beste Decision Boundary zu finden, wird versucht diese so zu legen, dass zu beiden Klassen der größtmögliche Abstand gegeben ist. Dieser Abstand wird *Margin* genannt. Somit ergibt sich ein Optimierungsproblem, bei dem eine Hyperebene gefunden werden muss, die die Margin maximiert. Punkte die auf der Margin liegen heißen Support Vektoren, von diesen bekommt die SVM ihren Namen. Diese Support Vektoren sind die einzigen Punkte, die für die Lösung des Optimierungsproblems relevant sind. Diese SVM wird auch *Lineare Hard-Margin SVM* genannt, da die Margin nicht verletzt werden darf.

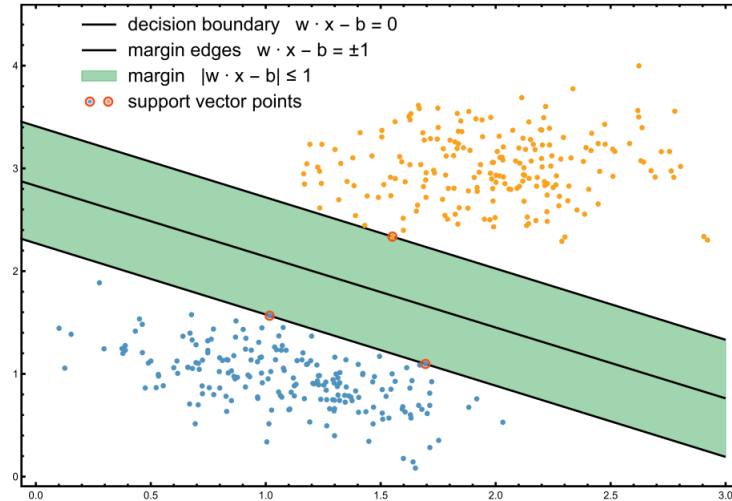


Abbildung 2.4.: Lineare Hard-Margin SVM [15, S. 46]

Bei Problemen, die nicht linear trennbar sind, ist es nicht möglich, eine Lösung zu finden, die die Margin nicht verletzt. Diese Modelle werden als *Soft-Margin SVM* bezeichnet. Die Verletzung sollte hierbei so gering wie möglich sein. Somit entstehen zwei Bedingungen: 1. Die Margin soll maximal sein, 2. Die Verletzungen der Margin sollen minimal sein. Eine sogenannte Schlupfvariable  $\zeta$  macht es möglich, die Verletzung der Margin zu regeln. Sie stellt den Abstand zur DB dar. Die Variable  $C$  beschreibt wie schwer die Verletzung bestraft werden soll. Das heißt wenn  $C$  groß ist, wird  $\zeta$  und somit die zugelassenen Verletzungen der DB kleiner. Es ergibt sich folgende Gleichung, die den Abstand zur Hyperebene angibt:

$$\frac{1}{2} \|v\|^2 + C \sum_{i=1}^n \zeta_i$$

Die Schlupfvariable kann auch durch den Hinge-Loss abgebildet werden, da sie sich wie diese Funktion verhält. Sie ist größer als 1 falls der Datenpunkt falsch klassifiziert ist. Falls die Punkte in der Margin liegen, sind die Werte zwischen 0 und 1. Bei richtiger Klassifikation entspricht der Wert 0. Somit ergibt sich eine Gleichung, die mithilfe des Gradientenabstiegsverfahrens minimiert werden kann:

$$\frac{1}{2} \|v\|^2 + C \sum_{i=1}^n L(y^{(i)} \cdot (v^t x^{(i)} + d))$$

Eine weitere gängige Methode um für nicht-linear trennbare Daten eine Lösung zu finden ist, diese in eine höhere Dimension abzubilden in der sie durch eine Hyperebene trennbar sind, um dann das Skalarprodukt der Daten zu berechnen. Das wird durch sogenannte Kerne erreicht. Ein weitverbreiteter Kern ist der lineare Kern  $\kappa(x_1, x_2) = x_1^t x_2$ . [15]

Die SVM ist ein White Box Model, das heißt jede Vorhersage dieses Modells kann nachvollzogen werden, da die SVM im Grunde ein Optimierungsproblem darstellt. Das ist vor allem für forensische Untersuchungen von Vorteil, da die Bestimmung des Dateitypen erklärbar ist.

#### **2.2.2.2. CarveML**

Der erste ausgewählte KI-basierte File Carver ist CarveML. Dieser wurde von Andrew Duffy entwickelt und 2014 veröffentlicht. In [16] zeigt Duffy die gewählten ML Ansätze und den Aufbau des Datensatzes auf. Der Code ist Open Source und auf Github<sup>13</sup> einsehbar. Bei diesem Tool handelt es sich um verschiedene Modelle, die auf dem govdocs<sup>14</sup> Datensatz von Digital Corpora trainiert wurden. Die Modelle wurden hierbei auf einem zufällig ausgewählten Teil des Datensatzes auf den ganzen Dokumenten trainiert und auf einzelnen Fragmenten der Dateien getestet.

Für das Training wurde für jede Datei ein Feature-Vektor berechnet, der die Datei mathematisch darstellt. Dieser Vektor hat eine Dimension von 257. Er setzt sich aus dem Byte Histogramm und der Shannon Entropie  $H$  zusammen. Das Byte Histogramm ist die Verteilung der 256 verschiedenen Werte die ein Byte annimmt, in einer Datei. Das gemittelte Histogramm für jeden in dieser Arbeit behandelten Dateitypen ist in Abbildung 2.5 dargestellt. Hierbei ist deutlich zu erkennen, wie unterschiedlich die Verteilung der Bytes ist. [16]

---

<sup>13</sup><https://github.com/a10y/carveml>, zuletzt aufgerufen am 20.12.2023

<sup>14</sup><https://digitalcorpora.org/corpora/file-corpora/files/>, zuletzt aufgerufen am 06.02.2024

### Byte Frequency Tables

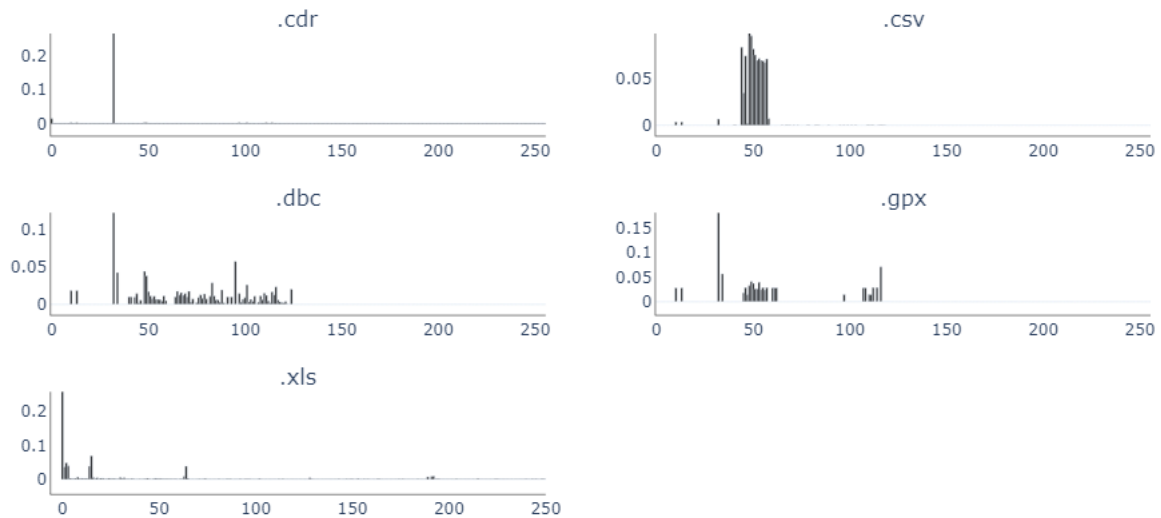


Abbildung 2.5.: Byte Frequenzen von 0 - 256 für jeden Dateityp in Prozent

Der zweite Wert, die Shannon Entropy  $H$ , lässt sich durch

$$H = - \sum_{i=0}^{255} p(x = i) \log_2 p(x = i)$$

berechnen. Hier bei ist  $p(x = i)$  die Wahrscheinlichkeit für jedes Byte ein Wert  $i$  anzunehmen. Die Shannon Entropie gibt Auskunft über die Unregelmäßigkeiten in der Verteilung der verschiedenen Bytes. Der Wert  $H$  liegt zwischen 0 und 8. Bei  $H = 0$  würde es bedeuten, dass das ganze Dokument nur aus einem zufälligen Byte besteht. Der größte Wert der angenommen werden kann ist  $H = \log_2(|X|) = \log_2(256) = 8$ , was einer Gleichverteilung der Bytes entsprechen würde. [17] Duffy beschreibt diese Entropie als Maß dafür, wie viel Informationen in den Bytes stehen. Dabei haben stark komprimierte Dateitypen eine Shannon Entropie von fast 8 und weniger komprimierte einen niedrigeren Wert. Die Shannon Entropie ist in Abbildung 2.6 dargestellt. Dabei wird auch der jeweilige Median und die Streuung der Werte aufgezeigt. [16]

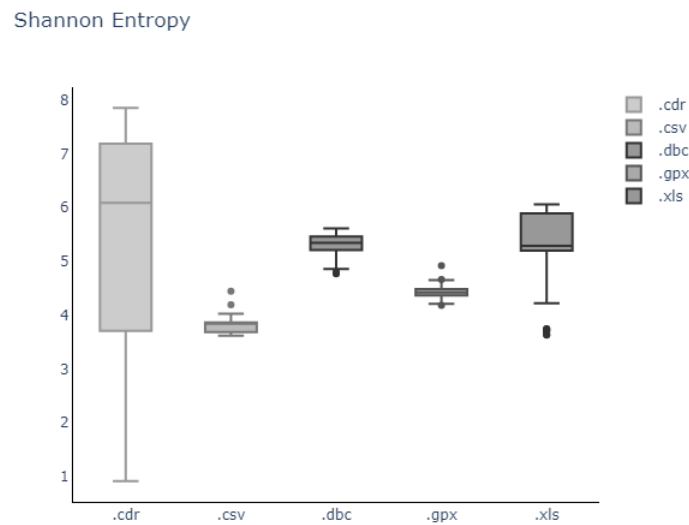


Abbildung 2.6.: Boxplot der Shannon Entropie für die verschiedenen Dateitypen

Duffy wählte drei Modelle zum Vergleich aus. Zum einen zwei traditionelle Klassifikationsalgorithmen, SVM und Linear Discriminant Analysis (LDA), und ein Algorithmus der meist als einfacher erster Ansatz in dem NLP verwendet wird, der Naive Bayes. Die LDA ist ein Algorithmus, der mithilfe der Linearkombinationen der Dateneigenschaften bestimmt, ob diese gruppiert werden können. Der Naive Bayes Algorithmus geht davon aus, dass Dateneigenschaften völlig unabhängig voneinander sind. Dies könnte zu den schlechten Ergebnissen von 47.90% führen. In 2.1 werden die einzelnen Genauigkeiten der Modelle angegeben. Es geht hervor, dass die SVM mit 75,03% das beste Modell ist. Diese SVM verwendet einen linearen Kern und die L1 Regularisierung mit  $C = 100$ . Die Regularisierung wird hier verwendet, um kleineren Werten wie den Byte Frequenzen mehr Bedeutung zukommen zu lassen, und damit dem möglichen Overfitting des Modells entgegenzuwirken. [16]

	Accuracy
SVM	75.03%
Multinomial Naive Bayes	47.90%
LDA	69.01%

Tabelle 2.1.: Accuracy der Modelle aus CarveML [16]

### 2.2.2.3. Scedan

Der zweite KI-basierte Ansatz ist Systematic Classification Engine for Advanced Data ANalysis (Scedan)<sup>15</sup>. Dies ist ein Open Source Tool zur Klassifikation von Dateitypen. Beebe et al. stellen in ihrem Paper [18] in 2013 dieses Tool vor. Es kann 30 Dateitypen und 8 Datentypen erkennen, die in Abbildung 2.7 aufgezeigt werden. Ein Teil dieser Dateien wurde wie in CarveML aus dem govdocs Datenset genommen. Weitere Dateien wurden beispielsweise von dem Projekt Gutenberg eBook Datenset und aus den internen Dateien von Windows 7 genommen.

Zum Trainieren der hier benutzten SVM wurde ebenfalls ein Feature-Vektor für jede Datei berechnet. Dieser setzt sich aus dem Unigram (1-gram) und dem Bigram (2-gram) der Bytes zusammen. Es wurden weitere Feature-Vektoren getestet, jedoch haben diese schlechtere Ergebnisse erzielt. Das n-gram ist ein Maß für die Wahrscheinlichkeit, dass  $n$  Wörter in diesem Fall Bytes in dieser bestimmten Reihenfolge in dem Dokument vorkommen. Berechnen kann man das n-gram durch:

$$P(\underline{w}) = \prod_{i=1}^{|\underline{w}|} P(w_i | w_1 \dots w_{i-1})$$

Das Unigram ist dabei die in CarveML erwähnte Byte Frequenz die auch in Abbildung 2.5 dargestellt wird. Die beste und somit in Scedan verwendete SVM besitzt einen linearen Kern und einen Wert von  $C = 256$ . Es wurde die L2-regularisierte Loss Funktion für das Training verwendet. Dieses Model wurde auf 60% der Daten trainiert und auf 40% getestet. Hierbei wurde eine Klassifikationsrate von 73,4% erreicht. Das vortrainierte Modell kann jedoch nicht mehr heruntergeladen werden und muss somit auf dem in dieser Arbeit erstellten Datensatz neu trainiert werden. [18]

TABLE I  
FILE AND DATA TYPES STUDIED

File Types				Data Types
JPG	BZIP2	WMV	HTML	TXT <sup>a</sup>
GIF	GZIP	MP3	XML	BASE64
BMP	DOC	MP4	JSON	BASE85
PNG	DOCX	AVI	CSV	URLENCODED
TIF	XLS	FLV	CSS	FAT FS DATA <sup>b</sup>
PDF	XLSX	M4A	LOG	NTFS FS DATA
PS	PPT	JAVA		EXT FS DATA
ZIP	PPTX	JS		AES256

<sup>a</sup>TXT was either .text, or .txt files containing simple ASCII text that is explicitly *not* one of the other text types listed (e.g. XML, HTML).

<sup>b</sup>“FS Data”=File System Data (e.g., \$MFT, inode tables).

Abbildung 2.7.: Dateitypen, die mit Scedan erkannt werden können [18]

<sup>15</sup><https://github.com/UTSA-cyber/scedan>, zuletzt aufgerufen am 24.01.2024



## 2.3. Dateitypen

Nachdem die verwendeten FC beschrieben wurden, soll in diesem Abschnitt die Struktur der ausgewählten Dateitypen erläutert werden. In einer Experten-Umfrage [4], die sich mit der Frage nach Dateitypen in der digitalen Fahrzeugforensik beschäftigt, wurden 24 verschiedene Typen genannt. Darunter die bekanntesten CSV und XLS Dateien, aber auch GPX, DBC und CDR Dateien wurden erwähnt. Ebenfalls wurden in [19] EDR Daten als die vielversprechendste Quelle für Informationen zur Untersuchung von Autounfällen dargestellt. Diese Daten werden als CDR Dateien ausgelesen. Somit wurden für den in dieser Arbeit synthetisch erstellten Datensatz diese fünf Dateitypen gewählt. Im Folgenden sollen diese genauer beschrieben werden.

### 2.3.1. Comma Separated Values (CSV)

Der Comma Separated Values (CSV) Dateityp ist ein viel genutzter Dateityp um Informationen aus Tabellenkalkulationsprogrammen darzustellen. Diese Dateien werden verschieden implementiert, da es keine offizielle Dokumentation gibt, sie sind jedoch immer gleich aufgebaut. Dieser Dateityp besitzt keine spezifische Header- oder Footerdatensequenz. Jeder Datenpunkt ist in einer Zeile dargestellt, dabei werden die verschiedenen Informationen mit einem Komma oder Semikolon getrennt. Das Ende dieser ist durch den Zeilenumbruch CRLF gekennzeichnet. Optional ist in der ersten Zeile ein Header, der die Namen der einzelnen Spalten enthält. Am Ende der letzten Zeile, nach dem letzten Eintrag, kann optional ein Zeilenumbruch stehen. Die Anzahl der Einträge in den Spalten pro Datenpunkt sollten immer dieselben bleiben. Das Encoding ist meist ASCII, aber andere sind auch zulässig. In Abbildung 2.8 sind die ersten beiden Zeilen einer CSV Datei in Text und Hexadezimal dargestellt. Hier lässt sich die Struktur dieses Dateitypen noch einmal eindeutig erkennen. [20]

1	,gpstime,latitude,longitude,altitude,speed,heading,hpdop,vdop,pdop																
2	0,1518263739.0,41.23910255,-8.54165405,138.8,0.051,209.00,12.4,?,?																
Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	2c	67	70	73	74	69	6d	65	2c	6c	61	74	69	74	75	64	,gpstime,latitude
00000010	65	2c	6c	6f	6e	67	69	74	75	64	65	2c	61	6c	74	69	e,longitude,altitude
00000020	74	75	64	65	2c	73	70	65	65	64	2c	68	65	61	64	69	tude,speed,heading
00000030	6e	67	2c	68	70	64	6f	70	2c	76	64	6f	70	2c	70	64	ng,hpdop,vdop,pdop
00000040	6f	70	0d	0a	30	2c	31	35	31	38	32	36	33	37	33	39	op..0,1518263739.
00000050	2e	30	2c	34	31	2e	32	33	39	31	30	32	35	35	2c	2d	.0,41.23910255,-
00000060	38	2e	35	34	31	36	35	34	30	35	2c	31	33	38	2e	38	8.54165405,138.8
00000070	2c	30	2e	30	35	31	2c	32	30	39	2e	30	30	2c	31	32	,0.051,209.00,12
00000080	2e	34	2c	3f	2c	3f	0d	0a	31	2c	31	35	31	38	32	36	.4,?,?..1,151826

Abbildung 2.8.: Darstellung der ersten zwei Zeilen einer CSV Datei in Text und Hexadezimal

### 2.3.2. Excel Binary File Format (XLS)

Das Excel Binary File Format (XLS) ist ein proprietärer Dateityp, der von Microsoft für Excel Dateien bis Microsoft Office Excel 2003 verwendet wurde. Jedoch ist die Dokumentation des Dateitypen öffentlich einsehbar. Das Dateiformat wurde durch das neuere XLSX Format abgelöst, das aus komprimierten XML Dateien besteht. Die XLS Datei ist eine Object Linking and Embedding (OLE) Compound File (CBF). Diese Datei enthält Storages, Streams und Substreams. Jede Datei muss mindestens einen Workbook Stream mit einem Sheet Substream besitzen. Dieser definiert globale Eigenschaften und Daten für das Workbook und die Tabelle. Die CBF besitzt eine Header Signatur D0 CF 11 E0 A1 B1 1A E1, die am Anfang an „docfile“ erinnert. Diese ist in Abbildung 2.9 in Text und Hexadezimal aufgezeigt. Ein 512 Byte Offset 09 08 10 00 00 06 05 00 spezifiziert die XLS Datei. Dabei stehen die Bytes für den Beginn der Datei und die Version. Die letzten zwei Bytes 05 00 geben den Anfang des globalen Streams an. [21, S. 56][22]

1																
2																

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	d0	cf	11	e0	a1	b1	1a	e1	00	00	00	00	00	00	00	00	ÐÏ.ä;±.ä.....
00000001	00	00	00	00	00	00	00	00	21	00	03	00	fe	ff	09	00	.....!...pÿ..
00000002	06	00	00	00	00	00	00	00	00	00	00	00	04	00	00	00	.....
00000003	fa	01	00	00	00	00	00	00	00	10	00	00	f9	01	00	00	ü.....ü...

Abbildung 2.9.: Darstellung des Header einer XLS Datei in Text und Hexadezimal

### 2.3.3. GPS Exchange Format (GPX)

Das GPS Exchange Format (GPX) 1.1 wurde 2004 veröffentlicht und basiert auf dem Extensible Markup Language (XML)-Format. Dieser Dateityp wird dazu benutzt Global Positioning System (GPS)-Daten zu speichern und zwischen verschiedenen Anwendungen auszutauschen. Da das XML Format klar definiert ist, hat die GPX-Datei eine eindeutige Struktur. Eine XML Datei hat keine definierte Headersequenz, an der das Dateiformat klar ersichtlich ist, jedoch sollte jede Datei mit `<?xml` gefolgt von der Version beginnen. Somit können 3c 3f 78 6d 6c 20 als Magic Bytes angesehen werden. Die erstellten GPX Dateien haben das folgende Schema. Das root Element ist der *gpxType*, hier können Metadaten und Tracks gespeichert werden. Die GPS Daten werden in *trackpoints* gespeichert. Diese enthalten die Variablen *latitude* und *longitude*. Diese Punkte können dann zu einem Tracksegment, die einen Track ergeben, zusammengefasst werden. Der Anfang einer beispielhaften GPX Datei ist in Abbildung 2.10 dargestellt. [23][24]

1	<?xml version="1.0" encoding="UTF-8"?>															
2	<gpx xmlns="http://www.topografix.com/GPX/1/1"															
3	<trk>															

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	3c	3f	78	6d	6c	20	76	65	72	73	69	6f	6e	3d	22	31	<?xml version="1
00000010	2e	30	22	20	65	6e	63	6f	64	69	6e	67	3d	22	55	54	.0" encoding="UT
00000020	46	2d	38	22	3f	3e	0d	0a	3c	67	70	78	20	78	6d	6c	F-8"?>..<gpx xml
00000030	6e	73	3d	22	68	74	74	70	3a	2f	2f	77	77	77	2e	74	ns="http://www.t
00000040	6f	70	6f	67	72	61	66	69	78	2e	63	6f	6d	2f	47	50	opografix.com/GP
00000050	58	2f	31	2f	31	22	20	78	6d	6c	6e	73	3a	78	73	69	X/1/1" xmlns:xsi

Abbildung 2.10.: Darstellung des Header einer GPX Datei in Text und Hexadezimal

#### 2.3.4. Crash Data Retrieval (CDR)

Die CDR Datei ist ein proprietärer Dateityp der Firma Bosch. Die Datei wird von dem 2018 erstmals veröffentlichten CDR Tool erzeugt. Dieses kann den in fast jedem Kraftfahrzeug enthaltenen EDR auslesen. Der EDR speichert alle technisch wichtigen Informationen eines Fahrzeugs in dem Zeitraum um einen Unfall ab. Es wird beispielsweise gespeichert, wann der Airbag ausgelöst wurde. Der Aufbau und somit Header oder Footer dieser Datei nicht öffentlich dokumentiert, da es sich um einen proprietären Dateitypen handelt. Es könnte jedoch möglich sein die Kennzeichnung der Datei ..Crash Data Retrieval Tool zu verwenden. In Abbildung 2.11 ist der Anfang einer solchen Datei in Text und Hexadezimal dargestellt. [25]

1	1FMCU9GD9HU*****															
2	FordAB10P_1															
3	GJ5T-14C028-AB															
4																
5																
6																
7																
8																
9	Crash Data Retrieval Tool 16.6															
10	Block number: 06															
11	Interface version: 03															
12	Date: 04-04-16															
13	Checksum: D800															

00000000	31	46	4d	43	55	39	47	44	39	48	55	2a	2a	2a	2a	2a	1FMCU9GD9HU*****
00000010	2a	0d	0a	46	6f	72	64	41	42	31	30	50	5f	31	20	20	*..FordAB10P_1
00000020	20	20	20	0d	0a	47	4a	35	54	2d	31	34	43	30	32	38	..GJ5T-14C028
00000030	2d	41	42	20	20	0d	0a	20	20	20	20	20	20	20	20	20	-AB ..
00000180	0a	43	72	61	73	68	20	44	61	74	61	20	52	65	74	72	.Crash Data Retr
00000190	69	65	76	61	6c	20	54	6f	6f	6c	20	31	36	2e	36	20	ieval Tool 16.6
000001a0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	

Abbildung 2.11.: Darstellung des Anfangs einer CDR Datei in Text und Hexadezimal

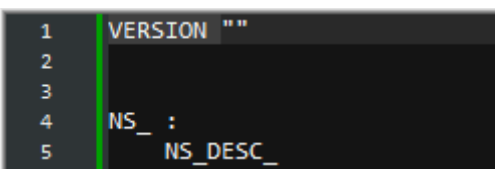
### 2.3.5. CAN Database Format (DBC)

Im Folgenden soll das CAN Database Format (DBC) näher beschrieben werden. Dieses Format ist von der Vector Informatik GmbH veröffentlicht worden, um die Kommunikation eines Controller Area Network (CAN) zu beschreiben. Es ist ein proprietärer Dateityp dieser Firma, weshalb nur wenige Informationen verfügbar sind. Zum Erstellen und Validieren dieser Dateien kann die kostenlose Software CANdb++<sup>16</sup> verwendet werden. Die DBC Datei gibt Aufschlüsse darüber, wie CAN-Bus Signale zu interpretieren sind. Dabei kann jeder CAN-Bus einzeln von einer eigenen DBC Datei beschrieben werden, wenn das Auto mehrere CAN-Busse besitzt. [26][27]

Der Aufbau der DBC Datei ist in [27] definiert. Die Datei ist eine Text Datei. Sie enthält somit keine Magic Bytes. Dennoch muss eine bestimmte Struktur eingehalten werden, um eine valide Datei zu erhalten. Der Header einer DBC Datei besteht aus der VERSION und den new\_symbols('NS\_ :'). Die Version kann auch einen leeren String enthalten. Unter 'NS\_ :' sind alle Variablen, die in der Datei benutzt werden, definiert. Die Baudrate wird in bit\_timing('BS\_:') angegeben. Diese Variable wird jedoch nicht mehr verwendet, ist für eine DBC Datei aber nötig. Die Knoten können mit 'BU\_:' definiert werden. Mit der Variable 'BO\_ ' wird eine Nachricht definiert. Diese besteht aus message\_id message\_name ':' message\_size transmitter signal. Das Signal ist durch 'SG\_ ' definiert. In Abbildung 2.12 ist der Anfang dieses Dateitypen dargestellt. Eine beispielhafte Message sieht wie folgt aus [27]:

BO\_ 100 EngineData: 8 Engine

SG\_ PetrolLevel : 24|801+ (1,0) [0|255] "1" Gateway



Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	56	45	52	53	49	4f	4e	20	22	22	0d	0a	0d	0a	0d	0a	VERSION "".....
00000010	4e	53	5f	20	3a	20	0d	0a	09	4e	53	5f	44	45	53	43	NS_ : ...NS_DESC
00000020	5f	0d	0a	09	43	4d	5f	0d	0a	09	42	41	5f	44	45	46	...CM_...BA_DEF
00000030	5f	0d	0a	09	42	41	5f	0d	0a	09	56	41	4c	5f	0d	0a	...BA_...VAL_...

Abbildung 2.12.: Darstellung des Anfangs einer DBC Datei in Text und Hexadezimal

<sup>16</sup><https://www.vector.com/int/en/products/products-a-z/software/candb/#c104686>, zuletzt aufgerufen am 20.11.2023

## 3. Implementierung

Nachdem im vorherigen Kapitel die Grundlagen dargestellt wurden, sollen in diesem Kapitel die kritischen Designentscheidungen aufgezeigt werden. Zunächst wird ein geeigneter Datensatz erstellt. Hier wird auf die Gewinnung der Daten und das Ausgleichen des daraus resultierenden unbalancierten Datensatzes eingegangen. Danach wird die Entscheidung für Docker dargelegt und die Erstellung der Dockerfile sowohl für Software, die nur auf linuxbasierten Betriebssystemen funktionieren, als auch für Scedan aufgezeigt. Als nächstes wird das Training der beiden KI-Carver dargestellt. Ziel dieses Kapitels ist es, relevante Designentscheidungen nachvollziehen zu können und somit diese Arbeit reproduzierbar zu machen.

### 3.1. Synthetischer Datensatz

Zunächst wird eine Grundlage benötigt, auf der die Software und KI-Modelle miteinander verglichen werden können. Da in der digitalen Fahrzeugforensik kein geeigneter Datensatz vorliegt, wird er synthetisch in diesem Abschnitt erstellt. Dieser Datensatz basiert auf relevanten Dateien, die natürlicherweise in Fahrzeugspeicherauszügen gefunden werden können.

#### 3.1.1. Vorbereitung

In diesem Abschnitt sollen zunächst die allgemeinen Anforderungen an den Datensatz dargestellt werden. Im Bereich der digitalen Fahrzeugforensik ist hervorzuheben, dass bei Speicherauszügen die aus forensischen Untersuchungen gewonnen wurden, meist nicht bekannt ist welche Dateien diese beeinhalteten, da in der Automobilbranche viele proprietäre Dateientypen verwendet werden und somit die einzelnen Speicherauszüge unterschiedliche und möglicherweise nicht bekannte Dateien enthalten können. Da das aber besonders wichtig für den Vergleich der Software ist, ist die erste Anforderung an diesen Datensatz somit, dass bekannt ist, welche Dateitypen und wie viele von jedem Typen enthalten sind. Des Weiteren sind nur relevante Dateien der digitalen FF enthalten, da diese erkannt werden sollen. Auf diesem erstellten und somit bekanntem Datensatz können

dann Tests und Evaluationen durchgeführt werden und so Vergleiche zwischen der Software und den KI-Modellen gezogen werden.

Des Weiteren gibt es in der KI auch Anforderungen an einen Datensatz um Modelle zu trainieren, die der gewünschten Genauigkeit entsprechen. Zunächst sollte dieser genug Dateien enthalten. Hierbei gilt meist je mehr Daten im Training zur Verfügung stehen, desto besser können die einzelnen Merkmale der Dateitypen gelernt werden und umso besser wird das Modell. Der Datensatz sollte auch ausgeglichen sein. Das heißt es sind ungefähr gleich viele Dateien von jedem Typen vorhanden. Das verhindert, dass ein möglicher Bias der größeren Klasse entsteht, denn wenn eine Klasse sehr viel weniger Dateien hat kann trotz einer guten Genauigkeit diese Klasse wahrscheinlich schlecht erkannt werden. Die Daten sollten auch alle Open Source sein, um den Datensatz reproduzierbar zu machen.

Als Grundlage für den Datensatz wurde im Internet nach Open Source Datensätzen gesucht. Als Suchbegriffe wurden die Dateitypen (DBC, CDR, CSV, XLS, GPX) verwendet. Des Weiteren wurden nach Datensätzen von GPS-Koordinaten im Automobilkontext gesucht. Folgende Datensätze sind für den synthetischen Datensatz verwendet worden:

**Opendbc** [26] ist ein Datensatz von *Comma.ai*. Dieser ist auf GitHub verfügbar und enthält 80 DBC Dateien, die für verschiedene Fahrzeuge rekonstruiert wurden.

**NHTSA Dataset** [28] ist durch die National Highway Traffic Safety Administration (NHTSA) veröffentlicht worden und enthält 9636 CDR Dateien.

**PVS - Passive Vehicular Sensors Datasets** [29] ist ein Datensatz der verschiedene Sensordaten enthält, die für ML-Modelle verwendet werden können. Er enthält 27 CSV Dateien, die GPS-Koordinaten enthalten.

**IO-VNBD (Inertial Odometry Vehicle Navigation Benchmark Dataset)** [30] enthält GPS-Dateien, die mit einem Smartphone während der Fahrt aufgenommen wurden. Er enthält 331 CSV Dateien.

**Road Vehicle Localization Dataset (MagLand)** [31] enthält GPS-Daten, die bei Fahrten durch Porto, Portugal mit einem Smartphone gemessen wurden. Er enthält 48 TXT Dateien, von denen 13 verwendet werden konnten.

### 3.1.2. Erstellung des Datensatzes

Nachdem im vorherigem Unterkapitel aufgezeigt wurde, wie die Daten erlangt werden konnten, soll nun dargestellt werden, wie die GPS-Dateien bereinigt und in die relevanten Dateitypen umgewandelt wurden. Zunächst wurden zur besseren Übersicht die Dateien bereinigt. Aus

dem IO-VNBD Datensatz wurden die Dateien in einen Ordner geladen. Hierbei wurde um die Struktur der vorherigen Ordner deutlich zu machen, jeweils für den Ordner „*Synchronized V and S datasets*“ ein „S“ und für den Ordner „*Unsynchronized V and S Dataset*“ ein „U“ vor den Dateinamen gesetzt. Somit kann die Herkunft aller Dateien nachvollzogen werden. Die Daten waren jeweils noch in „*Categorized IOVNBD Dataset*“ und „*Uncategorized IOVNBD Dataset*“ eingeteilt. Das sind jeweils die selben Daten, nur nach Fahrer sortiert. Hier wurden nur die Dateien herausgenommen, die sich im Ordner „Uncategorized“ befunden haben. Somit konnten 331 CSV Dateien erhalten werden. In dem MagLand Datensatz konnten 48 TXT Dateien gefunden werden, die GPS-Koordinaten enthalten haben. Dennoch war es nur bei 13 Dateien möglich, diese mit in den Datensatz aufzunehmen, da die Dateien, die „location\_session“ im Namen hatten unzuordenbar fehlende Felder besitzen und diese somit im nächsten Schritt nicht in einen Dataframe geladen werden konnten. Aus dem PVS Datensatz wurden nur die nicht benötigten Videos und die label, mpu und settings Dateien gelöscht, wodurch 27 Dateien erhalten wurden.

Da die GPS-Dateien nur CSV und TXT Dateien beinhalten, sollen diese nun in die relevanten Dateitypen CSV, XLS und GPX geändert werden. Diese wird mit der Python Bibliothek *pandas* erzeugt. Hierbei wurde jede Datei in einen Dataframe geladen. Diese können sehr gut mit großen Dateien umgehen und es existieren Funktionen, um die Dateien in den gewollten Formaten zu speichern. Da wenige GPS-Dateien vorhanden und diese zusätzlich sehr groß waren, wurden die eingelesenen Dateien zufällig zwischen 500 und 700 Zeilen geteilt, damit die Zusammenhänge in den Koordinaten noch erhalten bleiben. Somit konnten von anfangs 371 Dateien, 12957 CSV Dateien erzeugt werden. Ein Vorteil der Datenframes ist, dass beim erneuten Speichern der Dateien die Separator (sep = ',') und das Encoding (encoding='UTF-8') vereinheitlicht wurde. Dies war in den einzelnen Datensätzen zuvor unterschiedlich. Diese 12957 Dateien wurden in drei gleiche Teile aufgeteilt und mithilfe des Datenframes in XLS und GPX Dateien umgewandelt.

### **3.1.3. Ausgleichen der Dateien**

Da in dieser Arbeit der Datensatz auch verwendet werden soll um KI-Modelle zu trainieren, ist es nötig einen ausgeglichenen Datensatz zu erstellen. Das heißt alle Klassen, Anzahl der Dateien für einen Typen, sollen ungefähr gleich groß sein. Dies ist notwendig, damit die Modelle kleinere Klassen nicht „ignoriert“ und größere „bevorzugt“, und somit kein Bias entstehen kann. Des Weiteren kann es bei unausgeglichenen Daten dazu kommen, dass die kleinere Klasse nicht erkannt werden kann. Es können somit bessere Modelle erzeugt werden. In der KI gibt es viele Möglichkeiten einen Datensatz künstlich auszubalancieren. In diesem Kapitel wird auf das Random Over Sampling für die DBC Dateien und das Random Under Sampling für die CDR Dateien eingegangen.

	VERSION	NS_	BO_	BU_:	BS_:	CM_	VAL_
0	[VERSION ""\n\n]	\nitNS_DESC_\nitCM_\nitBA_DEF_\nitBA_\nit...	[BO_ 768 VEHICLE_STATE: 8 ADAS\n SG_ SET_ME_XF...	[BU_: ADAS RADAR NEO XXX\n\n]	[]	[CM_ SG_ 1024 RADAR_STATE 'need to find out mo...	[]
1	[]	[]	[BO_ 512 GAS_COMMAND: 6 EON\n SG_ GAS_COMMAND ...	[]	[]	[CM_ "AUTOGENERATED FILE, DO NOT EDIT"\n\n\n...	[VAL_ 513 STATE 5 "FAULT_TIMEOUT" 4 "FAULT_STA...

Abbildung 3.1.: Darstellung der ersten zwei DBC Dateien, gespeichert in einem Dataframe

Die CSV, XLS und GPX Daten wurden wie beschrieben zufällig alle 500 bis 700 Zeilen getrennt, um die Zusammenhänge der GPS-Koordinaten zu erhalten und dennoch mehr Dateien für den Datensatz zu bekommen. Ein weiterer Vorteil sind die kleineren Dateien und somit schnelleren Berechnungszeiten für die KI-Modelle.

Da die DBC Dateien im Vergleich zu den anderen Klassen weniger als 1% im originalen Datensatz ausmachen, müssen hier synthetisch neue geschaffen werden. Dies erfolgt mit der Random Over Sampling Methode. Hierbei werden zufällige Daten kopiert und neu in den Datensatz eingefügt. Da das aber schnell zu Overfitting des Modells führen kann, wurden die Dateien abgeändert. Overfitting heißt das Modell generalisiert nicht mehr so gut und kann nur noch die zum Training verwendeten Daten mit hoher Genauigkeit erkennen. Dateien, die das Modell noch nicht gesehen hat, können nicht erkannt werden. Um das zu verhindern und dennoch die Struktur der DBC Datei zu erhalten, wurden nicht funktionale DBC Dateien generiert. Es wurden zunächst 16 zufällige Dateien in einen separaten Ordner gelegt. Diese können somit im Testdatensatz verwendet werden. Um die Daten zu generieren, wurden die verbleibenden Dateien in ihre jeweiligen Blöcke, bestehend aus dem DBC-Keyword und den zugehörigen Informationen, aufgeteilt. Diese Informationen wurden in einem Dataframe gespeichert. In jeder Reihe des Dataframes ist die DBC Datei in die einzelnen Keywords aufgeteilt. Die einzelnen Informationen zu den Keywords sind in einer Liste gespeichert. In Abbildung 3.1 sind die ersten beiden Reihen dargestellt. Mithilfe dieses Dataframes kann nun eine zufällig zusammengesetzte DBC Datei aufgebaut werden. Hierbei wird für jede Spalte eine zufällige Reihe ausgesucht und innerhalb dieser Liste eine zufällig Anzahl an Objekten verwendet. So können 500 zufällige DBC Dateien erzeugt werden. Diese Dateien sind nichtfunktional und somit kann auch die zugehörigen Software `candb++` diese nicht öffnen. Jedoch können selbst funktionale Dateien von Comma.ai nicht immer geöffnet werden.

Die CDR Dateien machen 95,5% des originalen Datensatzes aus. Dieser Dateityp soll nun mit Random Under Sampling verkleinert werden um eine Favoritisierung dieser Klasse zu vermeiden. Das Random Under Sampling nimmt zufällig Datenpunkte aus dem Datensatz. Hierbei wird die Hälfte der vorhandenen CDR Dateien zufällig herausgenommen und für den synthetischen Datensatz verwendet. Im originalen Datensatz sind jedoch noch alle Dateien enthalten und können somit auch beliebig wieder mit aufgenommen werden.



	Originaler Datensatz	Synthetischer Datensatz
CDR	9.636	4.818
DBC	80	564
CSV	358	4.319
XLS	-	4.319
GPX	-	4.319
TXT	13	-

Tabelle 3.1.: Vergleich der Anzahl der Dateitypen im Originalen Datensatz mit dem im synthetisch erstellen Datensatz

Der Aufbau des finalen Datensatzes ist in der Tabelle 3.1 zu sehen. Dabei kann man erkennen, dass die Anzahl der Klassen angeglichen wurde und somit ein ausgeglichener synthetischer Datensatz erzeugt wurde.

### 3.1.4. Erstellung der SquashFS Datei

Das SquashFS-Dateiensystem ist ein read-only Dateisystem für Linux. Mit diesem können Dateisysteme, die beispielsweise archiviert werden sollen, komprimiert werden. Die komprimierte SquashFS Datei besteht aus maximal neun Blöcken. Zunächst steht der *superblock*, danach kommen die *Compression Options*. Hier werden Informationen zur Kompression gespeichert, falls diese von den Voreinstellungen abweichen. Danach werden die *datablocks* and *fragments* gespeichert. In den nächsten Blöcken *inode table*, *directory table* werden die Metadaten der Inodes und des Verzeichnisses in 8 Kbyte große Datenblöcke komprimiert. Danach folgt der *fragment lookup table*, der die Indexe der fragmentierten Dateien enthält. Der optionale *export table* enthält zusätzliche Informationen um das Squashfs Dateisystem exportierbar zu machen. Um Speicherplatz effizient zu nutzen werden in *uid/grid lookup table* uid und grid Indexe gespeichert. Der letzte Block ist die *xattr table*. Sie enthält für jede Inode erweiterte Attribute. [32] Da die Daten des Teslas in [2] in einer SquashFS Datei gefunden wurde, wird zum besseren Vergleich der synthetische Datensatz in ein SquashFS-Dateiensystem komprimiert. Unter Windows können Linuxbefehle mit dem Windows-Subsystem für Linux (WSL) ausgeführt werden. Mit *mksquashfs* können Dateisysteme in einer SquashFS Datei komprimiert werden. Die SquashFS-Datei wird aus dem synthetischen Datensatz erstellt. Unter Windows kann diese Datei mit der Software *7-zip* geöffnet werden. Auf einem Linuxsystem kann die Datei direkt gemounted werden. Der erstellte synthetische Datensatz, sowie die SquashFS Datei sind in GitHub [33] zur Verfügung gestellt.

```
$ wsl mksquashfs <source_directory> <data_directroy>/data.squashfs
```

## 3.2. Docker

Nachdem im vorgegangenen Abschnitt der Aufbau des erstellten Datensatzes beschrieben wurde, soll nun eine geeignete Implementierung der verschiedenen FC beschrieben werden. Da die ausgewählte Software auf verschiedenen Betriebssystemen läuft, wurde hier zur Virtualisierung eines Linuxsystems Docker verwendet. Zur vereinfachten Installation und Nachvollziehbarkeit sind in GitHub [33] die in diesem Unterkapitel beschriebenen Dockerfiles zur Verfügung gestellt.

### 3.2.1. Aufbau von Docker und Vergleich zur Virtual Machine

Docker ist eine in 2013 erstmals vorgestellte Software zur Virtualisierung von Applikationen. Dabei werden diese in einzelne Container gekapselt. Der Prozess nennt sich Containerisierung. Anwendungen können so in verschiedenen Umgebungen entwickelt und getestet werden, ohne auf die hohen Ressourcen einer Virtual Machine (VM) zurückgreifen zu müssen. Der größte Unterschied zur VM ist, dass kein komplettes Betriebssystem benötigt wird. In Abbildung 3.2 ist der Unterschied von Docker zur VM dargestellt. Die Container verwenden das Host Betriebssystem, was dazu führt, dass weniger Ressourcen verwendet werden müssen. Um Linux Container unter Windows zu starten muss zunächst das WSL installiert werden. Ein weiterer Vorteil ist, dass Container sehr leicht auf verschiedene Rechner geladen werden können um dort gestartet zu werden. Docker besteht aus vier Teilen, den Docker Containern, dem Docker Client-Server, den Docker Images und der Docker Engine. Die Docker Engine, die den Docker Daemon beinhaltet, ist der wichtigste Teil. Durch sie können Container erstellt und zum laufen gebracht werden. Durch den Docker Client-Server können Befehle an den Docker Daemon übergeben werden. Das Docker Image kann durch eine Dockerfile erstellt werden. Von diesem Image können dann die Container erstellt werden, auf denen die Applikationen laufen. [34, S. 7-17 und S.27-35][35]

Die vielen Vorteile, die Docker mit sich bringt, machen sich vor allem in der Zeit und den Ressourcen, die ein Docker Container benötigt, um zu starten, deutlich. In [35] wurde die CPU Leistung, der Speicherdurchsatz, der I/O eines Speichermediums, der Lasttest und die Betriebsgeschwindigkeit einer VM und von einem Docker Container gemessen. In allen Tests schnitt Docker besser ab als die VM. Wegen diesen Ergebnissen und dem Vorteil, dass Container sehr ressourcenarm laufen können, wurde für die Virtualisierung der Software in dieser Arbeit Docker verwendet. Ebenso können so alle Tests einfach nachimplementiert werden, was die Transparenz und Nachvollziehbarkeit dieser Arbeit erhöht.

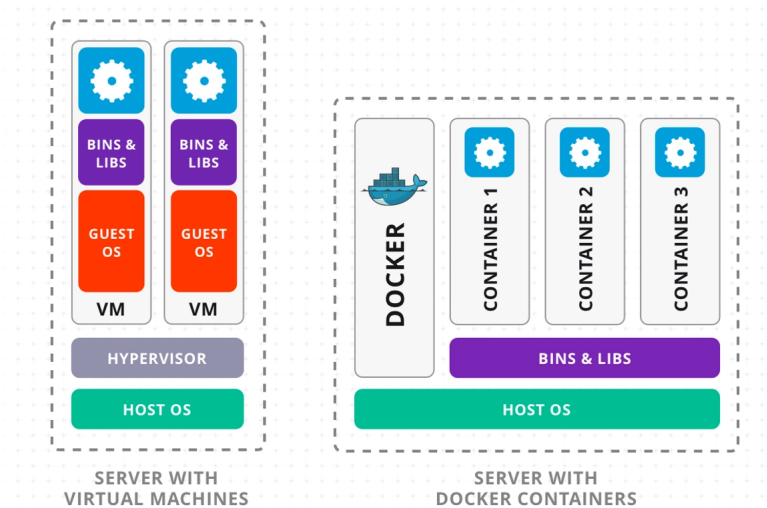


Abbildung 3.2.: Vergleich einer Virtuellen Maschine mit Docker [36]

### 3.2.2. Dockerfile

Die in dieser Arbeit erstellten Dockerfiles basieren auf bereits implementierten Dockerfiles<sup>123</sup>. Es wurden zwei Docker Images erstellt, um die klassischen von den KI-basierten FC zu trennen. Das erste **A.1** beinhaltet Autopsy, Foremost, Scalpel und Bulk\_extractor. Das zweite **A.2** beinhaltet Scedan. Bei beiden Dockerfiles wurde gleich vorgegangen. Zunächst werden die benötigten Packages mit apt und apt-get heruntergeladen und installiert. Danach werden die einzelnen Softwaretools heruntergeladen und installiert. Autopsy ist in dem erstellten Dockercontainer zwar verfügbar, jedoch wurde die Software auf dem Windows-PC verwendet.

Beide Docker Container können mit dem selben Befehl gestartet werden. Dabei können verschiedene Optionen mitgegeben werden. Diese werden in der Dokumentation von Docker run<sup>4</sup> beschrieben. Durch -d wird der Container im Hintergrund ausgeführt. Mit -rm wird der Container automatisch beim beenden gelöscht und -it sorgt dafür, dass mit der Console des Containers interagiert werden kann. Der Ordner case enthält alle Dateien und wird mit -v im Dockercontainer im Ordner /case gemountet.

```
$ docker build -t <containername> .
```

```
$ docker run -d --rm -it \
```

<sup>1</sup><https://github.com/k-gomez/ctf-docker>, zuletzt aufgerufen am 20.11.2023

<sup>2</sup>[https://github.com/bannsec/autopsy\\_docker](https://github.com/bannsec/autopsy_docker), zuletzt aufgerufen am 20.11.2023

<sup>3</sup><https://gist.github.com/jgru/eef181439fe4828b9281fae43fc9f193>, zuletzt aufgerufen am 20.11.2023

<sup>4</sup>[https://docs.docker.com/engine/reference/commandline/container\\_run/](https://docs.docker.com/engine/reference/commandline/container_run/), zuletzt aufgerufen am 20.01.2024

```
-v /d/Bachelorarbeit/FileCarver/case:/case <containername>  
$ docker exec -it <containerid> /bin/bash
```

### 3.3. KI File Carver

Nachdem in den vorgegangenen Abschnitten der Aufbau des synthetischen Datensatzes und der erstellten Dockerfiles dargestellt wurde, soll in diesem das Training der Modelle aufgezeigt werden. Es wurde hierfür Python 3.8.8, die Python-Bibliothek *scikit-learn* 0.24.1, und Jupyter Notebook 6.3.0 verwendet.

Hierbei wird zunächst auf CarveML eingegangen. Es wurden die einzelnen Funktionen aus dem Github von CarveML in ein Jupyter Notebook eingefügt. Das dient zu besserer Übersicht der einzelnen Funktionen. Einzelne Ausdrücke wurden überarbeitet, da CarveML im Jahr 2013 veröffentlicht wurde und Python und die verwendete Python-Bibliothek *sklearn* geupdatete Ausdrücke benötigen. Die Label der Klassen mussten auch angepasst werden, da für das Training die richtigen Dateitypen notwendig sind. Ebenfalls wurde die Variable `loss` der linearen SVM Funktion von `l2` zu `squared_hinge` geändert. Um das Modell zu trainieren muss zunächst eine Datei mit den Trainings- und Testvektoren erstellt werden. Da die gleichen Vektoren beim Training und beim Testen verwendet werden können wurden alle Vektoren der Dateien mit der Funktion `create_data_train()`<sup>5</sup> erstellt und in eine CSV Datei geladen. Diese CSV Datei kann dann von der Funktion `process_training_examples()`<sup>6</sup> eingelesen und in eine Menge von Variablen `X` und den zugehörigen Zielvariablen `y` geteilt werden. Diese Menge an Variablen wird mit `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=1)` in einen Trainings- und Testdatensatz unterteilt. Da Scadan die Testgröße von 50% als default verwendet, wurde hier derselbe Wert genommen, um die Beiden vergleichbar zu machen. Die SVM erreicht eine Genauigkeit von 100%. Hierbei ist anzumerken, dass der erstellte synthetische Datensatz sehr unterschiedliche Dateitypen enthält, wie in Abbildung 2.5 zu sehen ist und wahrscheinlich deshalb eine bessere Genauigkeit als die Modelle des Papers aufweist. Des Weiteren enthält dieser Datensatz weitaus weniger Dateitypen und mehr Dateien für jeden Typen, als der auf dem ursprünglich trainiert wurde. Wenn auf 1% der Daten des synthetischen Datensatzes trainiert wird, und damit näher an die Verteilung der Dateien in dem im Paper benutzten Datensatz angeglichen wird, verhalten sich die Ergebnisse wie in der originalen Arbeit. Die Genauigkeit wird in Tabelle 3.2 dargestellt. Da auch auf diesem Datensatz die SVM die besten Ergebnisse erzielt, wird nur das Modell weiter verwendet.

---

<sup>5</sup>[https://github.com/a10y/carveml/blob/master/make\\_data\\_train.py](https://github.com/a10y/carveml/blob/master/make_data_train.py), zuletzt aufgerufen am 20.12.2023

<sup>6</sup><https://github.com/a10y/carveml/blob/master/train.py>, zuletzt aufgerufen am 20.12.2023

	Accuracy
SVM	100.0%
Multinomial Naive Bayes	49.76%
LDA	86.16%

Tabelle 3.2.: Accuracy der Modelle mit CarveML trainiert auf 1% der Daten

In diesem Abschnitt soll das Training des zweiten ML-Modells Scedan aufgezeigt werden, da auch hier kein vortrainiertes Modell vorhanden ist. Scedan ist ein auf Linux getestetes Tool, deshalb wird auch hier in einem Docker Container gearbeitet. Um Scedan zu trainieren müssen alle Dateien in einer bestimmten Ordnerstruktur zur Verfügung stehen. Dabei müssen alle Dateien in der Hierarchie, die in Abbildung 3.3 zu sehen ist, gespeichert sein. Die Namen der Unterordner dienen hier auch gleichzeitig als Klassennamen. Die richtige Ablage der Daten kann mit `-validate` geprüft werden. Danach kann man das Modell trainieren. Hierbei muss mindestens ein Pfad zu den Trainingsdaten `-data` und ein Pfad zu dem Ordner `-exp` in dem die Ergebnisse gespeichert werden, angegeben werden.

```
$ cd /opt/scedan/tools
$ python3 scedan_train.py --data=/case/DATA/data --exp=<outputfolder>
```

Zum Trainieren wird zunächst der default Wert in allen Variablen genommen<sup>7</sup>. Jedoch wird die Confusionmatrix und die Accuracy als 0.0 ausgegeben. Dies kann daran liegen, dass die Ergebnisse in einem Python Shelf Objekt gespeichert werden. Diese Datenbank kann jedoch nicht mehr geöffnet werden, nachdem sie geschlossen wurde, da sie durch die Software beschädigt abgespeichert wird. Die verwendeten Python-Version des Dockerimage ist 3.8.10. Um dieses Problem zu lösen, wurde vor dem Schließen des Shelf Objekts dieses in eine Text Datei geschrieben. So kann es eingesehen und die Informationen weiterverwendet werden. Der Code 3.1 wurde dazu in `scedan_train.py` eingefügt. Durch diese Datei konnte erkannt werden, dass alle Dateien richtig klassifiziert worden sind, und somit eine Accuracy von 100% erreicht werden konnte. Wie bei CarveML schon erwähnt, könnte dieses sehr gute Ergebnis durch die wenigen und sehr unterschiedlichen Dateitypen zustande kommen.

<sup>7</sup>[https://github.com/UTSA-cyber/scedan/blob/master/doc/training\\_procedure.md](https://github.com/UTSA-cyber/scedan/blob/master/doc/training_procedure.md), zuletzt aufgerufen am 24.01.2024

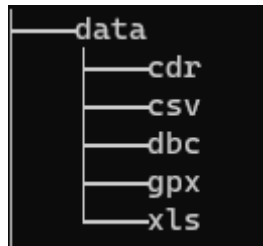


Abbildung 3.3.: Struktur der Ordner zum Training von Scedan

```
# write shelve db in txt file
db.sync()
with open(expname("dbexperiment.txt"), 'w') as f:
    for (key, val) in db.items():
        f.write("{}={}\n".format(key, val))
```

Code 3.1: Schreiben von Shelf Objekt in eine Text Datei

## 4. Testdesign und -durchführung

Nachdem im vorherigen Kapitel die Implementierung der Software und das Training der KI-Modelle dargestellt wurden, sollen nun diese Tools getestet werden. Hierzu werden zunächst die verwendeten Evaluationsmetriken und Akzeptanzkriterien definiert. Ebenso wird festgelegt, wie ein Test durchgeführt werden soll. Danach werden die einzelnen Tools getestet und gegebenenfalls verbessert.

### 4.1. Evaluationsmetriken und Akzeptanzkriterien

Zunächst sollen die Evaluationsmetriken und die Akzeptanzkriterien dargestellt werden. Anhand dieser können die verschiedenen Softwaretools verglichen werden. Es wurden sechs Akzeptanzkriterien festgelegt, die sich aus Anforderungen an diese Arbeit und denen der digitalen FF ergeben. Diese werden im Folgenden beschrieben.

#### Akzeptanzkriterien

**K1: Die zu testende Software sollte mehr als 93,09% der Daten des Datensatzes erkennen.** Die Software sollte besser sein als in [2] beschrieben. Hier wurden mit `os.walk()` und der Python Bibliothek `python-magic` 6,91% der Dateien nicht gefunden.

**K2: Alle Dateitypen sollten erkannt werden.** Dieses Kriterium befasst sich mit den Dateitypen. Es sollten alle definierten Dateientypen erkannt werden und nicht nur einzelne.

**K3: Die Software sollte nachvollziehbare Ergebnisse produzieren.** Diese Kriterium ist für die digitale Fahrzeugforensik besonders wichtig, da die gefundenen Daten zur Strafverfolgung nachvollziehbar sein müssen [37].

**K4: Die gefundenen Daten sollten reproduzierbar sein.** Die Software mit gleichen Einstellungen und gleichen Dateien sollte immer die selben Ergebnisse liefern.

**K5: Die Software sollte erweiterbar sein.** Es sollte möglich sein neue Dateitypen hinzuzufügen oder herauszunehmen.

**K6: Die Suche nach den Dateien sollte möglichst schnell sein.** Dieses Kriterium zählt zu den weichen Kriterien, da hier keine genauen Kennzahlen bestimmt werden können.

## Evaluationsmetriken der KI

In der Klassifikation in dem ML werden verschiedene Evaluationsmetriken verwendet. Diese dienen zur Bestimmung der Güte eines Modells. In dieser Arbeit werden verschiedene Metriken verwendet, die in [38], [39] vorgestellt wurden.

Die erste Metrik ist klassischerweise die **Confusionsmatrix**. Hierbei werden vertikal die richtigen Klassen angegeben und horizontal die vorhergesagten Klassen. Somit ergeben sich Werte für True Positive (TP), False Negative (FN), False Positive (FP) und True Negative (TN). Damit ergibt sich eine Diagonale, die richtig vorhergesagten Klassen anzeigt. Die Abbildung 4.1 zeigt beispielhaft eine Confusionsmatrix für zwei Klassen. In der Confusionsmatrix werden auch die Dateien aufgezeigt, die falsch klassifiziert wurden. Aus dieser Matrix lassen sich mehrere Werte ableiten.

Die **Accuracy** oder Genauigkeit ist der Quotient aus allen richtig klassifizierten Daten mit allen Daten.

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

Um die Accuracy mit mehr als zwei Klassen zu berechnen können einfach die Werte auf der Diagonale der Matrix addiert und durch die Gesamtanzahl der Werte geteilt werden. Die Accuracy gibt hierbei an, wie viele Dateien richtig klassifiziert worden sind.

Ein weiterer Wert, ist die **Precision**, diese gibt an wie viele der vorhergesagten Klasse richtig vorhergesagt wurden.

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

In einer multiklassen Klassifikation kann man entweder die Precision für jede Klasse einzeln berechnen oder man kann für alle Klassen einen Wert berechnen und diesen mitteln.

Der nächste Wert der hier berechnet werden kann ist der **Recall**. Dieser gibt an wie viele Daten einer Klasse richtig vorhergesagt werden können.

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

Wie auch bei der Precision kann der Recall bei mehreren Klassen gemittelt werden.

Da bei Precision und Recall nur eine Klasse berücksichtigt wird, wird zur weiteren Betrachtung auch der **F1-Score** verwendet.

$$F1 = 2 \cdot \left( \frac{Precision \cdot Recall}{Precision + Recall} \right) = \frac{2TP}{2TP + FP + FN}$$

Dieser Score betrachtet Precision und Recall gleichzeitig, somit ist er sensibel für falsch klassifizierte Daten.

Da die Klassen in dem erstellten Datensatz nicht alle gleich groß sind, wird auch noch der **Matthews Correlations Coefficient (MCC)** betrachtet. Dieser ist sensibler gegenüber der



unterschiedlichen Verteilung der Klassen.

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Er liegt zwischen  $[-1; 1]$  wobei  $MCC = 1$  eine perfekte Klassifikation bedeutet. Alle anderen beschriebenen Werte liegen alle zwischen  $[0; 1]$ .

		PREDICTED		
		Positive (1)	Negative (0)	Total
ACTUAL	Positive (1)	TP = 20	FN = 5	25
	Negative (0)	FP = 10	TN = 15	25
Total		30	20	<b>50</b>

Abbildung 4.1.: Confusionsmatrix einer binären Klassifikation [39]

Der **Cohen's Kappa Koeffizient**  $\kappa$  zeigt auf, ob das Modell besser als ein zufälliges ist. Die Accuracy eines Modells, das zufällig Klassen auswählt wird mit  $p_e$  bezeichnet und die reale Accuracy eines Modells mit  $p_0$ .

$$\kappa = \frac{(p_0 - p_e)}{(1 - p_e)}$$

### Evaluationsmetriken der FC

Um die Güte eines FC zu ermitteln, verwendet [6] modifizierte Metriken. Dabei werden im **carving Recall** die Anzahl der Dateien betrachtet, die tatsächlich durch das Tool gefunden werden. Es werden zunächst von der Anzahl an allen Dateien im Datensatz ( $all$ ), die nicht gefundenen Dateien, die aber durch die Software unterstützt werden ( $sf n$ ) und die, die nicht durch die Software erkannt wurden, da der Dateityp nicht unterstützt wird ( $uf n$ ), abgezogen und durch  $all$  geteilt.

$$carving\_Recall(cR) = \frac{all - sf n - uf n}{all}$$

Im **supported Recall** werden nur die Dateien betrachtet, die der FC auch erkennen sollte ( $sp$ ).

Dabei wird der Quotient aus den korrekt erkannten und allen Dateien, die unterstützt werden, gebildet. Dieser Wert gibt an, wie gut Dateien erkannt werden, die der FC unterstützt.

$$\text{supported\_Recall}({}_sR) = \frac{sp - sfn}{sp}$$

Die **carving Precision** gibt an wie korrekt Dateitypen erkannt werden. Ein hoher Wert wird erreicht, wenn wenige False Positives unter den gefundenen Dateien sind. Der Wert kann durch den Quotienten der korrekt gefundenen Dateien ( $tp$ ) und diesen addiert mit den Dateien, die nicht inkorrekt durch die Software identifiziert wurden, aber inkorrekt sind ( $ufp$ ) und den gefundenen Dateien, die als inkorrekt identifiziert wurden ( $kfp$ ), berechnet werden.

$$\text{carving\_Precision}({}_cP) = \frac{tp}{tp + ufp + \frac{1}{2}kfp}$$

Das **carving F-Measure** ist, wie der F1-Score um die Werte von Recall und Precision zu berücksichtigen. Es wird in dieser Arbeit ein  $\alpha = 0,5$  verwendet.

$$\text{carving\_Fmeasure}({}_cFm) = \frac{1}{\alpha \frac{1}{{}_cP} + (1 - \alpha) \frac{1}{{}_cR}}$$

### Testaufbau

Durch die oben genannten Akzeptanzkriterien und Evaluationsmetriken ergibt sich folgender Testaufbau. Die Software wird zweimal hintereinander auf dem synthetischen Datensatz getestet. Hierbei wird gespeichert welche Dateien erkannt wurden, und es wird auf falsch vorhergesagte Dateitypen geprüft. Ebenfalls wird die Dauer der Suche gespeichert.

## 4.2. Erster Ansatz

Nachdem im vorherigen Kapitel die Akzeptanzkriterien und Testmetriken definiert wurden, wird nun die Software auf dem erstellten Datensatz getestet. Der erste Test erfolgt ohne weitere Anpassungen, wenn diese nicht benötigt werden, um die Software zu verwenden. Dabei werden Autopsy und FTK Imager in Windows 11 verwendet und Foremost, Scalpel und Bulk\_extractor werden auf Ubuntu 20.04 in Docker gestartet. Die verwendeten Befehle werden unter dem Absatz angegeben. Zunächst werden nur die klassischen FC betrachtet, da das Training der KI-Modelle schon im vorherigen Kapitel aufgezeigt wurde. Getestet wurde auf einem Windows 11 PC mit einer i5-9400F CPU und 16 GB RAM.

**Scalpel** benötigt vor dem ersten Start der Software, die zu suchenden Dateitypen. Diese sind zunächst in `scalpel.conf` alle auskommentiert. Ein erster Versuch ist alle vordefinierten Da-

teitypen zu berücksichtigen, und somit alle Kommentare zu löschen. Scalpel wurde mit dem untenstehenden Befehl gestartet. Bei diesem muss der Pfad zur Konfigurationsdatei mit `-c` und der Ordnerpfad für den Ordner mit den Ergebnissen mit `-o` angegeben werden. Danach kann der Pfad zu dem zu überprüfenden Ordner angegeben werden. Nach allen möglichen Dateitypen zu suchen ist jedoch nicht empfohlen, da es so zu sehr vielen falsch gefundenen Dateien kommen kann. Da aber in einem Fahrzeugdump nicht bekannt ist, welche Dateitypen enthalten sind, wurde dies als erster Ansatz gewählt. Scalpel konnte dennoch keine Dateien in dem erstellten Datensatz finden. Um dieses Ergebnis zu verbessern sollen bekannte Header und Footer der XML/GPX und XLS Dateien im nächsten Schritt definiert werden.

```
$ scalpel -c /etc/scalpel/scalpel.conf -o /case/scalpel_data \
/case/squashfs/data
```

**Foremost** enthält, wie Scalpel, eine Konfigurationsdatei `foremost.conf`. Es können jedoch schon vordefinierte Dateien mit dem Befehl `-t` ausgewählt werden. Ein Problem, das wie bei Scalpel auch hier auftritt wenn nicht bewusst ist nach welchen Dateitypen gesucht wird ist, dass alle Dateitypen in der Konfigurationsdatei unkommentiert werden müssen. Das kann zu vielen falsch gefundenen Dateien führen. Mit Foremost können XLS Dateien im Gegensatz zu Scalpel standardmäßig erkannt werden. Jedoch ist es nicht möglich, den in dem Docker Container gemounteten Ordner einzulesen, da kein Dateiendzeichen (EOF) gefunden werden kann. Es konnte nur die SquashFS Datei eingelesen werden. Bei dieser konnten keine Dateien erkannt werden. Im nächsten Schritt soll der XML/GPX Dateityp in die Konfigurationsdatei eingefügt werden.

```
$ foremost -t xls -o /case/foremost_data_squashfs /case/data.squashfs
```

**Bulk\_extractor** kann Dateisysteme und Speicherabbilder durchsuchen. Zunächst wird die Squashfs Datei durchsucht. Es kann jedoch keine Datei oder Information gefunden werden. Das Dateisystem wird mit `-R` aufgerufen, damit werden die Ordner rekursiv durchsucht. Mit `-o` kann der Pfad zum Ausgabeordner angegeben werden. Bei dieser Suche konnten Internetdomainen, URL-Adressen und GPS-Daten gefunden werden. Es ist jedoch nur möglich JPEG, ZIP und RAR Dateien zurückzugeben. Da diese nicht in dem Datensatz enthalten sind, werden auch nach wiederholtem Aufrufen keine Dateien gefunden. Die Suche benötigte 228,7 Sekunden und hatte eine Geschwindigkeit von 9.828 MBytes/sec.

```
$ bulk_extractor -o /case/bulk_data_squashfs /case/data.squashfs
$ bulk_extractor -o /case/bulk_data -R /case/squashfs/data
```

In **Autopsy** muss zunächst ein neuer Fall erstellt werden. Danach kann eine Datenquelle hinzugefügt und die zu verwendenden Module bestimmt werden. Die folgenden Module wurden ausgewählt, um das Dateisystem zu durchsuchen:

File Type Identification, Extension Mismatch Detector, Embedded File Extractor, Interesting Files Identifier, PhotoRec Carver, Data Source Integrity.

Mithilfe dieser Module konnten von allen Dateien des Datensatzes die Multipurpose Internet Mail Extensions (MIME) Typen erkannt werden. Die Dateinamenserweiterung wurde jedoch nur bei den XLS Dateien erkannt. Der zweite Durchlauf ergab die gleichen Ergebnisse. Die Suche dauerte hier 188 Sekunden. Ein weiterer Vorteil von Autopsy ist, dass ein Report beispielsweise in einer XML oder HTML Datei erstellt werden kann, und darin alle gefundenen Dateien aufgelistet werden können. Somit muss nicht das komplette Speicherabbild manuell durchsucht werden.

Da **FTK Imager** den Windows Explorer verwendet um die Dateisysteme darzustellen, können mit dieser Software keine weiteren Informationen gewonnen werden. Es kann ebenfalls kein Report oder ähnliches erzeugt werden, der einen Überblick über die enthaltenen Dateitypen gibt. Es ist auch nicht möglich weiteren Einstellungen vorzunehmen. Jedoch kann ein Image von der mit 7-zip geöffneten SquashFS Datei erstellt werden. Dieses wird im nächsten Schritt verwendet um Foremost und Scalpel darauf zu testen, da in der Squashfs Datei nichts gefunden werden konnte.

	Autopsy	Foremost	Scalpel	FTK Imager	bulk_extractor
gefundene Dateien (in %)	100 (MIME)	-	-	0	0
Zeit (in Sek.)	188	-	-	-	228,7
	CarveML	Sceadan			
gefundene Dateien (in %)	100	100			
Zeit (in Sek.)	995,61	655,09			

Tabelle 4.1.: Ergebnisse des ersten Ansatzes der File Carver

### 4.3. Anpassung der Software

Nach dem ersten Test der Software, sollen nun hier die möglichen Verbesserungen beschrieben werden. Bei Bulk\_extractor können keine neuen Dateitypen definiert und nur eigene Scanner angebunden werden. In dieser Bachelorarbeit werden jedoch lediglich die vordefinierten Funktionen getestet, somit kann hier keine Verbesserung vorgenommen werden. Ebenso besitzt der FTK Imager keine Möglichkeit benutzerdefinierte Dateitypen zu erkennen. Da CarveML in der Klassifikation 100% erreicht, werden hier ebenfalls keine weiteren Anpassungen vorgenommen.

Zunächst sollen die Header und Footer der XML bzw. GPX Dateien in **Scalpel** und **Foremost**

ergänzt werden. Bei Scalpel soll zusätzlich die Information für die DOC Datei, zu denen die XLS Datei zählt, hinzugefügt werden. Diese ist aus der Konfigurationsdatei von Foremost entnommen worden. Die Header und Footersequenzen werden wie folgt definiert. Zunächst wird die Dateiendung angegeben, wie beispielsweise xml. Danach kann festgelegt werden, ob die Groß- und Kleinschreibung betrachtet werden soll. Dann folgt der Header und der Footer. In diesem Fall wurde sich für die XML und GPX Datei an der Definition einer HTML Datei orientiert. Da die GPX Datei eine XML Datei ist, wurden beide Dateitypen mit aufgenommen. Als Header wurde <?xml und <gpx und als Footer </xml> und </gpx> verwendet. Ebenfalls wurde mit FTK Imager ein Speicherabbild des geöffneten SquashFS Dateisystems gemacht, um eine mögliche Inkompatibilität der FC mit SquashFS zu lösen. Jedoch konnten selbst mit den neu definierten Dateitypen in Scalpel keine Dateien gefunden werden. Auch unter Verwendung des neu erstellten Speicherabbildes konnten keine Dateien erkannt werden. Die Suche dauerte dabei 5 Sekunden. Ebenso konnte Foremost keine der neu definierten Dateien finden. Die Suche dauerte 11 Sekunden. Das könnte möglicherweise durch eine Inkompatibilität von Foremost und Scalpel mit einem Dateisystem und dem verwendeten Speicherabbild erklärt werden.

```
# in foremost.conf und scalpel.conf hinzufügen
xml n 60000 \x3c\x3f\x78\x6d\x6c\x20 \x3c\x2f\x78\x6d\x6c\x3e
gpx n 60000 \x3c\x67\x70\x78\x20 \x3c\x2f\x67\x70\x78\x3e

# nur in scalpel.conf
doc y 12500000 \xd0\xcf\x11\xe0\xa1\xb1
```

**Autopsy** kann den MIME Typen jeder Datei erkennen, jedoch nicht die Dateiendungen. Der MIME Typ gibt an, um welche Dateiklasse es sich bei der gefundenen Datei handelt. Er wird meist von Browsern verwendet um empfangenen Dateien korrekt verarbeiten zu können [40]. Hier gibt es unter Tools > Options > File Types die Möglichkeit, diese selbst zu definieren. Die CDR Dateien werden unter dem MIME Type application/octet-stream kategorisiert. Dies ist der Standard für unbekannte Typen, die binäre Daten enthalten. Für diese Dateien wird ein benutzerdefinierter Typ unter dem Namen application/cdr erstellt. Die CDR Dateien enthalten alle an Offset 391 den Text „Crash Data Retrieval Tool“. Das könnte zur Erkennung dieses Dateitypen benutzt werden. Die GPX Datei wird ebenfalls, wie bei Scalpel, durch „<gpx“ an Offset 40 definiert. Beim wiederholten Testen von Autopsy konnte festgestellt werden, dass die benutzerdefinierten MIME Typen nicht erkannt werden konnten. Das könnte daran liegen, dass die ausgewählten Signaturen nur Text sind, der nicht immer an der gleichen Stelle stehen muss.

Bei **Sceadan** ist die Möglichkeit gegeben, den ersten Block mit `-train_noblock0` aus dem Training auszuschließen. Somit werden die Header nicht mit in das Training aufgenommen. Das Modell wurde mit diesem Modus noch einmal trainiert. Es wurde jedoch immer noch die selbe Accuracy von 100% wie in Kapitel 3.3 erreicht. Hierbei könnte es sich um ein Overfitting des Modells handeln. Das Ergebnis kann aber auch durch eine perfekte Trennbarkeit der einzelnen Dateitypen erklärt werden. In Kapitel 5.2 wird genauer darauf eingegangen.

## 5. Evaluation und Diskussion

Nachdem im vorherigen Kapitel die Software getestet und gegebenenfalls verbessert wurde, soll diese nun verglichen werden. Hier werden zunächst die FC und dannach die KI-Modelle behandelt. Zuletzt sollen die Ergebnisse diskutiert werden.

### 5.1. Evaluation der File Carver

Foremost, Scalpel, FTK Imager und Bulk\_extractor konnten keine Dateien erkennen. Somit liegen alle Evaluationsmetriken bei 0%. Der Vollständigkeit halber wurden sie trotzdem mit in die Tabelle 5.1 aufgenommen.

	Autopsy	Foremost	Scalpel	FTK Imager	bulk_extractor
<i>carving_Recall</i>	1,0	0,0	0,0	0,0	0,0
<i>supported_Recall</i>	1,0	0,0	0,0	0,0	0,0
<i>carving_Precision</i>	0,9778	0,0	0,0	0,0	0,0
<i>carving_Fmeasure</i>	0,9888	0,0	0,0	0,0	0,0

Tabelle 5.1.: Vergleich der Metriken der klassischen File Carver

**Autopsy** konnte alle Dateien im synthetischen Datensatz erkennen. Die in Abbildung 5.1 dargestellte Confusionsmatrix zeigt die gefundenen MIME-Typen. Da CDR und DBC Dateien keinem dieser Typen zugeordnet werden konnten, wurden sie allgemeinen Typen zugeordnet. Dabei beschreibt das Label `octet-stream` eine allgemeine Datei die nicht-leserliche Binärdaten enthält. Unter diesem Typen wurden die CDR Dateien erkannt. Unter das Label `text_plain` fallen alle Dateien, die einen lesbaren Text enthalten. Hier wurden die DBC Dateien erkannt. Es ist zu beobachten, dass bis auf 24 CDR Dateien allen das richtige Label zugeordnet werden konnte. Somit ergeben sich für den carving und supported Recall 100%, da alle Dateien des Datensatzes erkannt wurden. Die Korrektheit wird mit der carving Precision bestimmt. Als TP wurden hier nur die 12957 Dateien gezählt, die auch als ein spezifischer MIME-Type erkannt wurden. Das sind die XLS, XML und CSV Dateien. Es wurden keine Dateien inkorrekt identifiziert, die nicht erkannt wurden. Mit dem Modul `Extension Mismatch Detector` wurden die 588 als

text\_plain gelabelten Dateien, als inkorrekt identifiziert. Somit ergibt sich  $kfp = 588$  und eine carving Precision von 97,78%. Das carving F-measure lässt sich, mit dem gewählten  $\alpha = 0,5$  aus den errechneten Werten, berechnen. Alpha wurde so gewählt, da so Precision und Recall in gleichen Teilen betrachtet werden. Somit ergeben sich die folgenden Metriken, die in Tabelle 5.1 dargestellt werden.

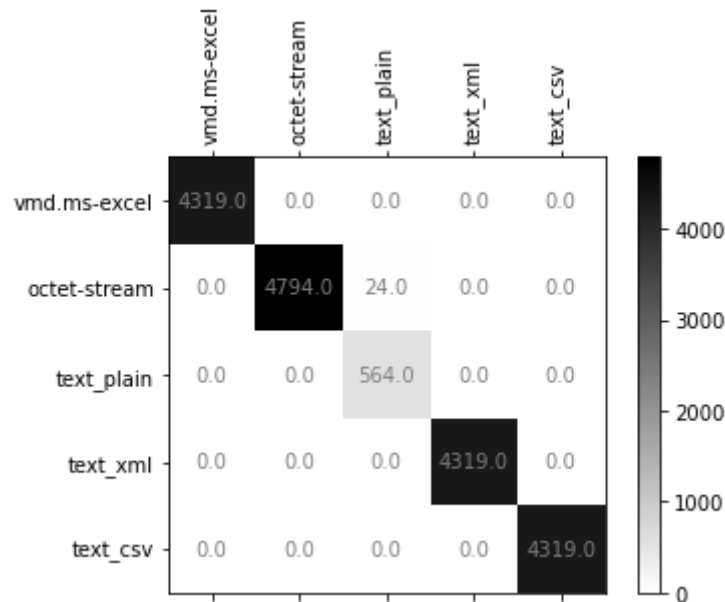


Abbildung 5.1.: Confusionsmatrix von Autopsy

Es sollen nun die Akzeptanzkriterien für alle FC überprüft werden. Das erste Akzeptanzkriterium **K1** wurde nur von Autopsy, mit einer Accuracy von 96,92%, erfüllt. Da bei keinem File Carver alle Dateitypen erkannt werden konnten, wurde **K2** von keinem erfüllt. Jedoch konnte Autopsy die meisten Typen erkennen, und hat somit dieses Kriterium am besten erfüllt. Alle FC sind nachvollziehbar, da sie aufgrund bekannter Dateisequenzen die Dateien extrahieren. Somit erfüllen alle FC **K3**. **K4** ist ebenfalls erfüllt, da ein neuer Durchlauf bei allen FC die selben Ergebnisse enthalten hat. Jeder FC außer der FTK Imager kann durch Scanner erweitert werden. Es können ebenfalls neue Dateitypen definiert werden. Bis auf den FTK Imager erfüllen alle **K5**. Die Zeit, die eine beispielhafte Suche auf dem synthetischen Datensatz, benötigt, wird in Tabelle 5.2 dargestellt. Bei dem FTK Imager kann keine Zeit bestimmt werden, da hier die das Dateiensystem nur geöffnet, aber nicht nach Dateien durchsucht wurde.



	Autopsy	Foremost	Scalpel	FTK Imager	bulk_extractor
<b>K1:</b> (in Prozent)	96,29	0	0	0	0
<b>K2:</b>	nicht erfüllt	nicht erfüllt	nicht erfüllt	nicht erfüllt	nicht erfüllt
<b>K3:</b>	erfüllt	erfüllt	erfüllt	erfüllt	erfüllt
<b>K4:</b>	erfüllt	erfüllt	erfüllt	erfüllt	erfüllt
<b>K5:</b>	erfüllt	erfüllt	erfüllt	nicht erfüllt	erfüllt
<b>K6:</b> (in Sekunden)	188	11	5	-	228.7

Tabelle 5.2.: Vergleich der Akzeptanzkriterien der File Carver

## 5.2. Evaluation der KI-basierten Ansätze

**Carve ML** erreicht auf dem synthetischen Datensatz eine Accuracy von 100%. Dies ist normalerweise ein Zeichen für Overfitting, das heißt das Modell ist zu angepasst auf den Trainingsdaten und generalisiert schlecht. Um dies ausschließen zu können wurde ebenfalls ein Test auf dem originalen Datensatz durchgeführt. Auf diesem wird ebenfalls 100% erreicht. Ein Grund für diese Ergebnisse kann sein, dass die Aufgabe zu einfach ist und zu wenig ähnliche Dateitypen vorhanden sind. Um die Eigenschaften der Dateitypen besser darzustellen, wurde mit Hilfe einer Principal Component Analysis (PCA) die Dimension der Vektoren auf zwei reduziert. Somit konnten die einzelnen Dateien in einem zweidimensionalen Scatterplot abgebildet werden. Dieser ist in Abbildung 5.2 dargestellt. Die verschiedenen Dateitypen werden in unterschiedlichen Farben angegeben. Hierbei ist gut zu erkennen, dass die einzelnen Typen, bis auf die CDR Dateien, sehr nah zusammenliegen. Die Punktwolken sind jedoch klar definiert und abgetrennt voneinander. Das könnte vor allem in höheren Dimensionen dazu führen, dass die einzelnen Typen sehr gut unterschieden werden können. Die CDR Dateien sind sehr weit verteilt, aber klar separiert von den anderen Typen, was auch bei dieser Klasse zu den guten Ergebnissen führen kann. Die SVM ist ebenfalls ein konvexes Optimierungsproblem, somit gibt es nur ein Minimum. Aufgrund dessen besteht die Möglichkeit genau dieses zu erreichen und damit eine perfekte Klassifikation zu erzielen. Eine Accuracy von 100%, ohne dass die SVM overfittet, ist also möglich. Die Confusionsmatrix des Modells ist in 5.3 dargestellt. Hier kann festgestellt werden, dass alle Dateien richtig klassifiziert wurden. Dies unterstützen ebenfalls die Metriken in Tabelle 5.4. CarveML klassifiziert somit auf den Trainings- und Testdaten perfekt.

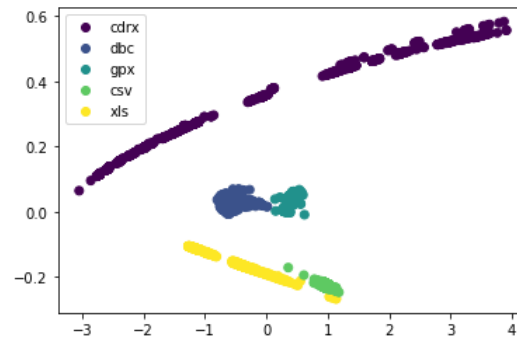


Abbildung 5.2.: Scatterplot der verschiedenen Dateitypen

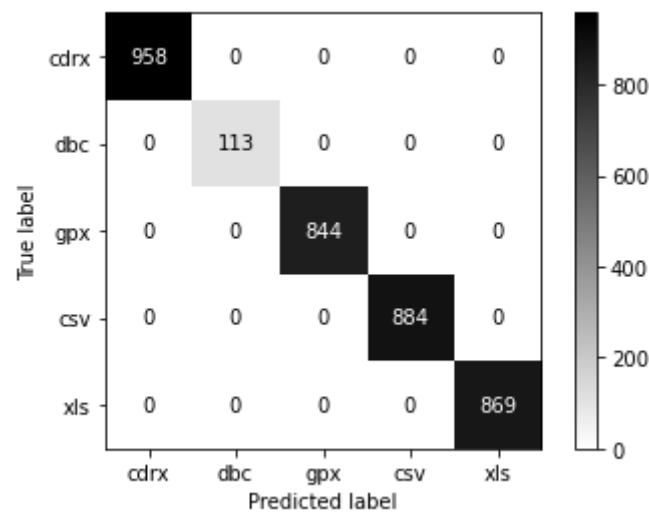


Abbildung 5.3.: Confusionsmatrix von CarveML

**Sceadan** kann nur auf Dateifragmenten trainiert werden. Es wird dennoch auf ganzen Dateien getestet. Auch mit diesem Modell wird eine Accuracy von 100% erreicht. Die Weiteren erreichten Metriken werden in Tabelle 5.4 dargestellt. Auf diese wird nicht weiter eingegangen, da sie alle bei 100% liegen, und dadurch auf eine die perfekte Klassifikation der Dateien schließen lassen. Da beide Modelle eine SVM und den selben Datensatz verwenden, kann bei Sceadan durch die erwähnten Argumente ebenfalls ein Overfitting ausgeschlossen werden.

	CarveML	Sceadan
<b>K1:</b> (in Prozent)	100	100
<b>K2:</b>	erfüllt	erfüllt
<b>K3:</b>	erfüllt	erfüllt
<b>K4:</b>	teils erfüllt	teils erfüllt
<b>K5:</b>	teils erfüllt	teils erfüllt
<b>K6:</b> (in Sekunden)	995,61	655,09

Tabelle 5.3.: Vergleich der Akzeptanzkriterien der KI-basierten File Carver

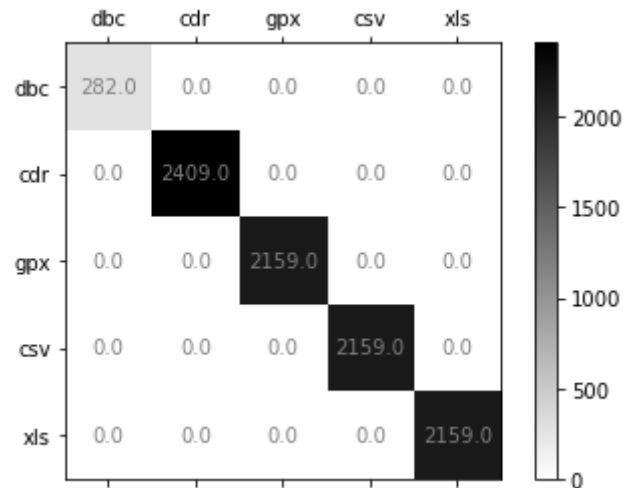


Abbildung 5.4.: Confusionsmatrix von Sceadan

Da beide Modelle gleich gut abschneiden und es sich bei beiden um das selbe Modell handelt, werden die gleichen Akzeptanzkriterien erfüllt bzw. nicht erfüllt. Diese werden in Tabelle 5.3 dargestellt. Es kann **K1** erfüllt werden, da alle Dateien perfekt erkannt wurden. Ebenfalls können alle Dateitypen richtig klassifiziert werden, was **K2** erfüllt. Da eine SVM nachvollziehbar ist, weil es sich um ein Optimierungsproblem handelt, wurde auch **K3** erfüllt. Das Akzeptanzkriterium **K4** wird erfüllt, solange das Modell nicht neu trainiert wird. Dies überschneidet sich mit **K5**, da dieses Kriterium grundsätzlich erfüllt ist. Da das Modell aber nicht neu trainiert werden darf um **K4** nicht zu verletzen, kann eines der beiden Kriterien nicht erfüllt werden. CarveML benötigt hierbei für das Erstellen der Vektoren 990,63 Sekunden und für das Training 4,98 Sekunden. Sceadan ist im Gesamtprozess schneller mit 655,09 Sekunden.

	CarveML	Sceadan	Sceadan_train_noblock0
Accuracy	1,0	1,0	1,0
Precision	1,0	1,0	1,0
Recall	1,0	1,0	1,0
F1-Score	1,0	1,0	1,0
MCC	1,0	1,0	1,0
Cohen´s Kappa Koeffizient	1,0	1,0	1,0

Tabelle 5.4.: Vergleich der Metriken von CarveML und Sceadan

### 5.3. Diskussion der Ergebnisse

Der Vergleich der klassischen FC macht deutlich, dass Autopsy sich am Besten für die digitale Fahrzeugforensik eignet. Dieser FC kann alle Dateien des synthetischen Datensatzes erkennen und klassifiziert diese zu 96,92% richtig. Ebenso werden die Dateien sehr übersichtlich in dem Ergebnis-Tree dargestellt, was eine Suche erleichtert. Die Dateitypen lassen sich erweitern, was vor allem bei proprietären Dateitypen, die in der Fahrzeugforensik gefunden werden können, ein großer Vorteil ist. Ein Nachteil dieser Software könnte sein, dass diese nur auf Windows vollständig getestet wurde. Ebenfalls könnte es zu Problemen kommen, da nicht definierte MIME-Typen alle unter dem default-Typen zusammengefasst werden. Jedoch ist dies ein deutlich kleinerer Teil, als der der zuvor zu durchsuchenden Dateien.

Zu den KI-basierten Ansätzen lässt sich sagen, dass festgestellt werden konnte, dass eine SVM sehr gut geeignet ist, um Dateitypen aus der digitalen Fahrzeugforensik zu erkennen. Der Unterschied der beiden verwendeten Modelle ist, dass CarveML deutlich flexibler angepasst werden kann, was bei Sceadan nicht der Fall ist. Es ist beispielsweise nicht möglich, auf ganzen Dateien zu trainieren. Ebenfalls ist CarveML schneller im Training und im Test als Sceadan. Algorithmen aus der KI benötigen zusätzlich einen großen Datensatz an zu klassifizierenden Dateitypen. Dieser ist nicht immer vorhanden und muss erstellt werden. Ist er jedoch vorhanden, könnten sich die Methoden als sehr hilfreich erweisen, da nichts über die Datei bekannt sein muss, um diese zu klassifizieren. Dies ist vor allem in der digitalen Fahrzeugforensik für proprietäre Dateitypen von Vorteil.

Ein großer Unterschied zwischen klassischen FC und KI-basierten Methoden ist, dass das Dateisystem für die Klassifikation mithilfe der KI benötigt wird. Das macht sie in der derzeitigen Form unbrauchbar für das Carving. Jedoch ist CarveML, mit fast fünf Sekunden, im Erkennen der Dateien deutlich schneller als Autopsy, was vor allem bei großen Datenmengen ein Vorteil sein kann. Dafür kann das Modell jedoch nur fünf Dateitypen erkennen. Hier ist Autopsy ersichtlich besser. Ein kurzer Test mit Autopsy auf dem in [2] erwähnten Tesla SquashFS zeigt, dass Autopsy

4030 von 4216 Dateien anhand der MIME-Typen einordnen kann. Somit konnten 4,41% der Dateien nicht erkannt werden, was einer Verbesserung um 2,5% entspricht.

## 6. Zusammenfassung und zukünftige Arbeit

Diese Bachelorarbeit vergleicht fünf bekannte FC miteinander. Sie untersucht, inwieweit diese im Bereich der digitalen Fahrzeugforensik spezifische Dateitypen erkennen können. Dabei wurde die Leistung anhand eines neu erstellten synthetischen Datensatzes ermittelt. Er beinhaltet relevante Dateien die in Dateisystemen von Fahrzeugen zu finden sind. Ebenfalls wurden zwei KI-basierte Ansätze betrachtet. Dabei soll festgestellt werden ob sich, durch das Erkennen von Dateitypen lediglich durch die Struktur dieser, ein Vorteil im Finden von proprietären Dateitypen ergibt.

Es konnte nachgewiesen werden, dass unter den klassischen FC Autopsy, mit einer Accuracy von 96,92%, die besten Ergebnisse auf dem synthetisch erstellten Datensatz liefern konnte. Ein Vorteil dieses FC ist es, dass benutzerdefinierte MIME Typen erstellt werden können, was vor allem in dem Bereich der digitalen Fahrzeugforensik im Umgang mit proprietären Dateitypen benötigt wird. Auch die einfache Handhabung und die verständliche Benutzeroberfläche sprechen für diese Software.

Ebenfalls wurde festgestellt, dass die SVM auf Dateien aus der digitalen Fahrzeugforensik mit einer Accuracy von 100% die besten Ergebnisse liefert. Jedoch müssen hier einige Einschränkungen gemacht werden. Zunächst arbeitet das Modell, im Gegensatz zu klassischen FC, mit dem Dateisystem. Auch können nur die im Training verwendeten Dateitypen, zugeordnet werden. Das ist eine geringere Anzahl an Dateitypen, als ein FC klassischerweise erkennen kann. Um die Klassifikation zu erweitern werden viele Dateien der neuen Klasse benötigt und das Netz muss neu trainiert werden, was sehr zeitaufwändig werden kann. Dennoch konnte CarveML in dieser Bachelorarbeit die Dateitypen schneller als die klassischen FC erkennen. Es sind noch viele Verbesserungen dieser Methode notwendig, um sie für die digitale FF brauchbar zu machen, jedoch lässt sich deutlich erkennen, dass die KI einen deutlichen Vorteil im Bereich des File Carving bieten kann.

### **Zukünftige Arbeit**

In zukünftigen Arbeiten könnte vor allem der Datensatz um weitere Dateitypen erweitert werden, um eine universellere Klassifikation zu erreichen. Hierbei könnte auch untersucht werden, ob die SVM auch bei komplexer werdenden Problemen die Dateitypen gut klassifizieren kann. Ebenfalls könnten die Erkenntnisse aus dem NLP Bereich genutzt werden, um Dateien innerhalb eines

Speicherabbilds zu erkennen, ohne die Dateistruktur zu benötigen. Dies könnte vor allem mit Deep Learning Verfahren erreicht werden. Auch sollte definiert werden, wie in dem Algorithmus mit unbekannten Dateitypen umgegangen werden soll.

## A. Anhang

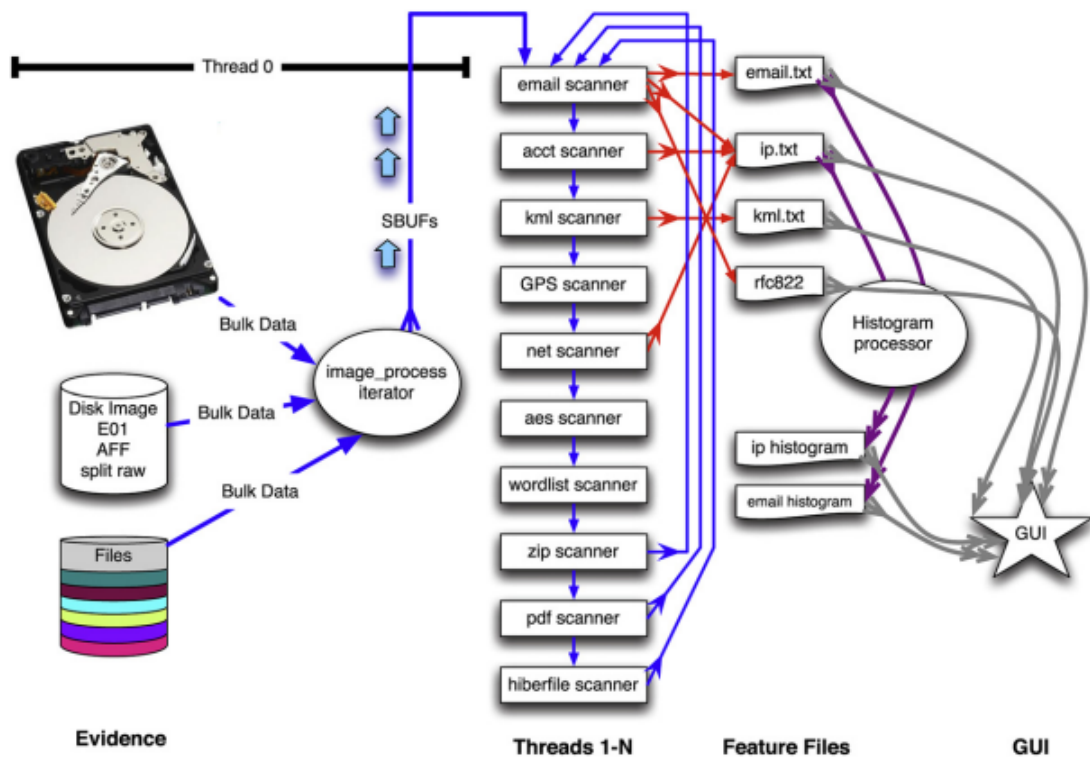


Abbildung A.1.: Arbeitsablauf von Bulk\_extractor [13]



```

FROM ubuntu:20.04

ARG DEBIAN_FRONTEND=noninteractive

# install java version
RUN apt update
RUN apt install -y testdisk wget gnupg

RUN wget -q -O - https://download.bell-sw.com/pki/GPG-KEY-bellsoft | apt-key
    add -
RUN echo "deb [arch=amd64] https://apt.bell-sw.com/ stable main" > /etc/apt/
    sources.list.d/bellsoft.list
RUN apt update
RUN apt install -y bellsoft-java8

ENV JAVA_HOME=/usr/lib/jvm/bellsoft-java8-amd64

# apt packages
RUN apt update && apt install -y \
    apt-utils \
    curl \
    dnsutils \
    libaflib0v5 \
    libaflib-dev \
    libboost-all-dev \
    libboost-dev \
    libc3p0-java \
    libewf2 \
    libewf-dev \
    libpostgresql-jdbc-java \
    libpq5 \
    libsqlite3-dev \
    libvhdi \
    libvhdi-dev \
    libvmdk1 \
    libvmdk-dev \
    openjfx \
    testdisk \
    unzip \
    xauth \
    x11-apps \
    x11-utils \
    x11proto-core-dev \
    x11proto-dev \
    xkb-data \
    xorg-sgml-doctools \

```

```

xtrans-dev \
libcanberra-gtk-module \
squashfs-tools \
git \
make \
openjdk-17-jdk openjdk-17-jre \
build-essential autoconf libtool automake git zip wget ant \
libde265-dev libheif-dev \
libpq-dev \
testdisk libaflib-dev libewf-dev libvhdi-dev libvmdk-dev \
libgstreamer1.0-0 gstreamer1.0-plugins-base gstreamer1.0-plugins-good
    gstreamer1.0-plugins-bad \
gstreamer1.0-plugins-ugly gstreamer1.0-libav gstreamer1.0-tools gstreamer1
.0-x \
gstreamer1.0-alsa gstreamer1.0-gl gstreamer1.0-gtk3 gstreamer1.0-qt5
    gstreamer1.0-pulseaudio \
&& rm -rf /var/lib/apt/lists/*

# create directory for Dataset
RUN mkdir -p /case

# install autopsy
RUN mkdir -p /opt \
    && cd /opt \
    && curl -L https://github.com/sleuthkit/autopsy/releases/download/autopsy
        -4.21.0/autopsy-4.21.0.zip > autopsy-4.21.0.zip \
    && unzip autopsy-4.21.0.zip \
    && rm autopsy-4.21.0.zip \
    && cd /opt \
    && curl -L https://github.com/sleuthkit/sleuthkit/releases/download/
        sleuthkit-4.12.1/sleuthkit-java_4.12.1-1_amd64.deb > sleuthkit-java_4
        .12.1-1_amd64.deb \
    && dpkg -i sleuthkit-java_4.12.1-1_amd64.deb \
        || apt-get install -fy \
    && cd /opt/autopsy-4.21.0/ \
    && chmod 744 unix_setup.sh \
    && ./unix_setup.sh

# install foremost
RUN cd /opt \
    && curl -L http://foremost.sourceforge.net/pkg/foremost-1.5.7.tar.gz >
        foremost-1.5.7.tar.gz \
    && tar -xf foremost-1.5.7.tar.gz && cd /opt/foremost-1.5.7 && make &&
        make install \
    && cd /opt \
    && rm foremost-1.5.7.tar.gz

```

```
# install scalpel
RUN apt update && apt install -y scalpel

# install bulk_extractor
RUN cd /opt \
    && git clone --recurse-submodules https://github.com/simsong/
    bulk_extractor.git
RUN apt-get update && apt-get install sudo
RUN cd /opt/bulk_extractor/etc \
    && bash CONFIGURE_UBUNTU20LTS.bash \
    && cd .. \
    && ./bootstrap.sh && ./configure && make && make install
```

Code A.1: Dockerfile für File Carver

```
FROM ubuntu:20.04
```

```
ARG DEBIAN_FRONTEND=noninteractive
```

```
RUN apt-get update && apt-get install -y \  
    apt-utils \  
    curl \  
        dnsutils \  
    libafflib0v5 \  
    libafflib-dev \  
    libboost-all-dev \  
    libboost-dev \  
    libc3p0-java \  
    libewf2 \  
    libewf-dev \  
    libpostgresql-jdbc-java \  
    libpq5 \  
    libsqlite3-dev \  
    libvhdi1 \  
    libvhdi-dev \  
    libvmdk1 \  
    libvmdk-dev \  
    openjfx \  
    testdisk \  
    unzip \  
    xauth \  
    x11-apps \  
    x11-utils \  
    x11proto-core-dev \  
    x11proto-dev \  
    xkb-data \  
    xorg-sgml-doctools \  
    xtrans-dev \  
    libcanberra-gtk-module \  
    squashfs-tools \  
    git \  
    make \  
    gcc \  
    libbz2-dev \  
    libclang-dev \  
    python3 python3-dev python3-pip python3-setuptools \  
    liblinear-tools liblinear-dev python3-liblinear \  
    libtool libtool-bin libtool-doc \  
    && rm -rf /var/lib/apt/lists/*
```

```
# create directory for Dataset
```

```
RUN mkdir -p /case

# install sceadan
RUN mkdir -p /opt \
    && cd /opt \
        && git clone --recursive https://github.com/UTSA-cyber/sceadan.git \
        && cd sceadan \
        && autoreconf --force --install \
        && sh bootstrap.sh && ./configure && make
```

Code A.2: Dockerfile für Sceadan

# Literatur

- [1] o. V. *CARVE-DL – Künstliche Intelligenz in der Kriminalitätsbekämpfung*. Kaiserslautern, 15.12.2022. URL: <https://www.dfki.de/web/news/carve-dl-kuenstliche-intelligenz-in-der-kriminalitaetsbekaempfung>.
- [2] Kevin Gomez Buquerin und Hans-Joachim Hof. „Digital Forensics Investigation of the Tesla Autopilot File System“. In: *SECURWARE 2022*. Hrsg. von George O. M. Yee. Wilmington, DE, USA: IARIA, 2022, S. 82–87. ISBN: 9781685580070.
- [3] A. Pal und N. Memon. „The evolution of file carving“. In: *IEEE Signal Processing Magazine* 26.2 (2009), S. 59–71. ISSN: 1053-5888. DOI: 10.1109/MSP.2008.931081.
- [4] Kevin Klaus Gomez Buquerin, Christopher Corbett und Hans-Joachim Hof. *Structured methodology and survey to evaluate data completeness in automotive digital forensics*. Ruhr-Universität Bochum, 2021. DOI: 10.13154/294-8351.
- [5] Xiaodong Lin. „File Carving“. In: *Introductory Computer Forensics*. Hrsg. von Xiaodong Lin. Cham: Springer International Publishing, 2018, S. 211–233. ISBN: 978-3-030-00580-1. DOI: 10.1007/978-3-030-00581-8{\textunderscore}9.
- [6] Thomas Laurenson. „Performance Analysis of File Carving Tools“. In: *Security and Privacy Protection in Information Processing Systems*. Hrsg. von Lech J. Janczewski, Henry B. Wolfe und Sujeet Sheno. Bd. 405. IFIP advances in information and communication technology. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, S. 419–433. ISBN: 978-3-642-39217-7. DOI: 10.1007/978-3-642-39218-4{\textunderscore}31.
- [7] Raj Kumar Pahade, Bhupendra Singh und Upasna Singh. „A Survey on Multimedia File Carving“. In: *International Journal of Computer Science & Engineering Survey* 6.6 (2015), S. 27–46. ISSN: 09763252. DOI: 10.5121/ijcses.2015.6603.
- [8] Timothy Correjou und Simson L. Garfinkel. *A comparative analysis of file carving software*. Hrsg. von Karl A. van Bibber. Washington, DC, 2011. URL: <https://apps.dtic.mil/sti/citations/ADA550119>.

- [9] Richard Golden und Roussev Vassil. „Scalpel: A Frugal, High Performance File Carver“. In: *The Digital Forensics Research Conference*. Bd. DFRWS USA 2005. URL: [https://dfrws.org/sites/default/files/session-files/2005\\_USA\\_paper-scalpel\\_-\\_a\\_frugal\\_high\\_performance\\_file\\_carver.pdf](https://dfrws.org/sites/default/files/session-files/2005_USA_paper-scalpel_-_a_frugal_high_performance_file_carver.pdf).
- [10] AccessData Group, Inc. *Imager User Guide*. 2021. URL: [https://d1kpmuw7gvu1i.cloudfront.net/Imager/4\\_7\\_1/FTKImager\\_UserGuide.pdf](https://d1kpmuw7gvu1i.cloudfront.net/Imager/4_7_1/FTKImager_UserGuide.pdf).
- [11] o. V. *Autopsy Forensic Browser Developer's Guide and API Reference*. o. D. URL: <https://sleuthkit.org/autopsy/docs/api-docs/4.21.0/index.html>.
- [12] o. V. *Autopsy User Documentation 4.21.0: Graphical digital forensics platform for The Sleuth Kit and other tools*. o. D. URL: <https://sleuthkit.org/autopsy/docs/user-docs/4.21.0//index.html>.
- [13] Simson L. Garfinkel. „Digital media triage with bulk data analysis and bulk\_extractor“. In: *Computers & Security* 32 (2013), S. 56–72. ISSN: 01674048. DOI: 10.1016/j.cose.2012.09.011.
- [14] Joachim Sester u. a. „A comparative study of support vector machine and neural networks for file type identification using n-gram analysis“. In: *Forensic Science International: Digital Investigation* 36 (2021), S. 301121. ISSN: 26662817. DOI: 10.1016/j.fsidi.2021.301121.
- [15] Leonid Berlyand und Pierre-Emmanuel Jabin. *Mathematics of deep learning: An introduction*. De Gruyter graduate. Berlin und Boston: De Gruyter, 2023. ISBN: 9783111025551. URL: <https://www.degruyter.com/isbn/9783111024318>.
- [16] Andrew Duffy. *CarveML : application of machine learning to file fragment classification*. 2014. URL: <https://cs229.stanford.edu/proj2014/Andrew%20Duffy,%20CarveML%20an%20application%20of%20machine%20learning%20to%20file%20fragment%20classification.pdf>.
- [17] ANNICK LESNE. „Shannon entropy: a rigorous notion at the crossroads between probability, information theory, dynamical systems and statistical physics“. In: *Mathematical Structures in Computer Science* 24.3 (2014). ISSN: 0960-1295. DOI: 10.1017/S0960129512000783.
- [18] Nicole L. Beebe u. a. „Sceadan: Using Concatenated N-Gram Vectors for Improved File and Data Type Classification“. In: *IEEE Transactions on Information Forensics and Security* 8.9 (2013), S. 1519–1530. ISSN: 1556-6013. DOI: 10.1109/TIFS.2013.2274728.

- [19] Nico Vinzenz und Tobias Eggendorfer. „Forensic Investigations in Vehicle Data Stores“. In: *Proceedings of the Third Central European Cybersecurity Conference*. New York, NY, USA: ACM, 2019, S. 1–6. ISBN: 9781450372961. DOI: 10.1145/3360664.3360665.
- [20] Y. Shafranovich. *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. 2005. DOI: 10.17487/RFC4180.
- [21] o. V. *[MS-XLS]: Excel Binary File Format (.xls) Structure*. Mai 2023. URL: <https://msopenspecs.azureedge.net/files/MS-XLS/%5bMS-XLS%5d.pdf>.
- [22] o. V. *Microsoft Office Excel 97-2003 Binary File Format (.xls, BIFF8)*. 2019. URL: <https://www.loc.gov/preservation/digital/formats/fdd/fdd000510.shtml>.
- [23] Tim Bray u. a. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. November 2008. URL: <https://www.w3.org/TR/xml/>.
- [24] o. V. *GPX 1.1 Schema Documentation*. o. D. URL: <https://www.topografix.com/GPX/1/1/>.
- [25] o. V. *The Bosch CDR Tool: Key Applications for Insurance and Law Enforcement Crash Investigators*. o. D. URL: <https://crashdatagroup.com/pages/the-bosch-cdr-tool>.
- [26] Comma.ai. *opendbc*. o. D. URL: <https://github.com/commaai/opendbc>.
- [27] Vector Informatik GmbH. *DBC File Format Documentation: Version 1.0.5*. 2010. URL: [http://mcu.so/Microcontroller/Automotive/dbc-file-format-documentation\\_compress.pdf](http://mcu.so/Microcontroller/Automotive/dbc-file-format-documentation_compress.pdf).
- [28] NHTSA. *NHTSA CDR Files*. URL: <https://www.nhtsa.gov/file-downloads?p=nhtsa/downloads/CISS/CDR\%20Files/>.
- [29] Jeferson Menegazzo. *PVS - Passive Vehicular Sensors Datasets*. 2021. DOI: 10.34740/kaggle/ds/1105310.
- [30] Uche Onyekpe u. a. *IO-VNBD: (Inertial Odometry Vehicle Navigation Benchmark Dataset)*. 2021. URL: <https://github.com/onyekpeu/IO-VNBD>.
- [31] Susana Cruz und Ana Aguiar. *Road Vehicle Localization*. 2020. DOI: 10.21227/92ya-v240.
- [32] o. V. *Squashfs 4.0 Filesystem*. o. D. URL: <https://docs.kernel.org/filesystems/squashfs.html>.
- [33] Lea Achter. *Bachelor\_KI*. 2024. URL: [https://github.com/AchterL/Bachelor\\_KI](https://github.com/AchterL/Bachelor_KI).



- [34] Nigel Poulton. *Docker deep dive: Zero to docker in a single book*. 2023 edition. Birmingham: Packt Publishing Limited, 2023. ISBN: 978-1-83508-791-6. URL: <https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=30662233>.
- [35] Amit M. Potdar u. a. „Performance Evaluation of Docker Container and Virtual Machine“. In: *Procedia Computer Science* 171 (2020), S. 1419–1428. ISSN: 18770509. DOI: 10.1016/j.procs.2020.04.152.
- [36] Hanwen Zhang. *An Overview of Virtual Machine v.s. Docker v.s. Kubernetes*. Hrsg. von Medium. 2023. URL: <https://hanwenzhang123.medium.com/docker-vs-virtual-machine-vs-kubernetes-overview-389db7de7618>.
- [37] Kim Strandberg, Nasser Nowdehi und Tomas Olovsson. „A Systematic Literature Review on Automotive Digital Forensics: Challenges, Technical Solutions and Data Collection“. In: *IEEE Transactions on Intelligent Vehicles* 8.2 (2023), S. 1350–1367. ISSN: 2379-8858. DOI: 10.1109/TIV.2022.3188340.
- [38] Željko Đ. Vujovic. „Classification Model Evaluation Metrics“. In: *International Journal of Advanced Computer Science and Applications* 12.6 (2021). ISSN: 2158107X. DOI: 10.14569/IJACSA.2021.0120670.
- [39] Margherita Grandini, Enrico Bagli und Giorgio Visani. *Metrics for Multi-Class Classification: an Overview*. 2020. URL: <http://arxiv.org/pdf/2008.05756v1>.
- [40] N. Freed und N. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. 1996. DOI: 10.17487/RFC2046. URL: <https://www.ietf.org/rfc/rfc2046.txt>.