HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

MASTERARBEIT

B. Sc.
Kira Marie Kähler

Konzeption, prototypische Implementierung und Evaluierung eines intelligenten Carving Algorithmus zur Rekonstruktion fragmentierter JPEG-Dateien

2022

Fakultät **Angewandte Computer- und Biowissenschaften**

MASTERARBEIT

Konzeption, prototypische Implementierung und Evaluierung eines intelligenten Carving Algorithmus zur Rekonstruktion fragmentierter JPEG-Dateien

Autor:
Kira Marie Kähler

Studiengang:
Cybercrime/ Cybersecurity

Seminargruppe:
CY19wC-M

Erstprüfer:
Prof. Ronny Bodach

Zweitprüfer:
M. Sc. Stefan Schildbach

Mittweida, 2022

Bibliografische Angaben

Kähler, Kira Marie: Konzeption, prototypische Implementierung und Evaluierung eines intelligenten Carving Algorithmus zur Rekonstruktion fragmentierter JPEG-Dateien, 109 Seiten, 56 Abbildungen, Hochschule Mittweida, University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften

Masterarbeit, 2022

Referat

Der technologische Fortschritt ermöglicht das Speichern von größeren Datenmengen. Dies hat zur Folge, dass Daten mehr Platz auf einem Datenträger einnehmen und die Wahrscheinlichkeit des Aufteilens einer Datei auf mehrere auf dem Datenträger verteilte Positionen steigt. Von dieser sogenannten Fragmentierung sind auch JPEG-Dateien betroffen, wobei sich die Frage stellt, wie ein Zusammensetzen der Fragmente ohne die notwendigen Informationen aus dem Dateisystem möglich ist.
Ziel dieser Masterarbeit ist es, eine prototypische Implementierung eines intelligenten Carving Algorithmus zur Rekonstruktion fragmentierter JPEG-Dateien zu entwickeln, welcher durch erzeugte Testszenarien evaluiert wird.
Um dieses Ziel zu erreichen, wird ein Programm erstellt, welches sich an dem Smart Carving-Prinzip orientiert und diverse Carving-Methoden einbezieht. Für die anschließende Beurteilung der Lauffähigkeit des Prototyps werden Szenarien zum Testen von Stärken und Schwächen entwickelt. Die Ergebnisse aus diesen werden durch eine Evaluationsstrategie bewertet.
Anhand der Ergebnisse wird deutlich, dass der entwickelte Prototyp noch viele Schwächen aufweist. Bei manchen Szenarien können vollständig korrekte Ergebnisse geliefert werden und die JPEG-Dateien rekonstruiert werden, bei anderen Szenarien sind die Ergebnisse unzureichend. Diese Ergebnisse und der Fakt, dass der Prototyp lediglich für kleine Datenmengen konzipiert und erprobt wurde, zeigen, dass ein Einsatz des Programms in einer realen Umgebung noch nicht möglich ist.

I. Inhaltsverzeichnis

II. Abbildungsverzeichnis

III. Tabellenverzeichnis

IV. Abkürzungsverzeichnis

AC	Alternating Current
APP	Application
ASCII	American Standard Code for Information Interchange
CED	Coherence of Euclidian Distance
COM	Comment
DC	Direct Current
DCT	Discrete Cosine Transform
DEV	Devaluation
DHT	Define-Huffman-Table
DQT	Define-Quantization-Table
DRI	Define-Restart-Interval
ED	Euclidian Distance
EOI	End-Of-Image
EXIF	Exchangeable Image File Format
fn	Falsch negativ
fp	Falsch positiv
HDD	Hard Disk Drive
ID	Identifikationsnummer
IJG	Independent JPEG Group
JFIF	JPEG File Interchange Format
JNG	JPEG Network Graphics
JPEG	Joint Photographic Experts Group
MS-DOS	Microsoft Disk Operating System
NTFS	New Technology File System
PIL	Python Imaging Library
PNG	Portable Network Graphics
PUP	Parallel Unique Path
rn	Richtig negativ
rp	Richtig positiv
RST	Restart
SHT-PUP	Sequential Hypothesis Test-Parallel Unique Path
SoD	Sum Of Differences

SOF ..... Start-Of-Frame

SOI ..... Start-Of-Image

SOS ..... Start-Of-Scan

SPF ..... Shortest Path First

SPIFF ..... Still Picture Interchange File Format

SSD ..... Solid State Drive

SSHD ..... Solid State Hybrid Drive

1 Einleitung

Das aktuelle Bundeslagebild des Bundeskriminalamtes im Bereich Cybercrime 2021 zeigt einen drastischen Anstieg der Cyberkriminalität in Deutschland von über zwölf Prozent im Vergleich zum Vorjahr. Damit haben die Delikte in diesem Bereich ihren Höchstwert mit 146.363 erreicht. [1] Bereits in den Jahren zuvor konnte ein Anstieg der Straftaten in diesem Bereich verzeichnet und eine steigende Relevanz der Cyberkriminalität festgestellt werden [2]. Daraus lässt sich ableiten, dass Cyberkriminalität durch ihre stetige und rasante Zunahme eine nicht mehr zu unterschätzende Gefahr darstellt.

Im allgemeinen Sprachgebrauch wird der Begriff Cyberkriminalität mit Hacking oder grundlegend mit Straftaten im Internet gleichgesetzt. Doch hierbei handelt es sich lediglich um einen kleinen Teilbereich, wie die Definition des Bundesministeriums des Inneren und für Heimat zeigt. Dort wird Cyberkriminalität definiert als „Straftaten, bei denen die Täter moderne Informationstechnik nutzen“, was Straftaten außerhalb des Internets miteinbezieht [3]. Relevant, um unter den Begriff Cyberkriminalität zu fallen, ist somit die Nutzung von Informationstechnik bei Straftaten, nicht das Begehen von Straftaten an der oder durch die Informationstechnik. Diese erwähnte Informationstechnik wird beeinflusst durch die stetige technische Entwicklung und Verbesserung von Geräten, die zu Herausforderungen für die Ermittlungsbehörden führt. Beispielsweise werden Datenmenge immer größer und die Anzahl der Geräte, die unter den Begriff Informationstechnik fallen, steigt. Dies führt dazu, dass die Aufgabenbereiche der Aufklärungsbehörden wachsen und an Relevanz zunehmen. Dadurch geraten Ermittler an zeitliche Grenzen, wodurch sie auf Hilfsmittel zur Vereinfachung ihrer Arbeit zurückgreifen müssen.

Legt man seinen Fokus zum Beispiel auf die Auswertung von Geräten wie Festplatten, die der Informationstechnik zugehörig sind, kommen Programme zur automatisierten Analyse zum Einsatz. Diese forensischen Werkzeuge dienen dazu, Daten aufzubereiten und Daten zu analysieren, was eine enorme Zeitersparnis zur Folge hat. Allerdings sind diese Werkzeuge noch nicht für alle Probleme und für jeden Fall einsetzbar, da sie noch Grenzen aufweisen. Einige Probleme können durch das Fehlen bestimmter Informationen, zum Beispiel des Dateisystems, entstehen. Ist dieses beschädigt oder fehlt es sogar gänzlich, kann das Auswirkungen auf das Ergebnis der Analyse durch forensische Werkzeuge haben. Diese greifen in der Regel auf Informationen aus dem Dateisystem zurück, um die Positionen, die Größe oder anderweitige Informationen von Dateien zu ermitteln. Fehlt das Dateisystem, muss über spezielle Kennungen für Anfang und Ende einer Datei nach den diversen Dateien gefiltert werden. Sind diese jedoch fragmentiert, was bei großen Dateien und Datenmengen zu erwarten ist, ist eine Rekonstruktion durch die gebräuchlichen Softwareanwendungen nicht möglich. Diese

Dateien würden in solchen Fällen gar nicht oder schlecht rekonstruiert werden, was besonders bei belastendem Material große Auswirkungen haben könnte. Dabei kann es sich neben wichtigen Dokumenten, relevanten E-Mail-Verläufen auch um Bilder handeln, die tatrelevante Szenen abbilden. Wenn diese Bilder nicht rekonstruiert werden können, kann es dazu führen, dass dem Ermittler wichtige Informationen verborgen bleiben.
Besonders im Hinblick auf die Kinderpornographie ist es jedoch wichtig, dass alle Bilder mit diesen Beweisen gesichert werden können, da jedes Bild relevant für ein mögliches Strafmaß sein kann. Des Weiteren ist es wichtig, die Opfer der Bilder zu identifizieren, um sie aus ihrer prekären Situation zu befreien. Findet ein gängiges, von den Ermittlern verwendetes Carving-Werkzeug Bilder mit tatrelevantem Beweismaterial aufgrund ihrer Fragmentierung nicht, kann dies den Ausgang eines möglichen Strafverfahrens beeinflussen.

Diese Aspekte sind Motivation für die vorliegende Arbeit. Es soll eine Konzeption und prototypische Implementierung eines intelligenten Carving Algorithmus erfolgen. Da eine Auseinandersetzung mit der Rekonstruktion aller möglichen Dateitypen einen im Rahmen dieser Arbeit nicht zu leistenden Aufwand darstellen würde, wurde der Fokus auf den Umgang mit fragmentierten JPEG-Bilddateien ohne Verweis im Dateisystem gelegt. Die Begrenzung auf Bilddateien ist mit dem Beispiel der Kinderpornographie begründet, bei welcher jedes einzelne Bild eine besondere Relevanz aufweist.
JPEG-Dateien, welche von gängigen forensischen Werkzeugen aufgrund ihrer Fragmentierung nicht rekonstruiert werden konnten, sollen mit Hilfe eines Algorithmus innerhalb eines Prototyps, welcher in der vorliegenden Arbeit erstellt wird, rekonstruiert werden können. Dies soll den Ermittlern bei der Fallarbeit Unterstützung bieten. Um zu überprüfen, inwieweit sich der entwickelte Prototyp bereits für den realen Gebrauch eignet und an welchen Stellen noch Überarbeitungsbedarf besteht, wird zusätzlich eine Evaluierung mit Testdatensätzen durchgeführt.

1.1 Problemstellung

Die rasante Zunahme von Delikten, die unter die Kategorie Cyberkriminalität fallen und das Wachstum der Nutzung von informationstechnischen Medien führen dazu, dass auch die Anforderungen an Ermittlungsbehörden stetig ansteigen. Deren Ermittler müssen in der Lage sein, in kurzer Zeit riesige Datenmenge unterschiedlicher Art zu analysieren und auszuwerten. Um dies zu gewährleisten, muss auf forensische Werkzeuge zurückgegriffen werden, die einen Teil der Arbeit durch Automatisierung abnehmen. Allerdings haben auch diese Werkzeuge Grenzen, die beispielsweise in der Fragmentierung von Daten liegen, zu welchen keine Informationen im Dateisystem zu finden sind [4] [5]. Durch immer weiterwachsende Datenmengen, nimmt die Häufigkeit der fragmentierten Daten auf einem Datenträger zu, wodurch der Umgang mit solchen Fragmenten in der Zukunft unumgänglich ist.

Die wachsende Größe von Dateien ist auch bei Bilddateien durch immer besser werdende Kameras und Steigerung der Auflösung erkennbar. Aus diesem Grund ist auch bei diesen Dateien mit dem zunehmenden Vorkommen von Fragmentierung zu rechnen, mit welchem die bisherigen forensischen Werkzeuge ohne Metadaten aus dem Dateisystem nicht umgehen können.
Grund hierfür ist das Prinzip, auf welchem die meisten Werkzeuge beruhen. Dabei werden Dateien, welche beispielsweise durch Löschen keinen Eintrag im Dateisystem haben, anhand einer Kennung gesucht. Diese Kennungen sind bei vielen Dateien sowohl am Beginn, als auch am Ende zu finden, was sich die Algorithmen der Werkzeuge zunutze machen. Diese filtern nach den Kennungen vom Anfang und Ende der Datei und extrahieren alle Daten dazwischen. Problematisch wird dieses Prinzip, sobald die Datei nicht an einem Stück vorliegt, sondern fragmentiert und durch eine andere Datei unterbrochen ist. Es kann nicht gefiltert werden, welche Bereiche zu dieser Datei gehören, welche zu einer anderen und an welcher Position die Datei weitergeht. Die fragmentierten Dateien können daher nicht wiederhergestellt werden, was ein Problem darstellt, sofern die Datei für einen vorliegenden Fall relevant ist.

1.2 Zielsetzung

Im Verlauf der vorliegenden Arbeit soll ein Ansatz zur Rekonstruktion von fragmentierten JPEG-Dateien entwickelt und in einer prototypischen Implementierung umgesetzt werden. Dieser Ansatz soll ohne das Zugreifen auf Metadaten des Dateisystems auskommen. In Bezug auf fragmentierte Bilddateien bedeutet dies, dass sowohl die Anzahl und Länge der Bilddateien, als auch die Häufigkeit der Fragmentierung unbekannt sind. Der entwickelte Prototyp soll diese Informationen eigenständig generieren und anschließend die Dateien, unabhängig von der Anzahl der Fragmente, rekonstruieren. Zu diesem Zweck soll das entwickelte Programm in der Lage sein, Cluster eines Datenträgerabbildes in JPEG-Cluster und nicht JPEG-Cluster aufzuteilen.
Anschließend sollen die Bildfragmente anhand ihrer Zugehörigkeit sortiert und im weiteren Verlauf zu einem Gesamtbild zusammengesetzt werden. Die korrekte Anordnung der Bildfragmente soll bei diesem Prozess ebenfalls Beachtung finden. Dieses Vorgehenbsoll automatisiert und ohne menschliches Eingreifen oder Bewerten erfolgen, allerdings um Funktionen erweiterbar sein. Zusätzlich dazu soll durch den Anwender eine Auswahl der zurückgelieferten Informationen getroffen werden können.

Für die Evaluierung sollen Speicherabbildes erzeugt werden, welche eine Vielzahl an Szenarien abdecken. Des Weiteren erfolgt durch eine eigens entwickelte Strategie eine Evaluierung der erhaltenen Ergebnisse und der Leistungsfähigkeit des erzeugten Programms. Darüber hinaus soll das Programm auf seine Schwächen anhand diverser Eingabedaten getestet und Grenzen eines solchen Verfahrens ermittelt werden.

Ziel dieser Arbeit ist es nicht, ein bereits vollständig lauffähiges Programm zu entwickeln, welches mit diversen Dateien umgehen kann. Es soll lediglich ein Ansatz in einem Prototyp umgesetzt werden, der Grundlage für zukünftige Erweiterungen sein kann. Dabei sollen zu Beginn nur kleine beispielhafte Datenmengen ausgewertet und erprobt werden, welche sich auf beispielhafte Bilder des Datentyps JFIF reduzieren. Des Weiteren wird nur mit Bilddateien gearbeitet, bei denen alle Fragmente vollständig vorhanden sind und die Bilddaten verschiedener Bilder klar voneinander abgegrenzt sind.

1.3 Methodische Vorgehensweise

Die vorliegende Arbeit beginnt mit dem Kapitel *Grundlagen*, welches ein Grundwissen über alle, für das Verständnis notwendigen Themengebiete, verschafft. Zu diesem Zweck wird mit der Auseinandersetzung mit Datenträgern und diversen Festplattentypen begonnen. Dies soll dazu beitragen, dass Zusammenhänge zum Speicherprinzip und dem Entstehen von Fragmentierung erkennbar werden.
Anschließend folgt der Themenkomplex um das JPEG-Kompressionsverfahren, wobei im Besonderen auf die Schritte der JPEG-Kodierung und die Marker und Segmente eingegangen wird. Diese Informationen haben für den weiteren Verlauf der Arbeit eine große Relevanz. Ebenfalls wichtig für das weitere Verständnis und den weiteren Verlauf ist das Verstehen des Carvings. Hierauf wird in den Grundlagen zuletzt eingegangen, indem diverse Methoden und Herangehensweisen des Carvings vorgestellt werden.

Nachdem alle relevanten Grundlagen erläutert sind, folgt das Kapitel *Methodik*. Zu Beginn dieses Kapitels wird der Hauptthemenkomplex der Arbeit, die Entwicklung des prototypischen Carving-Algorithmus, beschrieben. Hierbei werden die Grundstruktur und das Prinzip des Verfahrens zur Wiederherstellung fragmentierter JPEG-Dateien erläutert. Im Anschluss daran folgt eine Erklärung des Umgangs mit dem entwickelten Programm und dessen damit verbundenen Grenzen. Damit das Programm evaluiert werden kann, wird die Entwicklung und der Aufbau von diversen Testszenarien anschließend beschrieben. Im Zusammenhang damit folgt die Beschreibung eines Evaluierungsalgorithmus, mit welchem Ergebnisse aus den Testszenarien einheitlich beurteilt werden können.

Im vierten Kapitel, den *Ergebnissen*, finden die zuvor entwickelten Speicherabbilder, der Proto- typ und die Evaluierungsstrategie Verwendung. Anhand der Analyse der einzelnen Szenarien durch den Prototyp erhält man Ergebnisse, die mit Hilfe der Evaluationsstrategien dargelegt werden. Eine Auseinandersetzung mit diesen Ergebnissen und die Begründung für bestimmte Ausgänge erfolgt im Kapitel ‚Diskussion‘. Im Anschluss daran wird alles noch einmal im *Fazit* zusammengefasst und mögliche Verbesserungs- und Erweiterungsmöglichkeiten im *Ausblick* festgehalten.

2 Grundlagen

Im folgenden Kapitel werden die Grundlagen geklärt, welche für das weitere Verständnis der Arbeit relevant und unabdingbar sind.
Zu Beginn wird eine grobe Übersicht über die verschiedenen Festplattentypen gegeben, da diese die Grundlage für das Prinzip, wie Daten gespeichert werden, beeinflusst haben. Auf dieses Prinzip der Speicherung von Daten und der damit verbundenen Entstehung von Fragmentierung wird nachfolgend eingegangen. Im Anschluss wird im Besonderen auf das JPEG-Verfahren eingegangen, welches die Grundlage für die gesamte Arbeit ist. Abschließend werden Carving-Methoden vorgestellt, welche in der Grundstruktur des Prototyps Verwendung finden.

2.1 Datenträger

Zum Verständnis für den weiteren Verlauf ist zu Beginn wichtig zu verstehen, was für Festplattentypen existieren und worauf die Entstehung bestimmter Begriffe wie *Cluster* zurückzuführen sind. Des Weiteren beeinflusst die Wahl der Festplattentypen die Häufigkeit von Fragmentierungen. Anschließend wird auf das Speicherprinzip und die Verwaltung des Speicherplatzes eingegangen.

2.1.1 Festplattentypen

Das Funktionsprinzip ist abhängig von der Art der vorliegenden Festplatte. Der unterschiedliche Aufbau ist dafür verantwortlich, wie stark Dateien auf einer Festplatte fragmentiert werden. Es wird insgesamt zwischen vier Festplattentypen unterschieden. Diese sind mechanische, Halbleiter-, Hybrid- und virtuelle Festplatten [6].

- **Mechanische Festplatten**
  Unter einer mechanischen Festplatte versteht man eine, wie der Name schon sagt, mechanisch angetriebene Festplatte. Dieser Festplattentyp ist auch unter den Namen *Magnetische Festplatte*, *Hard Disk Drive* oder kurz *HDD* bekannt [7]. Sie besteht aus mehreren übereinanderliegenden Magnetplatten, die durch einen Motor angetrieben werden. Über den Platten schwebt ein Schreib- und Lesearm, an dem sich Schreib- und Leseköpfe befinden. Ein Mechanismus steuert diesen Arm zu Stellen auf der Scheibe, um sie an dieser Stelle zu lesen oder zu beschreiben, siehe Abbildung 2.2. [8]
  Die einzelnen Magnetplatten sind in diverse Bereiche unterteilt. In Abbildung 2.1 ist eine Übersicht der Unterteilung einer magnetisierten Scheibe schemenhaft dargestellt. Daten werden auf kreisförmigen Spuren gespeichert, welche in kleinere Abschnitte unterteilt sind. Diese Teilbereiche heißen Sektoren. Aufgrund der

kreisrunden Form der Drehscheibe, sind die Sektoren am äußeren Rand größer als die im Inneren. [6]
Eine Zusammenfassung mehrerer Sektoren auf einer Spur heißt Cluster. Das Gruppieren zu einem Cluster erfolgt über das Dateisystem. Werden mehrere parallel verlaufende Sektoren unterschiedlicher Spuren verknüpft, spricht man von einem Block.

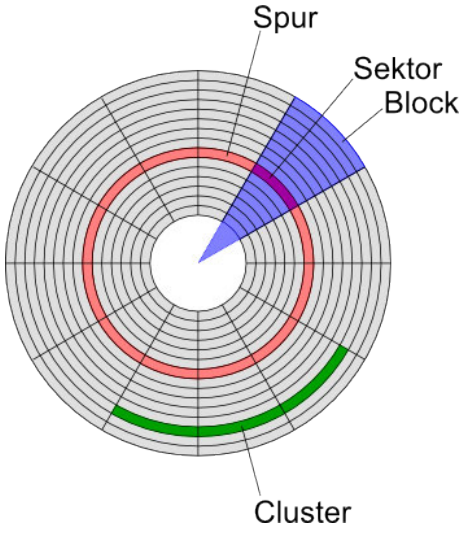


Abbildung 2.1: Aufteilung der Bereiche einer Magnetscheibe von mechanischen Festplatte [9]

- **Halbleiterfestplatte**
  Eine Halbleiterfestplatte ist eine robustere und schnellere Variante der mechanischen Festplatte [10]. Man kennt sie auch unter den Begriffen *Solid State Drive*, *Solid State Disk* oder kurz *SSD* [11]. Im Gegensatz zu ihrem Vorgängermodell, der HDD, besitzt sie keine mechanischen Bestandteile mehr, sondern besteht ausschließlich aus Halbleitern [6].
  Für die Speicherung werden Flash-Speicherzellen, ähnlich wie bei USB-Sticks, verwendet. Diese haben eine begrenzte Lebensdauer. Ein Controller steuert die Schreib- und Lesevorgänge der Speicherzellen. [12]
  In Abbildung 2.2 ist eine Festplatte dieser Art skizziert.
  Mit Hilfe des sogenannten *Wear-Leveling*-Verfahrens soll die Lebensdauer der Speicherzellen erhalten bleiben. Hierzu reguliert der Controller durch das Verfahren ein übermäßiges Schreiben auf einen einzelnen Bereich und teilt Schreibvorgänge gleichmäßig auf alle Speicherbereiche auf. [6]

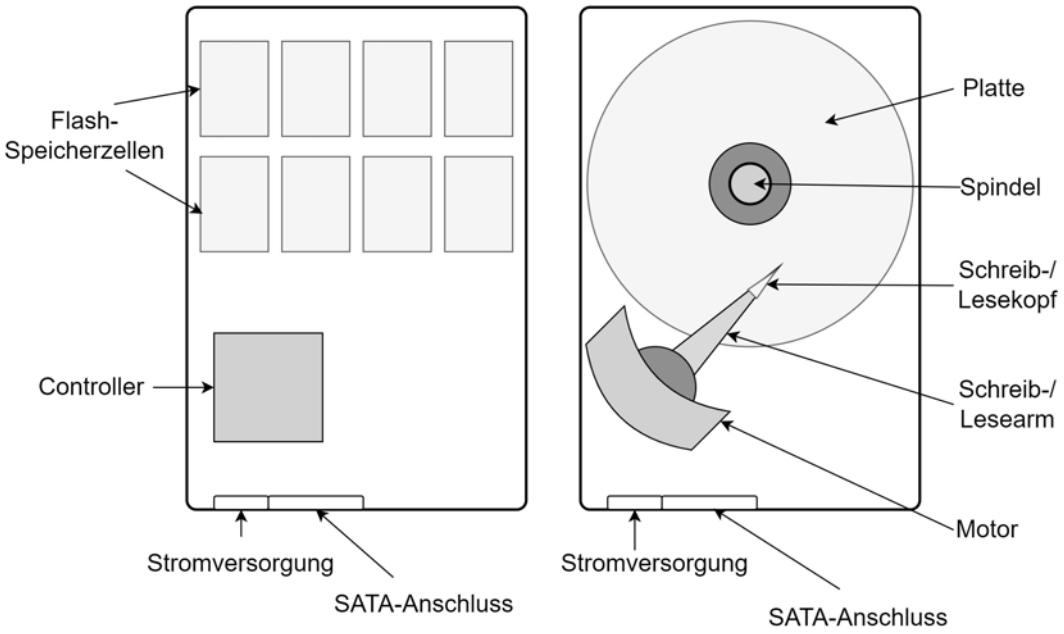


Abbildung 2.2: Gegenüberstellung des Aufbaus einer HDD und SSD nach D. Janßen [13]

- **Hybridfestplatte**
  Als Hybridfestplatte oder *Solid State Hybrid Drive* kurz *SSHD* wird eine Festplattenart bezeichnet, welche eine Kombination der mechanischen und Halbleiterfestplatte darstellt. Durch die Kombination sollen die Vorteile der hohen Speicherkapazität bei geringen Kosten einer HDD und die Schnelligkeit beim Schreib- und Lesezugriff einer SSD miteinander vereint werden. [14]
  Eine mechanische Festplatte nimmt hierbei den größten Teil des Speichers ein. Zusätzlich dazu sind Flash-Speicherzellen verbaut, auf welche Daten geschrieben werden, die häufig aufgerufen werden. Dies hat zum Ziel, dass die Zugriffszeit verkürzt wird. [15]
  Des Weiteren können während der Laufzeit weitere Daten in dem Flash-Speicher abgelegt werden, welche bei Annäherung an die Kapazitätsgrenzen auf die mechanische Festplatte geschrieben werden. Diese Festplattenart hat keine nennenswerte Verbreitung. [6]

- **Virtuelle Festplatte**
  Eine Virtuelle Festplatte wirkt für den Benutzer wie eine physische Festplatte. Allerdings handelt es sich hierbei lediglich um eine Datei auf einer physischen Festplatte. Diese läuft innerhalb einer virtuellen Umgebung. [6]
  Sie kann ein eigenes Betriebs- und Dateisystem besitzen [16].

2.1.2 Speicherprinzip

Das Speicherprinzip eines Systems basiert auf der Grundlage, dass für das Speichern feste Größen vorgeschrieben sind. Diese Größen hängen zum Teil mit dem Grundprinzip und Aufbau von HDDs zusammen. Eine HDD ist, wie bereits erwähnt, in Sektoren unterteilt. Diese Sektoren besitzen in den meisten Fällen eine Größe von 512 Bytes [17]. Für eine Zusammenfassung mehrerer Sektoren zu einem sogenannten Cluster ist das Dateisystem zuständig [18]. Die Begrifflichkeit ist von den Betriebssystemen abhängig. Beispielsweise kann auch von Datenblock oder Block gesprochen werden. Es handelt sich hierbei jedoch um einen vom Dateisystem abhängigen Block und nicht die Blöcke, welche in Abbildung 2.1 dargestellt sind. Um für den weiteren Verlauf der Arbeit Missverständnisse zu vermeiden, wird ausschließlich von Clustern gesprochen. Ein Cluster ist insofern relevant, da es sich hierbei laut Definition um die kleinste adressierbare Einheit handelt. Wird eine Datei gespeichert, wird sie in ein entsprechendes Cluster geschrieben.Ist die Datei zu klein, um ein Cluster zu füllen, kann das Dateisystem den noch ungefüllten Bereich nicht mit anderen Daten füllen, da es den Bereich nicht adressieren kann. Für das Dateisystem gibt es somit nur leere oder gefüllte Cluster, egal wie viel Speicher in der Realität von dem Cluster wirklich belegt ist. Cluster haben in den häufigsten Fällen eine Größe von 4096 Bytes. In neueren Systemen kann die Cluster- und auch Sektorgröße jedoch größer ausfallen. [18]
Allgemein kann jedoch gesagt werden, dass die Clustergröße in der Regel ein Vielfaches von 512 Bytes ist. Mit Hilfe der Clustergröße lässt sich die Größe der Festplatte ermitteln, indem die Größe eines Clusters mit der Anzahl der Cluster multipliziert wird. [19]

2.1.3 Speicherplatzverwaltung

Das Dateisystem ist neben der Clustergrößenbestimmung auch für die Verwaltung des Speicherplatzes zuständig. Es organisiert das Ablegen und Auffinden von Daten auf der Festplatte. Hierzu werden Verweise zu Beginn und Ende der Dateien, Identifikatoren dieser Dateien und teilweise auch Metadaten gespeichert. Die Art der Speicherung und Anzahl der Informationen zu den Dateien auf einer Festplatte hängt von dem verwendeten Betriebssystem ab. [20]
Bei Unix-Dateisystemen wird beispielsweise eine Inode, welche einen Indexknoten darstellt, zu jeder angelegten Datei erzeugt. Diese Inodes werden in einer Liste gespeichert und verweisen auf das Cluster, in welchem die zugehörige Datei liegt. [21] [22]
Im Vergleich dazu wird bei MS-DOS Betriebssystemen die *File Allocation Table* verwendet, in der es zu jedem Cluster einen Eintrag gibt [23]. Auch wenn die diversen Betriebssysteme das Verwalten von Dateien auf der Festplatte unterschiedlich bewältigen, basiert der Grundsatz darauf, dass das Dateisystem die Cluster adressiert und Informationen zu Speicherplätzen in diesem zu finden sind. Auch die Positionen der Einzelteile fragmentierter Dateien sind im Dateisystem festgehalten.

2.1.4 Fragmentierung

Nachdem das Prinzip der Speicherung geklärt ist, folgt als nächstes die Erläuterung des Vorkommens einer sogenannten Fragmentierung. Der Idealfall beim Speichern von Daten ist, dass diese zusammengehörig hintereinander auf der Festplatte abgelegt werden. Ist dies aufgrund bestimmter Faktoren nicht möglich und der Inhalt einer Datei muss auf mehrere Bereiche des Datenträgers in zwei oder mehrere Teile aufgeteilt werden, spricht man von Fragmentierung [6]. In Abbildung 2.3 ist skizziert, wie sich unfragmentierte und fragmentierte Dateien unterscheiden. Hierzu wurden in (a) zuerst zwei unfragmentierte Dateien A und B und anschließend in (b) eine fragmentierte Datei A mit einer unfragmentierten Datei B dazwischen aufgezeichnet.

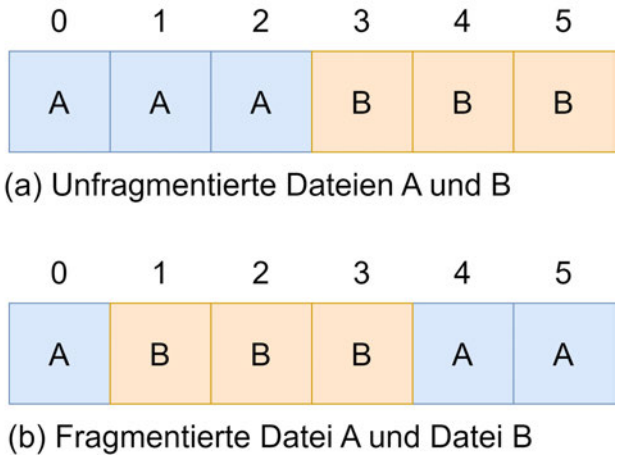


Abbildung 2.3: Skizzierte Gegenüberstellung von zwei unfragmentierten Dateien und der fragmentierten Datei A, unterbrochen von Datei B

Das Entstehen von Fragmentierung ist auf das Speicherprinzip mit den statischen Clustergrößen zurückzuführen [17]. Es kann beispielsweise zu einer Fragmentierung kommen, wenn der Inhalt einer Datei erheblich größer als ein Cluster ist. In so einem Fall muss die Datei aufgrund ihrer Größe auf mehrere Cluster aufgeteilt werden, die nicht zwangsläufig nebeneinander liegen. Dies führt zu einer Fragmentierung der Datei. [24]. Allerdings kommt es nicht nur bei großen Dateien zur Fragmentierung.
Weitere Gründe für das Fragmentieren von Dateien können auch geringer vorhandener Speicherplatz, eine nachträgliche Abänderung, Ergänzung einer Datei oder das Verwenden eines *Wear-Leveling*-Algorithmus sein. [19]
Bei gering vorhandenem Speicherplatz kann es sein, dass nur noch kleine Speicherbereiche verfügbar sind, weshalb eine neue Datei aufgrund ihrer Größe über die Festplatte auf freie Speicherbereiche verteilt gespeichert werden muss. Außerdem kann nachträgliches Bearbeiten einer Datei problematisch werden. Durch das Bearbeiten wächst die Dateigröße, wodurch diese nicht mehr in den ursprünglich vorgesehenen Speicherbereich passt. [19] [24]

Eine Fragmentierung ist an dieser Stelle unumgänglich. Das Dateisystem NTFS versucht diesem Problem vorzubeugen, indem ein kleiner Bereich hinter der Datei freigelassen wird für mögliche Dateivergrößerungen [6].
Der bereits erwähnte *Wear-Leveling*-Algorithmus trägt auch zum Fragmentieren bei. Allerdings handelt es sich hierbei um eine Fragmentierung auf physischer und nicht logischer Ebene. Das bedeutet, dass die Cluster physisch nicht nebeneinanderliegen müssen, um auf logischer Ebene benachbart zu sein. Der Fokus des *Wear-Leveling*-Algorithmus liegt nicht darauf, Dateien auf physischer Ebene möglichst nebeneinander zu speichern, sondern das Platzieren erfolgt aufgrund des Abnutzungsgrads der Speicherzellen. [19]

Für den Nutzer ist eine Fragmentierung unerheblich. Lediglich eine vergleichsweise größere Zugriffszeit als bei unfragmentierten Dateien kann auftreten. Dies ist bei HDDs der Fall. Die Zeitverzögerung entsteht dadurch, dass der Schreib- und Lesekopf bei fragmentierten Dateien über die Magnetplatte hin- und herspringen muss. Eine Defragmentierung ist bei diesem Festplattentyp zur Geschwindigkeitsgenerierung ratsam. Bei SSDs ist dies jedoch nicht ratsam, da hierbei die gleichmäßige Abnutzung der einzelnen Zellen nicht gewährleistet ist. [25]
Werden Dateien aufgerufen, die fragmentiert sind, geschieht dies über einen Metadateneintrag im Dateisystem. In diesem sind Verweise zu den Positionen der einzelnen Fragmente zu finden. [26] [27] Es können hierbei auch mehrere Fragmente vorhanden sind. Im Allgemeinen kann von drei Fragmentierungsmöglichkeiten gesprochen werden, welche in Abbildung 2.4 abgebildet sind. Bei (a) handelt es sich um eine nicht- zusammenhängende Fragmentierung, was bedeutet, dass die Fragmente in korrekter Reihenfolge durch eine Lücke voneinander getrennt sind. Die Darstellung (b) zeigt die Variante einer ungeordneten Fragmentierung, bei welcher die Cluster derselben Datei zwar direkt nebeneinander liegen, sich jedoch in falscher Reihenfolge befinden. Eine Kombination dieser beiden Fragmentierungsmöglichkeiten ist die nicht-zusammenhängende und ungeordnete Fragmentierung (c). Bei dieser sind die Fragmente sowohl durch Lücken getrennt, als auch in einer ungeordneten Reihenfolge. [28].

Die Diversität der Fragmentierung stellt eine Herausforderung bei der Rekonstruktion einer derartigen Datei dar. Bisher wurde lediglich eine einzelne fragmentierte Datei, eine sogenannte *Intrafile-Fragmentierung*, betrachtet. Allerdings zählt bereits das Speichern von zwei Dateien, welche im selben Ordner liegen, auf unterschiedlichen Bereichen der Festplatte als Fragmentierung. In diesem Fall spricht man von einer *Interfile-Fragmentierung*. [27]
Diese Art von Fragmentierung ist für den weiteren Verlauf der Arbeit irrelevant. Jedoch müssen einige Begrifflichkeiten geklärt werden. Der Begriff Basis-Fragment bezieht sich auf das erste Fragment einer Datei. In diesem ist der Header der Datei zu finden. Des Weiteren gibt es einen Fragmentationspunkt, worunter das letzte Cluster vor Entstehen einer Fragmentierung, verstanden wird. Bei einer Mehrfach-Fragmentierung existieren mehrere Fragmentationspunkte. [29]

Abbildung 2.4: Darstellung der drei Möglichkeiten einer Fragmentierung

2.2 JPEG

Bei dem Begriff JPEG handelt es sich um ein Kompressionsverfahren für Bilder. Es leitet sich von *Joint Photographic Experts Group* ab. Hierbei handelt es sich um eine Gruppe von Entwicklern, bestehend aus Forschern und Firmen. Die Entwicklung des Verfahrens begann bereits im Juni 1987 und die erste Veröffentlichung folgte im Jahr 1991. [30]
Der Begriff JPEG wird häufig im Sprachgebrauch gleichbedeutend mit der Bezeichnung Bilddatei benutzt und als eigenes Dateiformat gesehen. Allerdings steht der Ausdruck JPEG lediglich für ein Kompressionsverfahren für Bilddateien oder Methode zur Bildkompression. Es konkretisiert lediglich den JPEG-Kompressor und Dekompressor. Mit Hilfe der JPEG-Kompression ist ein Kompressionsfaktor von circa 1:25 möglich. [31]
Durch diese hohe Kompressionsrate hat JPEG sich für das Speichern von Bildern durchgesetzt und hat sich als häufigstes Verfahren zur Kompression von Bildern bei Digitalkameras etabliert [30].

2.2.1 Dateinamensuffix

Bilddateien, welche das JPEG-Kompressionsverfahren verwenden, können diverse Dateinamensuffixe besitzen. Verbreitete Endungen sind hierbei *jpeg, jpg, jfif* und *jpe* [32] [33]. Bei der Verarbeitung der Bilder haben die Suffixe keine Relevanz, da die Systeme, die derzeit verwendet werden, mit den unterschiedlichen Endungen umgehen kön-

nen [34]. Die Entstehung unterschiedlicher Endungen lässt sich beispielsweise anhand des Suffix *jpg* darlegen. Dieses ist noch auf das alte Windows Betriebssystem MS-DOS zurückzuführen. [35]
Das damalige Betriebssystem war lediglich in der Lage, Endungen mit drei Buchstaben zu verarbeiten. Die heutigen Systeme besitzen dieses Problem nicht mehr und können auch mit Dateiendungen mit mehr als drei Buchstaben arbeiten. Somit besteht zwischen *jpeg* und *jpg* kein inhaltlicher Unterschied. [34]
Eine Umbenennung von *jpeg* zu *jpg* und umgkehrt ist aus diesem Grund problemlos möglich. Die Verwendung der unterschiedlichen Suffixe kann auch mit der Größe und Qualität der Bilder zusammenhängen. Das Suffix *jpe* wir zum Beispiel für komprimierte 24-Bit-Rasterbilder genutzt. Sie haben eine vergleichsweise niedrige Auflösung. Zu Bildern dieser Art zählen neben Digitalkamera-Bildern schlechter Qualität auch Miniaturansichten und Albumcover. [36] [37]

2.2.2 Dateiformate

Da es sich bei JPEG lediglich um ein Kompressionsverfahren handelt, müssen die Dateiformate, welche dieses Kompressionsverfahren verwenden, Erwähnung finden. Zu den Dateiformaten, welche auf dem JPEG-Verfahren beruhen, zählen JFIF, SPIFF und JNG. Aufgrund ihres seltenen Vorkommens spielen SPIFF und JNG nur eine sehr geringe Rolle und werden bei der Prototyp-Entwicklung nicht berücksichtigt. [38]
Die Abkürzung SPIFF steht für *Still Picture Interchange File Format*. Dieses Bildformat wird von leistungsfähigen Programmen, welche der Bildbearbeitung dienen, verwendet. Eines dieser Programme ist Adobe Photoshop. [39]
Das Akronym JNG steht für *JPEG Network Graphics* und kann die Bilddaten sowohl verlustfrei, als auch verlustbehaftet speichern [40].

Das eigentliche Dateiformat, welches gemeint ist, wenn fälschlicherweise von einem JPEG-Format gesprochen wird, ist das JFIF Format. Es ist am verbreitetsten, weshalb sich die vorliegende Arbeit auf dieses Format stützt. [39] JFIF wurde von Eric Hamilton und der IJG, *Indendent JPEG Group*, entwickelt. Es steht für *JPEG File Interchange Format*. Es definiert die im JPEG-Standard nicht spezifizierten Informationen zum verwendeten Farbraum und der damit verbundenen Transformation. [31]. Als Farbraum wird nur der YCbCr Farbraum zugelassen, wodurch eine vorherige Farbraumtransformation unumgänglich ist [41].

Eine Abwandlung von JFIF stellt das EXIF-Dateiformat dar. Das *Exchangeable Image File Format* wird zur Speicherung von Bildaufnahmen von Digitalkameras genutzt. Mit Hilfe des EXIF-Formats können Metadaten, Aufnahmeparameter und Informationen über die Farbraum- und Farbanpassungseinstellungen der verwendeten Digitalkamera im Header gespeichert werden. Im Gegensatz zum JFIF Format existiert nur eine unzureichende Standardisierung, wodurch die diversen Kamerahersteller unterschiedliche

Informationen der Bilddatei anhängen können. In der Regel ist jedoch bei allen EXIF-Bildern ein Vorschaubild enthalten. Die Anzahl der Vorschaubilder kann abhängig vom Hersteller variieren. [31] [42]

2.2.3 JPEG-Kompressionsmodi

Für den JPEG-Standard kommen vier unterschiedliche Kompressionsmodi zum Einsatz. Diese vier werden als sequenzielle, progressive, hierarchische und verlustfreie Kompression bezeichnet [17].

- **Sequenziell**
  Die sequenzielle Kompression wird nochmal unterteilt in baseline und erweiterte Kompression [43]. Bei der sequenziellen Methode läuft die Kodierung von Bildern von oben nach unten ab [44].
  Dies ist sowohl bei der baseline, als auch der erweiterten Kompression identisch. Der einzige Aspekt, welcher zu der tiefergehenden Unterteilung führt, ist die Anzahl der zu verwendenden Quantisierungs- und Huffman-Tabellen. Bei der baseline Kompression existiert eine Limitierung von zwei Quantisierungs- und vier Huffman-Tabellen. Diese entfällt bei der erweiterten Variante. [43]

- **Progressiv**
  Bei der progressiven Kompression wird nicht, wie bei der sequenziellen, das Bild von oben nach unten kodiert. Die einzelnen Komponenten des Bildes werden in mehreren Scans kodiert. Das Bild wird quasi erst grob aufgelöst und dann immer feiner, wodurch bereits nach ersten Scans eine grobe Darstellung des Bildes entsteht. [44]

- **Verlustfrei**
  Bei einer verlustfreien Kompression handelt es sich um eine Kompression, bei welcher, wie der Name bereits vermuten lässt, keine Informationen verloren gehen. Hierbei wird das exakte Originalbild beibehalten. [44]

- **Hierarchisch**
  Die hierarchische Kompression unterteilt das Bild zu Beginn in kleine Bildteile. Man spricht bei den entstandenen Teilbildern von Frames. Ein erstes Frame dient zur ersten Darstellung des Bildes in niedriger Auflösung. Alle folgenden Frames werden für die Verfeinerung des Bildes benötigt. [44]

Der Unterschied zwischen der progressiven und baseline Kompression zeigt sich beim Laden dieser Bilder. In Abbildung 2.5 sind die Ladephasen dargestellt, woran zu erkennen ist, dass baseline JPEGs Zeile für Zeile laden und progressive JPEGs im Ladever-

lauf schärfer werden.
Spricht man von einem JPEG-Bild, bezieht man sich in der Regel auf die baseline Kompression, da diese am häufigsten Verwendung findet [45]. Diese zeigt die verschiedenen Ladestadien eines Bildes über einen Browser. Anhand dieses Bildes wird veranschaulicht, wie diese Kompressionsmodi ablaufen und sich für den Anwender unterscheiden.

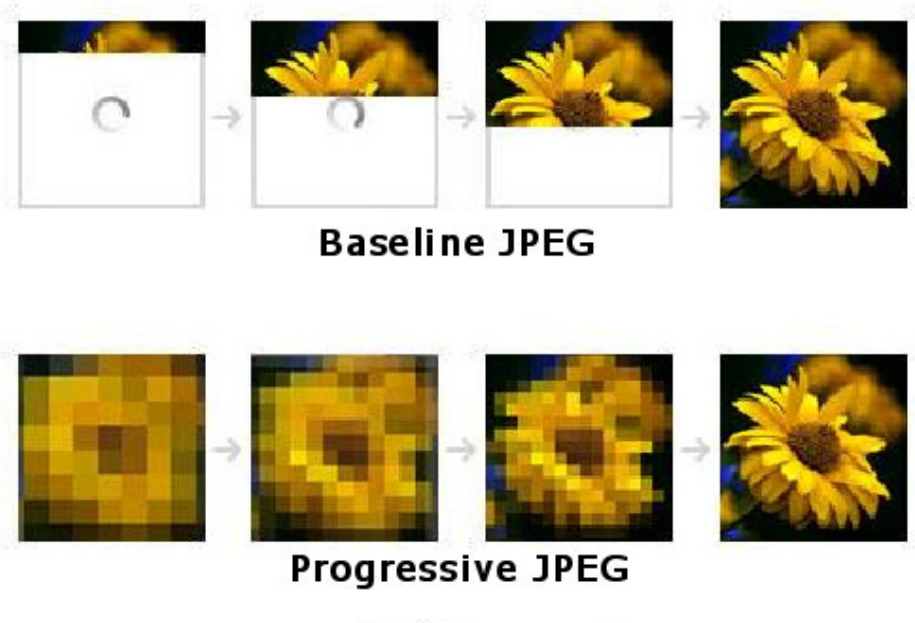


Abbildung 2.5: Gegenüberstellung der Ladephasen einer baseline und progressiven JPEG-Kompression [46]

2.2.4 JPEG-Kodierung

Der JPEG-Standard setzt für die Bilddatenkompression ein Verfahren voraus, welches im Folgenden in seinen einzelnen Schritten definiert wird. In Abbildung 2.6 sind diese zum besseren Verständnis graphisch dargestellt. Blau markierte Schritte symbolisieren die Bereiche, welche irreversible Reduktionen beinhalten. Bevor diese Schritte beginnen können, muss allerdings ein digitales Rasterbild vorliegen, Dabei handelt es sich um ein für den Computer interpretierbares Bild, welches in kleine Elemente, sogenannte Pixel, zerteilt ist. [30].
Dieses Rasterbild wird in den meisten Fällen, beispielsweise standardisiert bei JFIF-Dateien, in einem RGB-Farbraummodell dargestellt. Die Abkürzung RGB steht hierbei für die Grundfarben rot, grün, blau, welche jedem Pixel zugeordnet werden. Aus diesen Grundfarben werden durch ein additives System alle anderen Farben erzeugt. Die JPEG-Kodierung beginnt damit, dass das Bild in einen anderen Farbraum transformiert werden muss. [32]

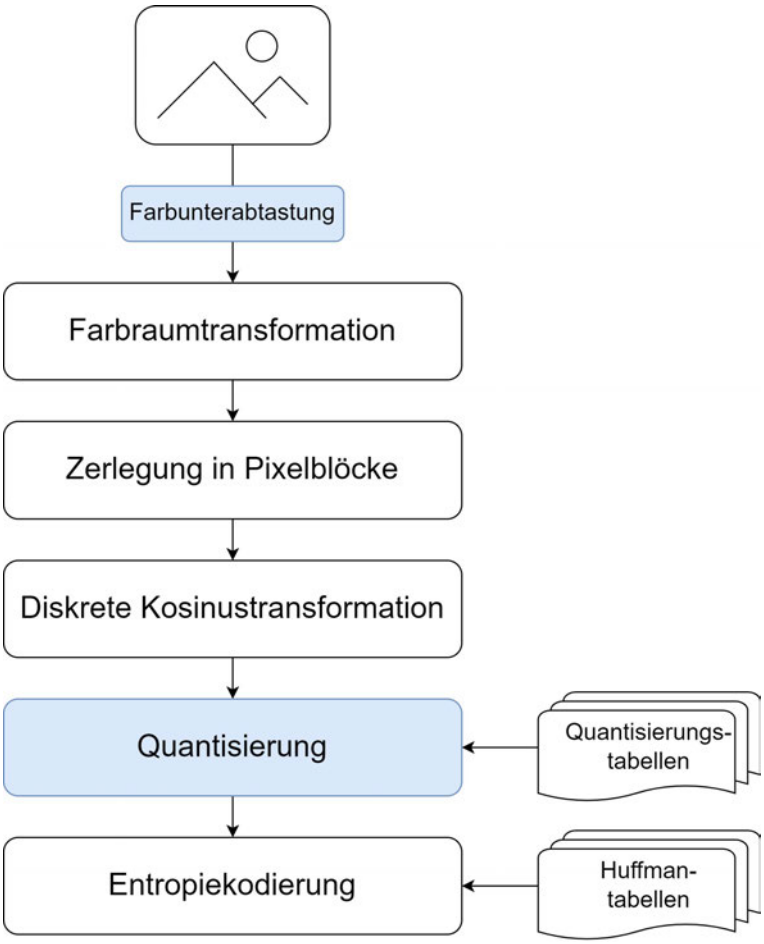


Abbildung 2.6: Schritte der JPEG-Kodierung

1. **Farbraumtransformation**
   In den häufigsten Fällen liegt ein Bild im RGB-Farbraum vor, allerdings wird für die Weiterverarbeitung der YCbCr-Raum benötigt. Hierbei werden nicht die Farben, sondern die Grundhelligkeit Y (Luminanz), die Farbabweichung in Richtung Blau Cb (Chrominanz Blau) und die Farbabweichung in Richtung Rot Cr (Chrominanz Rot) dargestellt. Eine Gegenüberstellung beider Farbräume zeigt Graphik 2.7. Für die Konvertierung des Farbraums muss lediglich eine Umrechnung mit Hilfe einer Formel erfolgen, welcher nach Bedarf Gewichtungen zugefügt werden können. [32] [43]
   Die zuerst angegebene Formel zeigt die Formel zur Umrechnung vom RGB in den YCbCr Raum [32].

$$\begin{aligned} Y &= Rot + Grün + Blau \\ Cb &= Blau - Rot - Grün \\ Cr &= Rot - Blau - Grün \end{aligned} \tag{2.1}$$

Wie zuvor Erwähnung gefunden, kann die Formel mit Gewichtungen ergänzt werde, um eine größere Datenreduktion zu erzielen. Wie die Berechnung mit Gewichtung erfolgt, ist in der folgenden Formel beispielhaft angegeben [32].

$$\begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix} = \begin{pmatrix} 0,2990 & 0,5870 & 0,1140 \\ -0,1687 & -0,3313 & 0,5000 \\ 0,5000 & -0,4187 & -0,0813 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{2.2}$$

Zweck der Transformation ist die Reduktion von irrelevanten Informationen. Während der Informationsgehalt beim RGB-Farbraum aller Komponenten etwa gleichverteilt ist, sind die meisten Informationen beim YCbCr-Farbraum im Luminanzanteil zu finden. Das menschliche Auge hat die Eigenschaft, minimale Veränderungen in der Farbe nicht erkennen zu können, weshalb ein Verringern der Chrominanzanteile gar nicht oder wenn nur minimal auffällt. Die Qualität bleibt für den Betrachter gleich. Dieser Prozess der irreversiblen Datenreduktion wird Farbunterabtastung genannt. [43]
Als Ergebnis erhält man drei Bilder, welche in den nächsten Schritten separat voneinander weiterverarbeitet werden.

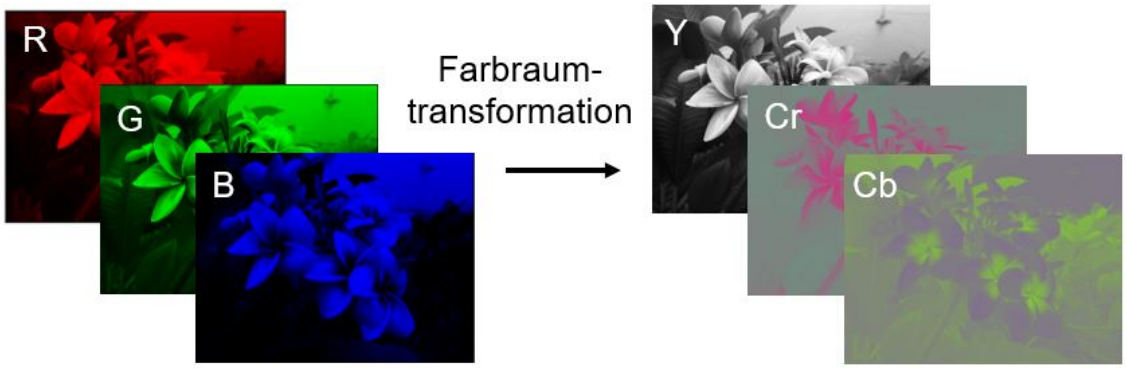


Abbildung 2.7: Gegenüberstellung der Farbräume durch die Darstellung eines Bildes in jedem RGB- und YCbCr-Farbkanal

2. **Zerlegung in 8x8 Blöcke**
   Für die Weiterverarbeitung des Bildes muss dieses in gleichgroße Blöcke aufgeteilt werden. Hierzu werden kleine Felder von 8x8 Pixeln gebildet. [31] Ist eine Zerteilung in Blöcke dieser Größe aufgrund der Bildgröße nicht möglich, wird das Bild um die fehlenden Pixel von Bildrändern ergänzt.

3. **Diskrete Kosinustransformation**
   Die vorherige Zerlegung in gleichgroße Blöcke ist für die zweidimensionale Diskrete Kosinustransformation notwendig, welche anschließend auf jeden einzelnen Block angewendet wird. Bei der Diskreten Kosinustransformation (kurz DCT) handelt es sich um eine orthogonale Transformation, verwandt mit der Fourier-Transformation, welche die Informationen von dem Orts- in den Frequenzraum umwandelt. Im Allgemeinen ist die DCT verlustfrei und lässt sich durch die inverse Diskrete Kosinustransformation zurückrechnen, jedoch kann es durch Rundungsfehler zu Verlusten kommen. Als Ergebnis der DCT erhält man zu jedem Pixelblock 64 Koeffizienten, welche sich in einen Gleichanteil, genannt DC (*direct current*) und 63 Wechselanteile, den sogenannten ACs (*alternating current*) aufteilen. Die zurückgelieferten Koeffizienten stellen eine Amplitude einer zweidimensionalen Frequenz dar. [47] Der DC-Koeffizient befindet sich in der oberen linken Ecke und entspricht dem Frequenzanteil von Null. Je weiter die Entfernung von diesem ist, desto höher werden auch die Frequenzen der AC-Werte. [48]
   Die Sortierung der Koeffizienten einer zweidimensionalen DCT sind in Abbildung 2.8 erkennbar.

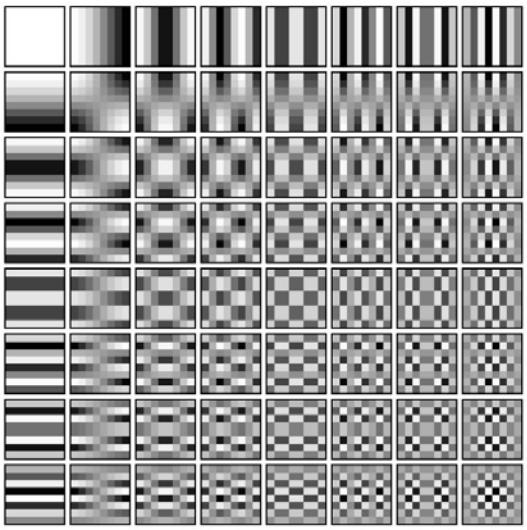


Abbildung 2.8: Basisbilder einer zweidimensionalen Diskreten Kosinustransformation [32]

4. **Quantisierung**
   Auf die Diskrete Kosinustransformation folgt die Quantisierung, durch welche Informationen verlustbehaftet reduziert werden. Die Quantisierung erfolgt mit Hilfe einer Quantisierungstabelle, zu finden im Header der JPEG-Datei. Wo genau sich diese im Header befindet, wird in Kapitel *JPEG-Marker und -Segmente* behandelt. Die Quantisierungstabelle ist eine Matrix mit genauso vielen Werten wie es DCT-Koeffizienten gibt. Die Werte, die sich in ihr befinden sind abgeleitet von der

Empfindlichkeit des menschlichen Auges in Bezug auf die Ortsfrequenzen. Diese werden durch die DCT-Koeffizienten geteilt und anschließend auf die nächste ganze Zahl gerundet. Durch diesen Prozess entsteht eine Vielzahl von Nullwerten, welche zusammengefasst werden können und so eine Reduktion von irrelevanten Informationen stattfindet. Das Ergebnis der Quantisierung wird umsortiert und anhand der Frequenz in einem Zickzack-Schema angeordnet. Das Zickzack-Schema wird wie in Abbildung 2.9 gelesen. Man beginnt beim Gleichanteil, dem DC, in der oberen linken Ecke und endet mit dem höchsten AC-Koeffizienten. [48]

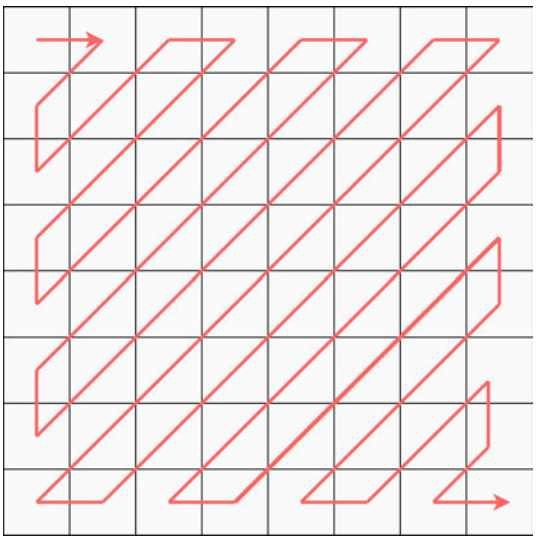


Abbildung 2.9: Zickzack-Schema der umsortierten DC- und AC-Koeffizienten nach M.Y.U. Khalid [49]

5. **Entropiekodierung**
   Im Anschluss wird das Bild mit Hilfe von Methoden der Entropikodierung verlustfrei komprimiert. Verwendet werden hierbei diverse Verfahren, die beispielsweise Redundanzen zusammenfassen, ohne dabei Informationen zu entfernen. Für den JPEG-Standard werden üblicherweise die Huffman-Kodierung, in manchen Fällen die Lauflängenkodierung, sowie die Delta-Enkodierung genutzt. Es gibt noch weitere Methoden, die eine verlustfreie Komprimierung ermöglichen, allerdings wird auf diese im weiteren Verlauf nicht weiter eingegangen. [31]

   Lauflängenkodierung
   Eine Methode, um die Datenlast zur reduzieren, ohne dass Verluste entstehen, ist die Lauflängenkodierung. Diese nutzt das Zickzack-Muster aus, welches nach der Quantisierung entsteht, und erzeugt einen Vektor. Der anschließend vorliegende Vektor weist Wiederholungen von Werten und eine Anhäufung von Nullen auf. Damit nicht jeder Wert wiederholt abgespeichert werden muss, wird bei der Lauflängenkodierung lediglich der Wert und die Häufigkeit des Vorkommens, sei-

ne sogenannte Lauflänge, gespeichert. [50]
Für das bessere Verständnis wird die Lauflängenkodierung anhand des folgenden Vektors *(3 3 3 1 5 5 5 2 2 2)* betrachtet.

3 3 3	1	5 5 5	2 2 2
(3,3)	(1,1)	(3,5)	(3,2)

Wie zu erkennen ist, enthält dieser eine Vielzahl an sich wiederholenden Zahlen, welche mit Hilfe der Lauflängenkodierung reduziert werden können. Der erste Wert *3* kommt dreimal hintereinander vor, weshalb er nach dem vorliegenden Prinzip als *(3,3)*, also dreimaliges Vorkommen des Wertes *3*, abgespeichert werden kann. Macht man das mit dem vollständigen Vektor, erhält man *(3,3), (1,1), (3,5) (3,2)* und reduziert die Datenmenge auf diese Weise. Diese Reduktion ist nicht irreversibel, da aus den vorhandenen Informationen der ursprüngliche Vektor wieder hergeleitet werden kann.

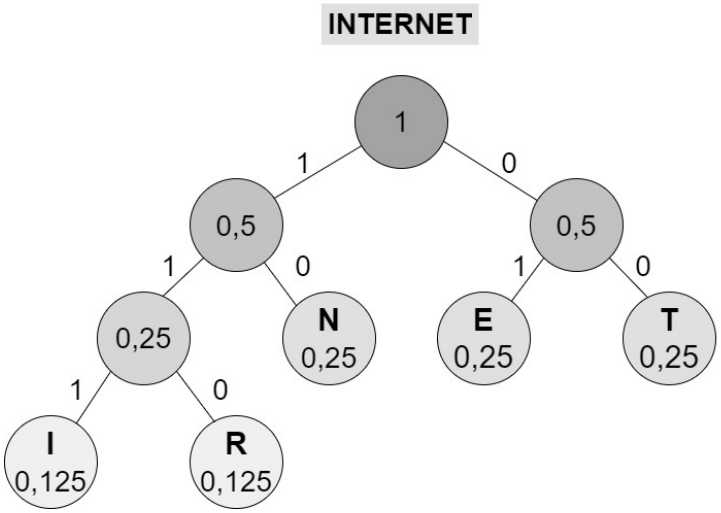
Deltakodierung
Eine weitere Methode die Informationslast ohne Verlust von Informationen zu reduzieren ist die Deltakodierung. Bei diesem Prinzip wird wie bei der Lauflängenkodierung der Vektor betrachtet. Anstatt die einzelnen Werte zu speichern, wird lediglich die Veränderung zum vorherigen Wert hinterlegt. [49]
Diese Methode macht sich ebenfalls das Zickzack-Muster und die damit verbundene Sortierung zunutze. Die Koeffizienten unterscheiden sich durch ihre aufsteigende Anordnung nur minimal vom vorherigen Wert.
Beispielhaft wird der Vektor *(10 10 12 15 17 18)* betrachtet.

10	10	12	15	17	18
10	0	2	3	2	1

Wendet man die Deltakodierung auf die ersten beiden Werte an, dann erhält man ein Ergebnis von *0*, da zwischen zehn und zehn kein Unterschied besteht. Betrachtet man den nächsten Wert *12*, ist eine Differenz von *2* vorhanden. Dieses Vorgehen wendet man auf die gesamten Werte an und erhält als einen Ausgangsvektor von *(10 0 2 3 2 1)*. Anhand des erhaltenen Ergebnisses erkennt man, dass die Zahlen wesentlich kleiner geworden sind, wodurch die Datenmenge reduziert wurde.

Huffman-Kodierung
Die Idee hinter der reversiblen Datenreduktion mit Hilfe der Huffman-Kodierung ist ein binärer Baum, welcher die Häufigkeit des Auftretens der einzelnen Zeichen berücksichtigt. Zu Beginn der Huffman-Kodierung müssen die Häufigkeiten der einzelnen Zeichen in einer Häufigkeitstabelle festgehalten werden. Aus dieser

werden anschließend zu jedem Zeichen Blattknoten eines Baumes erstellt, wobei die Häufigkeit als Gewichtung genommen wird. Der Baum wird von den Knoten ausgehend zu den Wurzeln gelesen und daraus eine Code-Tabelle erstellt. Diese Code-Tabelle definiert anhand des Baumes zu jedem Zeichen einen Bitcode, welcher bei häufig vorkommenden Zeichen kurz und bei selten vorkommenden Zeichen länger ist. [51]
Zur Veranschaulichung wurde für das Wort *INTERNET* die Huffman-Kodierung einmal beispielhaft in Abbildung 2.10 durchgeführt.



	Wahrscheinlichkeit	Code
I	1/8 = 0,125	111
N	2/8 = 0,25	10
T	2/8 = 0,25	00
E	2/8 = 0,25	01
R	1/8 = 0,125	110

Abbildung 2.10: Beispielrechnung der Huffman-Kodierung für das Wort *INTERNET*

Zu Beginn muss die Wahrscheinlichkeiten des Vorkommens der einzelnen Buchstaben im Wort *INTERNET* bestimmt werden. Da das Wort aus acht Buchstaben besteht, wird in Tabelle *Wahrscheinlichkeiten* zu Beginn die Anzahl des Auftretens des jeweiligen Buchstabens durch acht geteilt. Im Anschluss zeichnet man den Huffman-Baum zu der Tabelle, indem man mit den untersten Knoten und der geringsten Wahrscheinlichkeit beginnt. Danach arbeitet man sich Stück für Stück bis zum Wurzelknoten, der die Gesamtwahrscheinlichkeit von hat, hoch. Bei dem Vorgehen kann es verschiedene Lösungen geben und die dargestellte ist lediglich eine von diversen Möglichkeiten. Hat man den Baum gezeichnet, wird den Pfaden, die nach links gehen, der Wert *0* oder *1* zugeteilt. Der rechte Pfad erhält die andere der beiden Zahlen. Für dieses Beispiel wurde für den linken Pfad die *1* und den rechten die *0* gewählt.
Zum Ermitteln des Codes wird von dem Wurzelknoten zur Wahrscheinlichkeit des jeweiligen Buchstabens heruntergewandert und die Zahlen des Pfades bis zu dem gewünschten Knoten notiert. Hierbei dürfen keine Dopplungen auftreten.

2.2.5 JPEG-Marker und -Segmente

Dateien, die mit Hilfe des JPEG-Standards komprimiert wurden, besitzen einen prägnanten Aufbau. Sie sind in einzelne Segmente aufgeteilt, welche anhand eindeutiger Marker eingeleitet werden. Die relevantesten Marker werden im Folgenden behandelt. Alle diese Marker werden mit dem Byte-Wert `0xFF` eingeleitet, gefolgt von einem weiteren Byte, welches die Identifikationsnummer beinhaltet. Anhand dieser Identifikationsnummer wird die Art des folgenden Segments beschrieben. [52]
Nach der Identifikationsnummer für das folgende Segment folgt in der Regel eine Angabe über die Länge des Segments. Der *Start-Of-Image*, *End-Of-Image* und die *Restart-Marker* benötigen keine Längenangabe, da sie eine Ausnahme bilde. Außerdem leiten sie, wie die anderen Marker, kein Segment ein. In Abbildung 2.11 ist der Header einer JPEG-Datei, geöffnet in einem Hexeditor, abgebildet. Anhand dieses Ausschnitts sind die verschiedenen Marker gefolgt von der Längeangabe zu erkennen.

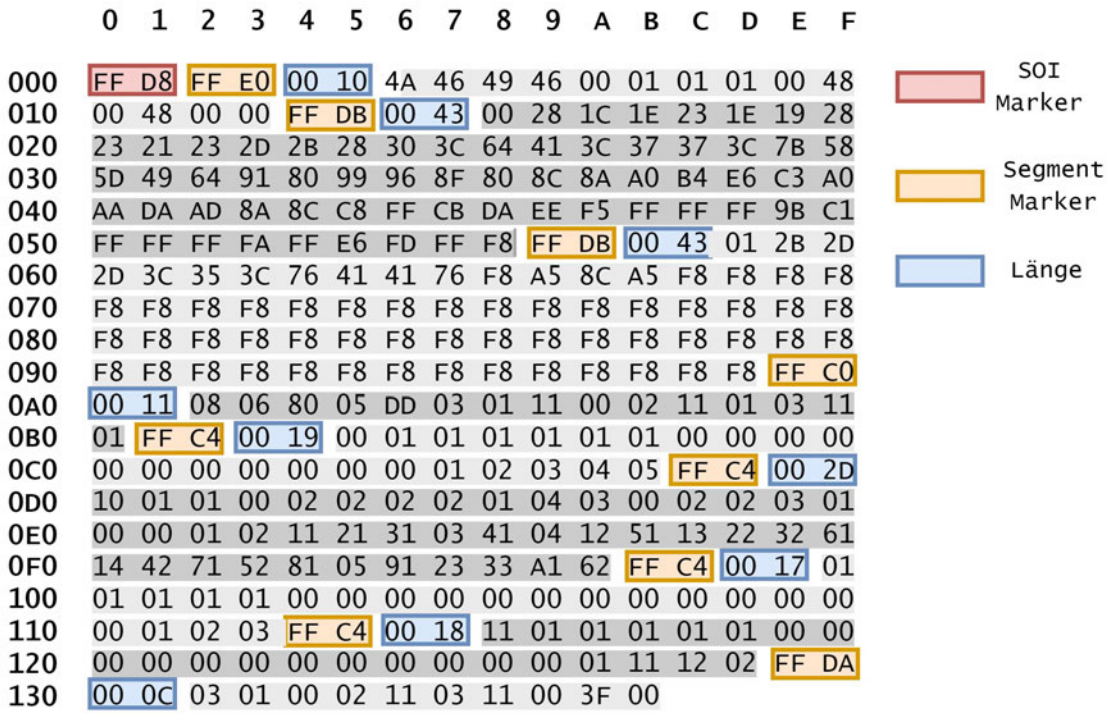


Abbildung 2.11: Ausschnitt des JPEG-Headers geöffnet mit einem Hexeditor

In den eigentlichen Bilddaten kann es vorkommen, dass der Wert `0xFF` zufälligerweise vorkommt. Damit es zu keiner Verwechslung mit den Markern kommt, wird dieser durch einen darauffolgenden Bytewert `0x00` entwertet. [53]
In Tabelle 2.1 ist eine Auflistung der relevantesten Marker mit ihrem Hexadezimalwert und ihrer Beschreibung zu finden. Der Beginn einer JPEG-Datei wird mit einem sogenannten *Start-Of-Image Marker (SOI)* gekennzeichnet. Dieser befindet sich immer

am Anfang eines Clusters, kann allerdings weitere Male auftauchen. Das wiederholte Vorkommen ist auf eingebettete Vorschaubilder zurückzuführen. Ob ein Vorschaubild vorhanden ist, hängt von dem Marker ab, welcher auf den SOI Marker folgt, dem sogenannten *Application Marker (APP)*. Dieser signalisiert wie die vorliegende Bilddatei zu interpretieren ist. Im Allgemeinen sind in dem Application Segment Informationen zur Pixeldichte und JPEG-Informationen wie beispielsweise eine Kennung des JPEG Dateiformats vorhanden. Am weitesten verbreitet sind die Kennungen `0xFFE0` und `0xFFE1`, wobei es sich um die Identifikationsnummer der JFIF- und EXIF-Dateien handelt. `0xFFE1` leitet das Application Segment von EXIF-Dateien ein, welche wie bereits erwähnt Vorschaubilder enthalten, wodurch an dieser Stelle mit einem wiederholten Vorkommen des SOI Markers zu rechnen ist. Aufgrund dieser Eigenschaft wurde lediglich das JFIF-Dateiformat für den Prototyp berücksichtigt. Zusätzlich zu der Kennung folgen Byte-Werte, die JFIF und EXIF durch ASCII-Kodierung, welches genutzt wird, um Zeichensätze zu kodieren, in ASCII-Zeichen, ein Code, um Zeichensätze zu kodieren, repräsentieren. [53]
Auf die anderen vorkommenden IDs wird nicht weiter eingegangen, da diese aufgrund seltenen Vorkommens nicht relevant sind.
Das nächste Segment, eingeleitet durch die Byte-Sequenz `0xFFDB`, definiert die Quantisierungstabellen. Es können mehrere dieser Segmente in der JPEG-Datei vorkommen. Neben der Längenangabe sind auch Informationen zum Index der Quantisierungstabelle vorhanden. [54]
Auf dieses Segment folgt das *Start-of-Frame Segment (SOF)*. Die IDs dieses Markers sind abhängig von dem für die Kompression verwendeten Algorithmus vergleiche Kapitel 2.2.3. Nachdem der Modus durch die ID festgelegt wurde, folgen Informationen zu Bildgröße, Genauigkeit der Daten, Downsamplingfaktor sowie Anzahl der Farbkomponenten [55]. In der Regel kommt dieses Segment lediglich einmal vor, allerdings stellt der hierarchische Kompressionsmodus hierbei eine Ausnahme dar [52].
Die nächste relevante Byte-Sequenz ist `0xFFC4`, welche das Huffman-Tabellen Segment einleitet. In diesem Segment befindet sich neben der eigentlichen Huffman-Tabelle noch ein Index zu den betreffenden Farbkomponenten [54].
Das Segment, welches die eigentlichen Bilddaten enthält, ist der *Start-Of-Scan (SOS)*. Dieser wird durch die Marker-Sequenz `0xFFDA` eingeführt. Neben der Länge des Segments sind Informationen zur Anzahl der Komponenten und Zuordnung der zugehörigen Tabellen in diesem Segment zu finden. Um zu signalisieren, dass die Datei beendet ist, folgt ein *End-of-Image Marker (EOI)*. Nach diesem folgen keine Daten mehr, die der Datei zugehörig sind. Zusätzlich zu den genannten Markern können noch weitere weniger relevante Marker existieren. Zu diesen zählen beispielsweise der *Restart-(RST)*, der *Define-Restart-Interval (DRI)* und *Comment Marker (COM)*. Der RST und DRI Marker kommen nur gemeinsam in den entropiekodierten Daten innerhalb des *Start-Of-Scan Segments* vor. Der RST zeigt hierbei an, dass Daten dekodiert werden können, ohne dass Wissen über die Daten zuvor notwendig ist. Der DRI dient dabei der Festlegung der Abstände der Marker. [52]

Der bereits erwähnte *Comment Marker* muss nicht, aber kann vorkommen. Er leitet das Kommentar Segment ein, welches für textuelle Anmerkungen genutzt wird [56].
Eine Übersicht über die wichtigsten JPEG-Marker mit einer Beschreibung, wofür die einzelnen Byte-Werte ist im Anhang A unter A.1 zu finden.

Tabelle 2.1: Auflistung der relevanten JPEG-Marker nach S.A. Sari und M. Mohamad [26]

Hexadezimalwert	**Abkürzung**	**Beschreibung**
FFD8	SOI	Start-of-Image: Beginn der JPEG-Datei
FFE0-FFEF	APP$_n$	Application: Anwendungsspezifischer Marker
FFDB	DQT	Define-Quantization-Table(s): Tabelle(n) für die Quantisierung
FFC4	DHT	Define-Huffman-Table(s): Tabelle(n) für die Huffman-Kodierung
FFDA	SOS	Start-of-Scan: Komprimierte Bilddaten
FFD9	EOI	End-of-Image: Ende der JPEG-Datei
FFD0-FFD7	RST	Restart: Neustart der Dekodierung
FFDD	DRI	Define-Restart-Interval: Definiert Abstände der Marker
FFF0-FFFD, FFFE	COM	Comment: Textuelle Anmerkungen
Start-of-Frame Marker, nicht differentiell, Huffman-Kodierung		
FFCO	SOF$_0$	Baseline DCT
FFC1	SOF$_1$	Erweiterte sequentielle DCT
FFC2	SOF$_2$	Progressive DCT
FFC3	SOF$_3$	Verlustfrei
Start-of-Frame Marker, differentiell, Huffman-Kodierung		
FFC5	SOF$_5$	Differentielle sequentielle DCT
FFC6	SOF$_6$	Differentielle progressive DCT
FFC7	SOF$_7$	Differentiell verlustfrei
Start-of-Frame Marker, nicht differentiell, arithmetische Kodierung		
FFC9	SOF$_9$	Erweiterte sequentielle DCT
FFCA	SOF$_{10}$	Progressive DCT
FFCB	SOF$_{11}$	Verlustfrei
Start-of-Frame Marker, differentiell, arithmetische Kodierung		
FFCD	SOF$_{13}$	Differentielle sequentielle DCT
FFCE	SOF$_{14}$	Differentielle progressive DCT
FFCF	SOF$_{15}$	Differentiell verlustfrei

2.3 JPEG 2000

Die *Joint Photographic Experts Group* hat eine überarbeitete und angepasste Version von JPEG entwickelt [57]. Dies ist das JPEG 2000 Kompressionsverfahren, welches wie anhand des Namens erkennbar, im Jahr 2000 entwickelt wurde. Dieses Verfahren ermöglicht sowohl eine verlustfreie als auch verlustbehaftete Komprimierung [58]. Im Gegensatz zum Vorgänger-Verfahren ist eine stärkere Komprimierung möglich [31] [58]. Der Unterschied beider Verfahren liegt in der Transformation. JPEG arbeitet mit der Diskreten Kosinus Transformation, welche zuvor eine Zerlegung in 8x8 Blöcke benötigt. Bei JPEG 2000 erfolgt eine Unterteilung frequenz- und auflösungsbasiert und anschließend folgt eine Diskrete Wavelet Transformation. [31] [59]
JPEG 2000 findet bereits Anwendung in einigen Bereichen wie der Medizin [57]. Da es allerdings nicht kompatibel ist mit JPEG, ist eine Umstellung auf das Format aufwändig, wodurch es von wenigen Systemen und Programmen akzeptiert und unterstützt wird [58]. Aus diesem Grund konnte es sich bisher noch nicht durchsetzen, weshalb es für den weiteren Verlauf der Arbeit nur sehr wenig Relevanz hat und außer Acht gelassen werden kann [57].

2.4 Carving

Carving, *File Carving* oder auch als *Data Carving* bezeichnet, befasst sich mit der Wiederherstellung von Dateien, ohne Zugriff auf die Metadaten des Dateisystems [60]. Grund hierfür kann ein Fehlen, ein Defekt des Dateisystems oder das Löschen/ Verstecken von Dateien im nicht zugewiesenen Bereich sein [61]. Ein nicht zugewiesener Bereich ist der Platz auf der Festplatte, welcher keinen Eintrag im Dateisystem besitzt und somit von diesem nicht reserviert ist [62].
File Carving wird häufig mit Datenwiederherstellung gleichgesetzt, ist aber ein Spezialfall der Dateiwiederherstellung. Bei der einfachen Dateiwiederherstellung ist das Dateisystem vorhanden und bei der Datenaufbereitung wird auf diese Informationen zurückgegriffen. [61]
Beim *File Carving* werden die Rohdaten betrachtet und aus diesen anhand bestimmter Dateistrukturen und dem Inhalt der Dateien, diese wiederhergestellt [24] [63]. Die Metadateninformationen werden hierbei nicht berücksichtigt [60]. Für das *File Carving* wurden diverse Methoden und Herangehensweisen entwickelt. Die Wahl der Methode im Zusammenhang mit den Dateitypen der wiederherzustellenden Dateien und der Häufigkeit der Fragmentierung sind abhängig von der Erfolgsquote der Wiederherstellung. Im Folgenden werden die diversen Carving Methoden vorgestellt. Die Unterteilung in diese Methoden kann in der Literatur dadurch variieren, dass eine gröbere oder kleinere Aufteilung der Methoden vorgenommen wurde.

2.4.1 Signaturbasiertes Carving

Bei dem signaturbasierten Carving handelt es sich um eine Methode zum Herstellen von Dateien, welche von den ersten Carving-Algorithmen benutzt wurden. Das Prinzip dieser Methode ist sehr simpel und deckt den Umgang mit fragmentierten Dateien nicht ab. Wie der Name schon andeutet, setzt das signaturbasierte Carving auf bestimmte Signaturen in Dateien. Einige Dateitypen, wie auch JPEG-Dateien, verwenden bestimmte Byte-Folgen als Markierungen, kurz Marker, um den Beginn einer Datei zu symbolisieren. Damit die gesamte Datei aufgespürt wird, gibt es drei Methoden, die auf unterschiedliche Weise die Daten der Datei extrahieren. [64]

- **Header / Footer:**
  Das Header-Footer Carving ist bei Dateien möglich, die neben dem Marker, der den Beginn einer Datei symbolisiert, auch einen haben, der das Ende angibt. Wendet man dieses Prinzip beispielsweise auf JPEG-Dateien an, so werden diese mit dem Marker `0xFFD8`, dem sogenannten Header eingeleitet, und mit `0xFFD9`, dem Footer, beendet. Diese Eigenschaft nutzen Signaturbasierte Carver aus, indem nach den bekannten Byte-Sequenzen gesucht wird und die Bereiche zwischen Header und Footer Marker mit extrahiert werden. [64]

- **Header / Eingebettete Dateigröße:**
  Eine weitere Möglichkeit Dateien herauszufiltern, allerdings ohne auf den Footer zurückzugreifen, geschieht anhand der Größe. Einige Dateitypen speichern im Header Angaben zur Größe der Datei. Nach dem Auffinden des Headers durch einen entsprechenden Marker, kann die Größe aus diesem ermittelt und anschließend der relevante Bereich extrahiert werden. [60]

- **Header / Maximale Dateigröße:**
  Bei einigen Dateitypen können sowohl Footer, als auch Angaben zur Größe der Datei nicht vorhanden sein. In diesem Fall kann die maximale Größe verwendet werden. Hierzu wird erneut mit der Suche des Headers durch einen entsprechenden Marker begonnen, anschließend die maximale Größe dieser Datei genommen und der Bereich extrahiert. Der Nachteil hierbei besteht darin, dass irrelevante Informationen ebenfalls herausgefiltert werden. Aus diesem Grund muss nachträglich noch ein Bereinigen der nicht notwendigen Daten erfolgen. [65]

2.4.2 Bifragment Gap Carving

Eine Weiterentwicklung des signaturbasierten Carvings stellt das Bifragment Gap Carving dar. Es arbeitet nach einem ähnlichen Prinzip, kann jedoch fragmentierte Dateien verarbeiten. Allerdings ist die Anzahl der vorliegenden Fragmente auf zwei reduziert. Dieser Fakt ist auf die Struktur des Prinzips zurückzuführen. Es bedient sich ebenfalls des Faktes, dass Anfang und Ende bei bestimmten Dateitypen durch jeweilige Marker gekennzeichnet sind. Nach diesen wird zu Beginn gefiltert und der gesamte Bereich extrahiert. Auf diese Weise erhält man die zwei Dateifragmente mit anderen nicht zugehörigen Daten dazwischen. Damit die Fragmente ohne irrelevante Daten wieder zusammengesetzt werden können, wird der Bereich zwischen Header und Footer so lange variiert, bis die Datei dekodierbar ist. Hierfür ist ein Prüfalgorithmus zuständig. [24]
Bei großen Lücken zwischen den Fragmenten kann diese Methode des Carvings viel Zeit in Anspruch nehmen, weshalb sie sich eher für Szenarien eignet, in denen zwischen den zwei Fragmenten nur eine kleine Lücke existiert [60]. Für den Normalfall eignet sich diese Methode aus dem vorher genannten und einem zusätzlichen Grund nicht. Das Grundprinzip geht neben der Annahme einer kleinen Lücke auch davon aus, dass die zwei zueinander gehörenden Fragmente nebeneinander liegen. Allerdings muss damit gerechnet werden, dass eine andere Datei desselben Typs zwischen den Fragmenten liegen kann. Es fehlt somit eine Überprüfung der Zugehörigkeit der beiden Fragmente.

2.4.3 Blockbasiertes Carving

Das blockbasierte Carving ist eine Herangehensweise, welche sich auf weitere Carving-Methoden stützt und im engeren Sinne kein eigenständiges Prinzip des Carvings ist. Ihm liegt zugrunde, dass eine Fragmentierung nur an der Clustergrenze auftreten kann und ein Cluster nur eine Datei beinhalten kann [18]. Der Begriff blockbasiertes Carving ist mit clusterbasiertem Carving gleichzusetzen, wurde an dieser Stelle allerdings übernommen, da in der Literatur von Block die Rede ist.
Das Vorgehen hinter dieser Methode beginnt mit dem Klassifizieren der einzelnen Cluster. Hierzu wird der Dateityp eines Clusters bestimmt, ausgehend davon, dass ein Cluster lediglich zu einer Datei gehören kann. Die Klassifizierung erfolgt zum Beispiel durch das Suchen spezieller, dateitypischer Marker oder anderer Eigenschaften, die Aufschluss auf den Dateityp geben können. [60] [63]
Viele Methoden machen sich dieses Prinzip, clusterweise vorzugehen, zunutze.

2.4.4 Dateistrukturbasiertes Carving

Das dateistrukturbasierte Carving, auch bekannt unter den Namen *Deep Carving* und *Semantic Carving*, ist eine abgewandelte Variante des signaturbasierten Carvings. Neben der Suche nach den Header- und Footer-Markern wird zusätzlich die interne Dateistruktur für die Identifizierung des Dateityps hinzugezogen. [64]
Zu Informationen, die auf die Dateistruktur zurückzuführen sind, zählen beispielsweise bekannte Zeichenketten zur Identifizierung des Dateityps oder spezielle Byte-Sequenzen [24]. Der Dateityp wird, auf Grundlage des blockbasierten Carvings, für jedes Cluster bestimmt. Durch einen Abgleich der Erkennungsmerkmale innerhalb des zu identifizierenden Clusters kann eine Klassifizierung erfolgen. Im Anschluss kann die Dateistruktur ebenfalls Aufschluss darüber geben, in welcher Reihenfolge die einzelnen Teile angeordnet werden müssen.

2.4.5 Inhaltsbasiertes Carving

Dem inhaltsbasierten Carving liegt das blockbasierte Carving und deren Annahmen, dass ein Cluster nur zu einer Datei gehören kann, zugrunde [61]. Die vorliegenden Rohdaten werden in ihre Cluster aufgeteilt und anschließend wird jedem Cluster ein bestimmter Dateityp zugeordnet. Auf welche Weise die Bestimmung des Dateityps erfolgt, also die Klassifizierung der einzelnen Cluster auf Grundlage des Dateiinhalts, hängt von dem verwendeten Ansatz ab.

- **Häufung von Zeichen:**
  Eine Möglichkeit den Dateitypen anhand des Inhalts zu bestimmen ist das Zählen des Vorkommens bestimmter Zeichen. Beispielsweise lassen viele ASCII-interpretierbare Zeichen darauf schließen, dass eine Textdatei vorliegt und die Zugehörigkeit des Clusters zu einer Bild-, Multimedia- oder Videodatei mit sehr hoher Wahrscheinlichkeit ausgeschlossen werden kann. [66]

- **Sprachanalyse:**
  Ebenfalls hilfreich kann in einigen Fällen auch das Betrachten der Sprache sein. Im Besonderen, wenn bereits festgestellt wurde, dass eine Textdatei vorliegt, kann ein Analysieren der Sprache dabei helfen, zusammengehörende Cluster aufzufinden. Existieren beispielsweise zwei Cluster in französischer Sprache und ein weiteres kann der deutschen Sprache zugeordnet werden, kann davon ausgegangen werden, dass die französischen Cluster zusammengehören. [66]

- **Vorkommen bestimmter Byte-Werte:**
  Eine weitere Methode für die Kategorisierung der Cluster kann das Bestimmen der Häufigkeit des Auftretens bestimmter Byte-Werte sein. Das wiederholte Vorkommen von Byte-Werten/-Sequenzen ist bei bestimmten Dateiformaten üblich. Neben dem Auftreten bestimmter Byte-Werte kann auch die Kreuzkorrelation betrachtet werden. Hierfür muss ermittelt werden, welche Byte-Werte bei welchen Dateitypen häufig in Kombination miteinander auftreten. Dies kann anschließend mit den vorliegenden Clustern abgeglichen werden. [62]

- **Entropie:**
  Besonders hilfreich bei der Suche nach Multimedia-Dateien, wie Bildern, ist die Methode der Berechnung der Informationsentropie [66].
  Als Informationsentropie oder auch kurzgefasst Entropie wird der mittlere Informationsgehalt bezeichnet. Bezieht man diese Definition auf Dateien, ist damit die Wahrscheinlichkeit des Vorkommens der einzelnen Zeichen gemeint. Strukturierte Dateien bei denen nur eine Auswahl an bestimmten Zeichen auftritt, was bei Textdateien der Fall ist, besitzen einen niedrigen Entropiewert. Im Gegensatz dazu sind beispielsweise Bilddateien unstrukturiert, wodurch sie einen hohen Entropiewert besitzen. Diese Eigenschaft ist förderlich für das Ermitteln und Ausschließen bestimmter Dateitypen. [62]
  Die Formel zur Berechnung der Entropie (H) lautet wie folgt:

$$H = -\sum_{i}^{n} p_i \cdot log_2 \cdot p_i \tag{2.3}$$

  Die Variable n steht für die Anzahl der Zeichen und pi für die Wahrscheinlichkeit des Auftretens eines bestimmten Zeichens. [67]

2.4.6 Graphbasiertes Carving

Die bisher vorgestellten Carving-Algorithmen sind dazu da, den Clustern eines Speicherabbildes den passenden Dateitypen zuzuordnen. Während beim Bifragment Gap Carving und signaturbasierten Carving bereits Theorien zum korrekten Zusammensetzen der Cluster vorgestellt werden, fehlt dies bei anderen Carvern. Der Schritt der Bestimmung der Dateizugehörigkeit der einzelnen Cluster geschieht mit Hilfe von graphbasierten Carvern. Bei diesen wird das Zusammensetzungsproblem der einzelnen klassifizierten Cluster als Graph betrachtet. Jedes Cluster stellt innerhalb eines Graphen einen Knoten dar, welcher über Kanten mit den anderen Knoten verbunden ist. Den Kanten wird eine Gewichtung zugeordnet, welche die Wahrscheinlichkeit der Zusammengehörigkeit der beiden Cluster repräsentiert. [26] [60]
Im Folgenden werden Methoden zur Abarbeitung des Graphen nach dem Grundprinzip, dass jedes Cluster nur zu einer Datei gehören kann, vorgestellt.

- **Parallel Unique Path (PUP):**
  Bei dem PUP-Algorithmus wird mit dem Auffinden der Header vorgegangen. Anschließend wird mit dem ersten Header begonnen und ein Cluster ausfindig gemacht, welches die geringsten Unterschiede zu dem Header aufweist. Dieses wird im Anschluss an den Header angehängt und mit dem Suchen nach dem nächsten am besten passenden Cluster weitergemacht. Alle angehängten Cluster können nicht erneut verwendet werden und werden daher aus der Menge der verfügbaren Cluster entfernt. Ist ein Bild fertig rekonstruiert, wird mit dem nächsten Header weitergemacht. [60] [26]

- **Shortest Path First (SPF):**
  Der SPF-Algorithmus sucht nach den durchschnittlich geringsten Kosten eines Pfades. Es wird davon ausgegangen, dass auf diese Weise die besten Ergebnisse erzeugt werden. Im Gegensatz zum PUP sind für die erste Rekonstruktion alle Cluster für alle Dateien uneingeschränkt verfügbar. Die Dateien werden anhand der berechneten durchschnittlichen Pfadkosten zusammengesetzt. Um diese zu bestimmen muss die Summe der Gewichte der Kanten zwischen den Knoten einer rekonstruierten Datei durch die Gesamtanzahl der Knoten geteilt werden. Wenn dies für alle Dateien geschehen ist, wird lediglich die Datei mit dem geringsten Wert endgültig wiederhergestellt. Die Cluster, aus welchen sich das Bild zusammensetzt, werden aus der Menge genommen und alle Dateien mit diesen Clustern müssen neu zusammengesetzt werden. [60]

- **Sequential Hypothesis Test-PUP (SHT-PUP):**
  Eine Weiterentwicklung des PUP-Algorithmus stellt der SHT-PUP-Algorithmus dar. Er bedient sich der Erkenntnis, dass Cluster, die aufeinanderfolgen, mit einer sehr hohen Wahrscheinlichkeit auch zur selben Datei gehören. Zur Überprüfung dieser Annahme wird eine Hypothese entwickelt. Gehört das nächste Cluster zur Datei, ist die Hypothese H_0 erfüllt, gehört es nicht dazu, ist die Hypothese nicht erfüllt H_1. Der Algorithmus überprüft folgend anhand eines Vergleichswertes, ob die Hypothese erfüllt ist. Der Vergleichswert muss zuvor mit Testdaten berechnet werden, indem die Diskontinuität benachbarter zusammengehöriger Cluster bestimmt und daraus ein Mittelwert gebildet wird. Je mehr Testdaten verwendet werden, desto genauer ist dieser Referenzwert. [68]

Nachdem mehrere Methoden zum Umgang mit einem Graphen bestehend aus den einzelnen Clustern als Knotenpunkte vorgestellt wurden, muss noch geklärt werden, wie die Berechnung der Kantengewichtung erfolgt. Auch hierzu gibt es eine Vielzahl an Möglichkeiten mit unterschiedlicher Erfolgsaussicht.

- **Prediction by Partial Matching:**
  Bei Textdateien lässt sich eine Gewichtung anhand des *Prediction by Partial Matching* bestimmen. Hierbei werden die Zeichen am Endstück und Beginn des Clusters betrachtet und es wird ermittelt, mit welcher Wahrscheinlichkeit ein bestimmter Buchstabe auf den anderen folgt. Dieses Prinzip ist für Text-, jedoch nicht Multimediadateien zu empfehlen. [60]

- **Histogramm Intersection:**
  Diese Methode ist für die Rekonstruktion von Bilddateien ausgelegt und deshalb für beispielsweise Textdateien eher ungeeignet. Die *Histogramm Intersection* arbeitet, wie der Name schon vermuten lässt, mit einem Histogramm, in dem die Verteilung der Bildwerte von zwei Bildfragmenten verglichen wird. Bei ähnlichen Bildern ist weist auch das Histogramm wenige Unterschiede der Gewichtungen auf. Sucht man also zusammengehörige Fragmente, sollten die Diversitäten nur sehr gering sein. Hat man nur sehr kleine Bildfragmente oder möchte clusterweise vorgehen, bietet sich dieses Prinzip wegen zu geringer Datenmengen zum Vergleich nicht an. Auch fehlende Fragmente können für dieses Prinzip problematisch werden, da sie Einfluss auf die anderen bezüglich der Farbe oder Position haben können. [26]

- **Pixelvergleich:**
  Der sogenannte Pixelvergleich ist ebenfalls nur für Bilddateien entwickelt worden. Beim Pixelvergleich werden die Pixelwerte an den Schnittkanten von zwei Bilddateien verglichen. Hierzu wird zu Beginn die Breite des Bildes aus dem Bildheader extrahiert und anschließend die Pixel entsprechend der Breite des Bildes verglichen. Beim ersten Fragment werden die Pixel am Ende des Fragments genommen und beim Fragment, welches mit diesem verglichen wird, werden die ersten Pixel verglichen. [68]
  Für die Visualisierung wurde in Abbildung 2.12 ein Vergleich zwischen sechs Pixeln des Basisfragments mit einem Vergleichsfragment dargestellt. Durch den Vergleich der Pixel zweier Bildfragmente erhält man einen Wert, der für die Kantengewichtung genutzt wird. Verfahren für den Vergleich der Pixel sind *Sum Of Differences Summe der Unterschiede (SoD)*, *Euclidian Distance Euklidische Distanz (ED)* und *Coherence of Euclidean Distance Kohärenz der Euklidische Distanz (CED)*.

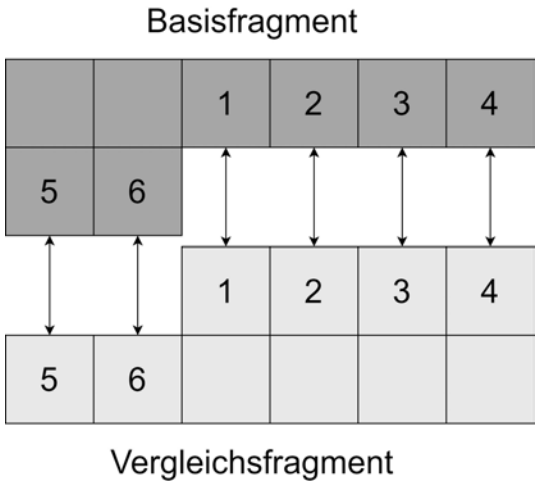


Abbildung 2.12: Veranschaulichung des Vergleichs von zwei Fragmenten anhand des Pixel Vergleich Algorithmus nach A. Pal und N. Memon [19]

Summe der Unterschiede (SoD):
Beim SoD werden die Pixel anhand ihrer RGB-Werte verglichen. Für das Beispiel in 2.12 würde man zuerst die RGB-Werte des Pixels 1 des Basisfragments mit den Werten des Pixels 1 des Vergleichsfragments vergleichen. Da ein Pixel aus drei Werten besteht, wird ein Pixel als Vektor angesehen und ein Vergleich der beiden Vektoren der beiden Pixel verglichen. Dieses Vorgehen wird für alle Pixel, in diesem Beispiel sechs Pixel, gemacht und anschließend ein Durchschnitt aus dem Vergleich aller Pixel-Vektoren berechnet. Das erhaltene Ergebnis wird als Gewichtung der Kanten verwendet. [26]

$$SoD = \frac{1}{n}\sum_{i=1}^{n}|x_i - y_i| \tag{2.4}$$

Euklidische Distanz (ED)
Die euklidische Distanz wird zur Bestimmung des Abstands- oder Ähnlichkeitsmaßes beispielsweise für Vektoren genutzt. Da die Pixel wie bei SoD als Vektoren betrachtet werden, bietet sich diese Methode an. Das Vorgehen bei der Berechnung der Euklidischen Distanz ist ähnlich wie beim SoD-Prinzip. Die RGB-Werte beider Fragmente werden miteinander verglichen, jedoch zuvor zum Quadrat genommen und daraus im Anschluss die Wurzel gezogen, wie man anhand der unten aufgeführten Formel erkennen kann. [26] [69]

$$ED = \frac{1}{n}\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2.5}$$

Kohärenz der Euklidische Distanz (CED)

Eine Weiterentwicklung der Euklidischen Distanz stellt die Berechnung des CED-Wertes dar. Er soll die Ergebnisse verbessern und mögliche Fehlerquellen ausgleichen. Hierzu werden nicht nur die zwei Bildzeilen der beiden Fragmente verglichen, sondern es wird eine dritte Bildzeile hinzugezogen um so ein Variationsmuster zu betrachten. Für die Berechnung des CED-Wertes müssen zuerst zwei ED-Werte bestimmt werden. Der erste ED-Wert wird als *EDnearby* bezeichnet und aus der Bildzeile an der Fragmentationsgrenze und der Zeile vor dieser berechnet. Dieser wird mit dem *EDboundary* verglichen, welcher sich aus den Pixelwerten an den Grenzen der zwei zu vergleichenden Fragmente zusammensetzt. [69]

In Abbildung 2.13 ist die Gewinnung der Werte graphisch dargestellt.

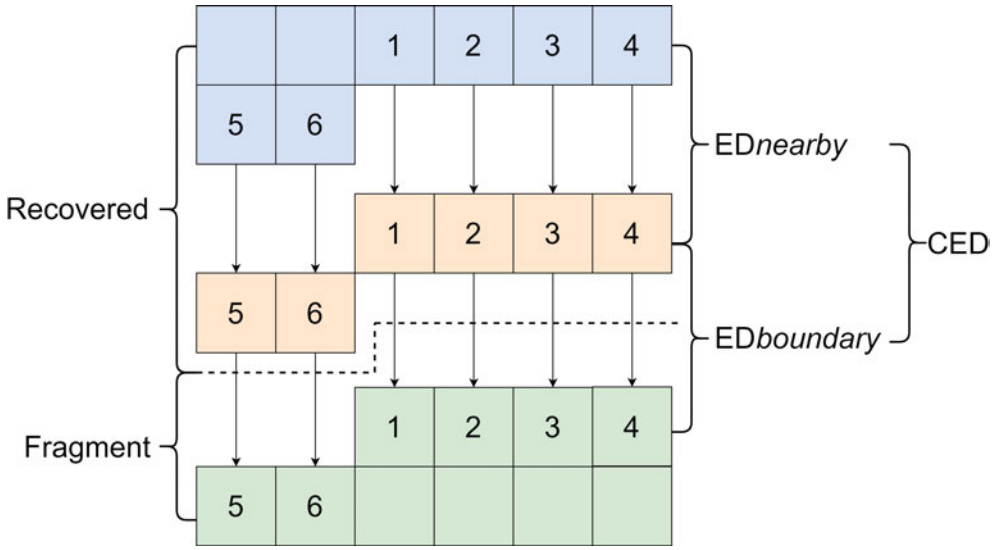


Abbildung 2.13: Veranschaulichung der zu vergleichenden Elemente für die Berechnung des CED-Wertes nach Y. Tang et al. [69]

Für die Berechnung werden die benötigten ED-Werte nach Formel 2.5 berechnet und im Anschluss für die Berechnung des CED-Wertes nach Formel 2.6 weiterverarbeitet. [69]

$$CED = \left| ED_{boundary} - ED_{nearby} \right| \tag{2.6}$$

2.4.7 Smart Carving

Das Smart Carving ist eine Zusammenführung der bisher kennengelernten Carving-Algorithmen und ist für das Wiederherstellen fragmentierter Dateien unabhängig von der Häufigkeit der Fragmentierung zuständig. Durch das Vereinen der diversen Carving-Algorithmen wie dem inhaltsbasierten und signaturbasierten Carving, soll ein besseres Ergebnis erzielt werden [60].
Des Weiteren deckt das Smart Carving neben der Klassifizierung der einzelnen Fragmente, auch die anschließende Zusammensetzung ab und kann als eigenständiger Algorithmus verwendet werden. Es wird in die Phasen Vorverarbeitung (*Preprocessing*), Klassifizierung (*Collating*) und Zusammensetzung (*Reassembly*) unterteilt, welche im Folgenden genauer beschrieben werden [62].
Eine Übersicht über den Zusammenhang der einzelnen Phasen ist in Abbildung 2.14 zu erkennen.

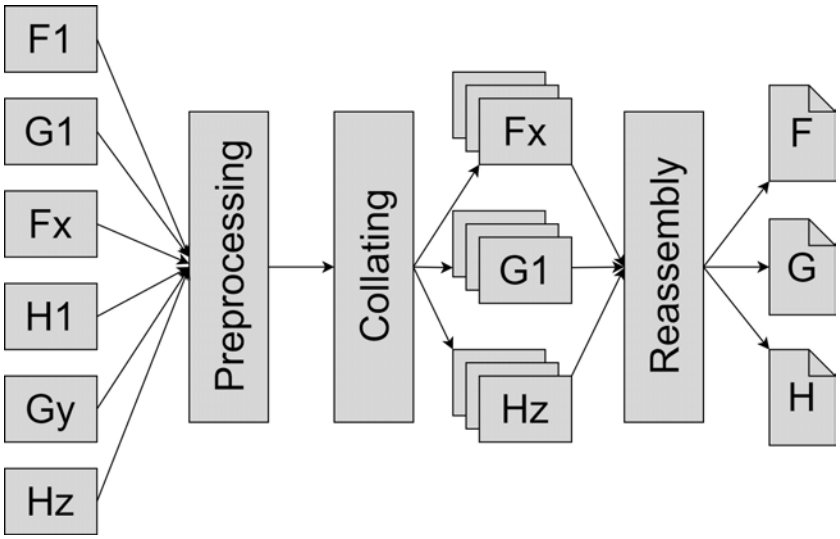


Abbildung 2.14: Phasen des Smart Carving Algorithmus nach A. Pal und N. Memon [19]

1. **Preprocessing**
   Die Preprocessing-Phase dient der Vorbereitung der Daten für die weitere Verarbeitung. Beispielsweise wird hierfür das Laufwerk, sofern notwendig, entschlüsselt und entkomprimiert. Um die Datenmenge zu reduzieren, werden an dieser Stelle außerdem Daten entfernt, welche sich im zugewiesenen Bereich befinden, also Daten, welche einen Eintrag im Dateisystem haben. Ist das Dateisystem nicht mehr vorhanden, wird die gesamte Festplatte als nicht zugewiesen angesehen. [62]
   Neben diesen Aufgaben kann die Preprocessing-Phase auch zur anderweitigen Verarbeitung genutzt werden. Es ist lediglich festgelegt, dass eine Vorbereitung für die folgenden Phasen erfolgt, was je nach Fall anders ausgelegt werden kann.

2. **Collating**
   Die Collating Phase dient dazu, die Festplatte in Cluster zu unterteilen und die einzelnen Cluster zu analysieren. Hierfür wird jedes Cluster anhand des Dateityps klassifiziert und Cluster desselben Typs gruppiert. Die Bestimmung des Dateityps eines Clusters geschieht mit Hilfe der bereits vorgestellten Carving Methoden. Es kann in den Clustern nach bekannten Byte-Sequenzen gesucht werden, nach Schlüsselwörtern, die für einen bestimmten Dateitypen relevant sind oder auch die Anzahl der ASCII-Zeichen kann gezählt werden. An dieser Stelle bietet sich ebenfalls das Abgleichen der Entropiewerte an, um den Dateityp einzugrenzen. Des Weiteren besteht die Möglichkeit, die Änderungsrate der Byte-Werte, den sogenannten *Rate-of-Change*, hinzuzuziehen. [60] [62]
   Als Ergebnis der Collating-Phase sollte eine Sammlung an Clustern stehen, die einem bestimmten Dateitypen auf Grundlage diverser Eigenschaften zugeteilt wurden. Je nach Ziel des Smart Carvings können auch nur Cluster eines speziellen Dateityps zurückgeliefert und irrelevante Cluster entfernt werden.

3. **Reassembly**
   Nachdem alle Cluster klassifiziert sind, müssen diese als Datei wieder zusammengesetzt werden. Hierbei besteht die Herausforderung darin, dass mehrere Fragmente von Dateien desselben Typs existieren und korrekt zugeordnet werden müssen. Außerdem muss beim Kombinieren der Einzelteile auf die richtige Reihenfolge der Fragmente geachtet werden. Diese Aufgaben laufen beim Smart Carving in der Reassembly-Phase ab.
   Zu Beginn werden die Cluster mit den Headern herausgearbeitet und anschließend die anderen Fragmente dem passenden Header zugeordnet. Um zugehörige Fragmente zu finden, werden Methoden des graphbasierten Carvings verwendet. [60] [62]
   Die Wahl der verwendeten Methoden bestimmt die Qualität der zurückerhaltenen Ergebnisse. Es bietet sich an, eine Kombination aus mehreren Methoden zu verwenden.

3 Methodik

Das Kapitel Methodik beschreibt den Entwicklungsprozess der prototypischen Implementierung eines Programms zur Wiederherstellung sowohl fragmentierter als auch unfragmentierter JPEG-Dateien. Zu diesem Zweck wird auf die zuvor behandelten Grundlagen zurückgegriffen und diese zum Teil für das zu behandelnde Problem angepasst.
Im besonderen Maße wird auf die durch den JPEG-Standard festgelegten Schritte, Phasen und Marker eingegangen. Daran orientiert wird ein erster Prototyp entwickelt, der nach dem Smart Carving-Prinzip aufgebaut ist. Im Folgenden werden die einzelnen Funktionen, Methoden und Anwendungen des entwickelten Programms zur Übersichtlichkeit und für ein besseres Verständnis in die drei Phasen des Smart Carvings eingeordnet und andere hinzugezogene Carving-Ansätze erläutert.
Des Weiteren folgt eine kurze Anleitung zur korrekten Anwendung des Prototyps mit einer Auflistung von Grenzen, die bei seiner Verwendung berücksichtigt werden müssen. Anschließend wird die Entwicklung diverser Testszenarien zum Evaluieren der Lauffähigkeit des Prototyps beschrieben. Auf diese folgt das letzte Unterkapitel *Evaluationsstrategien*, welches ein Prinzip für die Bewertung der Ergebnisse aus den Testszenarien darstellt.

3.1 Architektur Prototyp

Im Folgenden wird die Architektur des Prototyps, welcher den Namen `Valkyra` trägt, beschrieben, wobei aus Verständlichkeitsgründen nicht im Detail auf den Programmcode eingegangen wird. Die Struktur des Prototyps lässt sich in die drei Hauptbereiche des bereits kennengelernten Smart Carvings einteilen. Im Folgenden werden diverse Klassen, Funktionen und Methoden in die Aufgabenbereiche *Preprocessing, Collating, Reassembly* eingeteilt. Abbildung 3.1 stellt die einzelnen Schritte des Smart Carvings dar, abgewandelt für den entwickelten Prototyp. Zur erkennen ist, dass die Preprocessing-Phase ein Speicherabbild (*Image*) erhält, dieses in Cluster aufteilt, durch die Collating-Phase JPEG-Cluster gefiltert und diese in der Reassembly-Phase wieder zusammengesetzt werden. Eine ausführlichere Beschreibung folgt in den Unterkapiteln.

Der Prototyp besteht im Gesamten aus sechs Dateien, welche nachfolgend detaillierter beschrieben werden. In Abbildung 3.2 sind alle Dateien abgebildet, wobei Kästchen als Symbol für Klassen stehen, während Dateien die lediglich Funktionen beinhalten als Kreise dargestellt werden. Zusätzlich dazu steht das Dokumentensymbol, verwendet für die `Static.py`, für ein Dictionary. Die Pfeile zeigen, welche Dateien auf Funktionen, Methoden oder andere Informationen einer anderen Datei zugreifen.

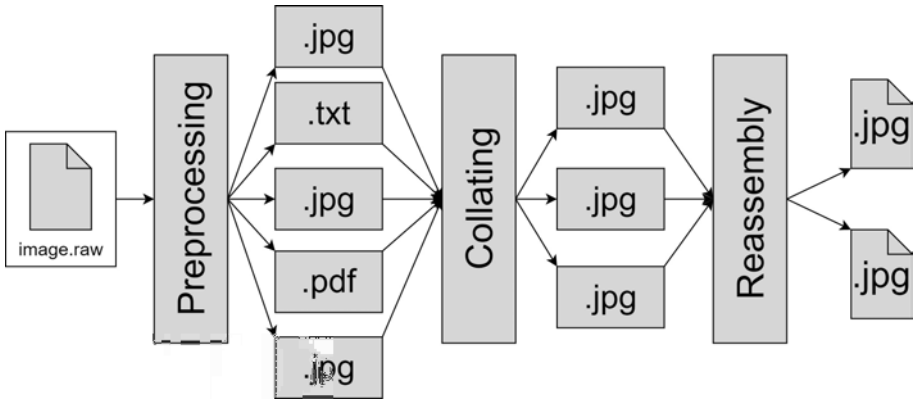


Abbildung 3.1: Phasen des Smart Carving Algorithmus angepasst auf die Abläufe des Prototyps

Das Modul, über welches das Programm ausgeführt wird, ist `main.py`. In diesem Modul ist festgehalten, welche Informationen zum fehlerfreien Starten des Prototyps mitgeliefert werden müssen. Eine Information, ohne die nicht gestartet werden kann, ist das Speicherabbild, welches fragmentierte oder unfragmentierte JPEG-Dateien zum Wiederherstellen beinhaltet. Zusätzlich zu dieser Information kann auch eine Clustergröße übergeben werden, was jedoch nicht zwingend erforderlich ist. Ohne eine mitgelieferte Größe der Cluster wird von einem Standard-Wert, welcher sehr häufig vorkommt, von 4096 Bytes pro Cluster ausgegangen [18].

In der `main`-Funktion findet des Weiteren eine Überprüfung der Existenz der ange-

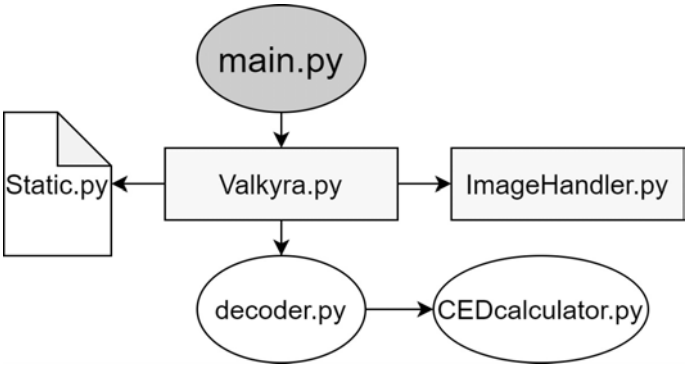


Abbildung 3.2: Schematische Darstellung der zusammenhängenden Programmdateien des Prototyps

gebenen Datei statt. Außerdem werden in dieser alle Kommandozeilenparameter, sowie die Beschreibung des Programms festgelegt. Die `main`-Methode startet die Klasse `Valkyra`, welche alle relevanten Schritte enthält und alle Dateien zusammenführt. Diese greift zum Laden und zur Verarbeitung des Speicherabbildes auf die ausgelagerte Klasse `ImageHandler` zu. In Abbildung 3.3 sind beide Klassen mit ihren jeweiligen Methoden in einem Klassendiagramm dargestellt.

Abbildung 3.3: Klassendiagramme der Klassen `Valkyra` und `ImageHandler` des Prototyps

In der Klasse `ImageHandler` werden Methoden definiert, die das Speicherabbild anhand der Clustergröße lädt und diese verarbeitet. Damit nicht immer das gesamte Speicherabbild geladen werden muss, werden die Cluster anhand von Indexen allokiert, welche mit Hilfe der Methode `getClusterByNumber` aufgerufen werden.
Jedem Cluster wird so ein Index, wie in der Informatik üblich, beginnend bei 0, zugeordnet. Bei einer Speicherabbildgröße von 40.960 Bytes mit einer durchschnittlichen Clustergröße hat man zehn Cluster allokierbar durch die Indexe von 0-9.
In den nächsten Abschnitten werden das Vorgehen und die einzelnen Klassen anhand des Smart-Carving Vorgehens genauer beschrieben.

3.1.1 Preprocessing

Die Preprocessing Phase dient im Allgemeinen der Vorbereitung des Speicherabbildes für die weitere Analyse und Verarbeitung. Sie erfüllt neben notwendigen Schritten, um die Daten einer Festplatte zugänglich zu machen, auch Aufgaben zur Reduzierung der Datenmengen. In der Literatur werden dieser Phase zum Beispiel die Aufgaben der Entschlüsselung der Festplatte und das Herausfiltern allokierter Bereiche zugeordnet [18].

Diese allgemein gefasste Beschreibung findet in der Preprocessing-Phase des vorliegenden Carving-Prototyps keine Anwendung. Da das Entschlüsseln einer Festplatte eine aufwändige, mühsame und komplizierte Aufgabe darstellt und der Fokus der Arbeit nicht auf der Entschlüsselung liegt, wird der Fall eines Vorkommens einer verschlüsselten Festplatte bewusst außer Acht gelassen.
Des Weiteren soll der Carving-Algorithmus so konzipiert sein, dass keine Abhängigkeit zum Datei- oder Betriebssystem besteht. Auch ein Fehlen des Dateisystems soll kein Problem bei der Rekonstruktion von den JPEG-Dateien innerhalb der Speicherabbildes darstellen. Diese aufgeführten Gründe legen nahe, dass die, in der Literatur häufig anzutreffenden Aufgaben der einzelnen Phasen, in dem vorliegenden Prototyp keine Relevanz haben werden.
Das Vorbereiten des Speicherabbildes, welches grundlegend Aufgabe der Preprocessing-Phase ist, erfolgt hier auf andere Weise.

Zur Vorbereitung in diesem speziellen Fall zählt zu Beginn das Entgegennehmen von Informationen und deren anschließende Verarbeitung. Diese Informationen sind beim Prototyp die Datei, welche am angegebenen Pfad liegt und optional eine Clustergröße. Diese Informationen werden in Attributen gespeichert, die für alle Methoden innerhalb der Klasse zugreifbar sind. Zuvor wird geprüft, ob ein korrekter Pfad angegeben wurde, die Datei existiert und es sich bei dieser wirklich um eine Datei handelt. Des Weiteren wird überprüft, ob eine formal korrekte Clustergröße mitgeliefert wurde oder die als Standard festgelegte Größe genutzt werden muss.
Anschließend folgt die Hauptaufgabe der Preprocessing-Phase, das Unterteilen der zu

analysierenden Datei in seine Cluster. Dieser Vorgang beruht auf dem Prinzip des blockbasierten Carvings, welches eine Analyse auf Clusterebene vorsieht. Wie bereits in den Grundlagen festgehalten, ist ein Cluster die kleinste Einheit, die ein Betriebssystem ansteuern kann. Eine Datei kann somit nur nach einem Cluster fragmentiert sein. Mit diesem Wissen kann jedem Cluster ein Dateityp zugeordnet werden, was jedoch erst Aufgabe der nächsten Phase ist. Damit eine solche Klassifizierung erfolgen kann, wird die mittels Dateipfad übergebene Datei clusterweise eingelesen.
In der Klasse `ImageHandler` wird die Datei in der Methode `loadCurrentCluster` geladen. Ziel für den weiteren Verlauf soll es sein lediglich einzelne Cluster ansprechen zu können. Der Grund hierfür ist, dass ein Laden einer kompletten Datei kann zu Problemen führen, insofern der zur Verfügung stehende Arbeitsspeicher kleiner ist, als die zu ladende Datei. Dementsprechend wir die Datei clusterweise durch das Arbeiten mit Indexen eingelesen. Auf diese Weise wird nur das Cluster geladen, welches durch Index angesprochen wird. Den Aufruf eines Clusters über einen Index erledigt die Funktion `getClusterByNumber` im `ImageHandler`.
Die Aufgaben der Preprocessing Phase sind hiernach abgeschlossen und die Klassifizierung der Cluster kann folgen.

3.1.2 Collating

Das Ziel der Collating-Phase besteht in der Klassifizierung der einzelnen Cluster eines Festplattenabbildes. Die Aufgabe der Collating-Phase aus der Theorie unterscheidet sich in diesem Fall nicht von der des Prototyps.
Das Programm fokussiert sich auf das Wiederherstellen von JPEG-Dateien, weshalb bei der Clustertyp-Bestimmung geschaut wird, ob es sich um ein JPEG-Cluster oder nicht handelt. Es wird eine Falsch/Richtig Hypothese getroffen, ob ein JPEG-Cluster vorliegt oder nicht. Hierfür wird geschaut, ob ein Cluster JPEG typische Eigenschaften vorweist. Tut es dies nicht, wird es verworfen. Erfüllt es allerdings die Eigenschaften nach denen gefiltert wird, wird der Index des Clusters innerhalb einer Liste gespeichert. Auf diese Weise erhält man am Ende eine Liste an Indexen, welche im besten Fall lediglich Cluster des Typs JPEG enthalten. In Abbildung 3.4 ist der Prozess der Filterung der Cluster bildlich dargestellt. Die einzelnen Schritte werden folgend genauer erläutert.
Die gesamten Methoden der Collating-Phase befinden sich in der Klasse `Valkyra`. Für die Klassifizierung wird zu Beginn das signaturbasierte Carving durch die Suche nach bekannten irrelevanten Markern des weit verbreiteten Graphikformats *Portable Network Graphics* kurz *PNG* und relevanten JPEG-Markern hinzugezogen. Zu Beginn werden Cluster, welche den SOI Marker mit den Byte-Werten `0x89504E470D0A1A0A` am Anfang des Clusters enthalten aussortiert. Des Weiteren folgt eine Aussortierung der Cluster, die den EOI Marker mit der Byte-Sequenz `0x49454E44AE426082` enthalten. [70] Dieser Prozess läuft in der Methode `removePNGClusters` ab, indem in allen Clustern der Datei nach den hexadezimalen Werten des SOI Markers an den ersten acht Byte-Positionen gesucht wird.

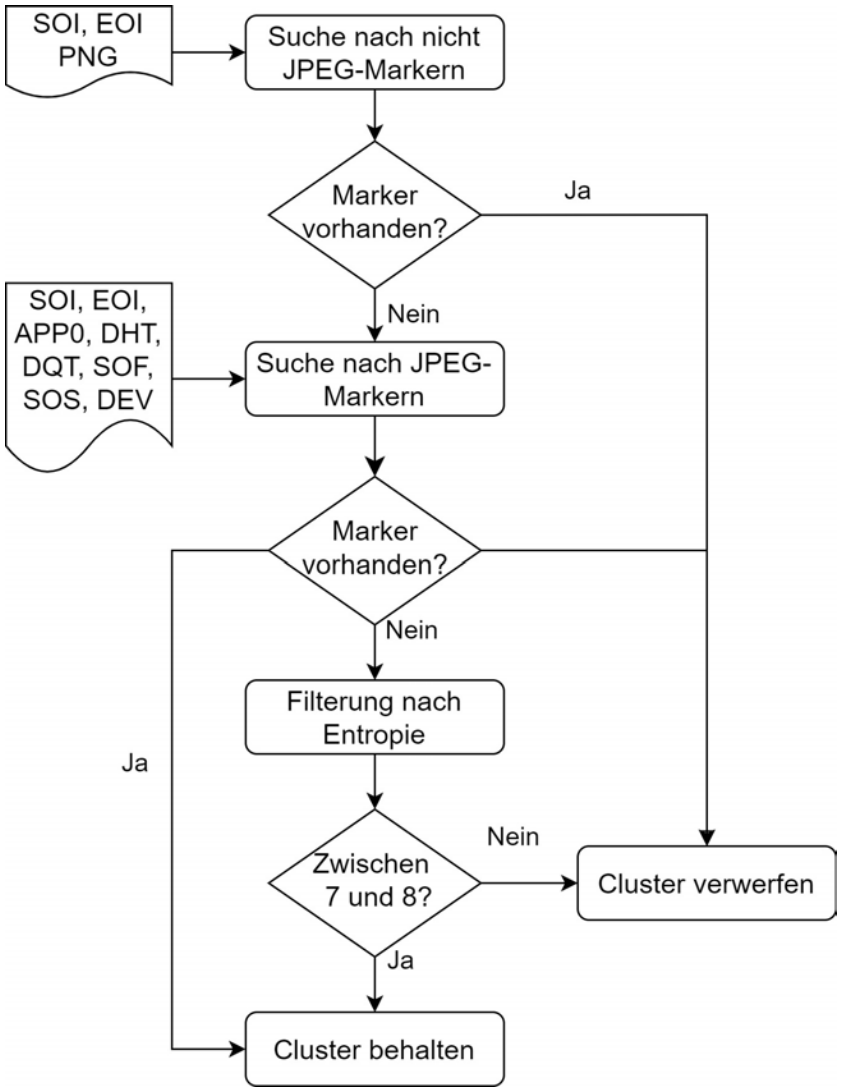


Abbildung 3.4: Ablauf der Filterung nach JPEG-Clustern innerhalb der Collating-Phase

Des Weiteren wird in allen Clustern nach dem Footer von PNG-Dateien gesucht. Sind beide Marker nicht enthalten, werden diese Cluster weiter gefiltert durch die Suche nach JPEG-Markern. Da diese JPEG-Marker im weiteren Verlauf häufiger Verwendung finden, wurden diese in einer extra Datei `Static.py` festgehalten und können durch einen Import aus allen Klassen aufgerufen werden. Hierzu wird ein Dictionary erzeugt, welches als Schlüssel die Abkürzung des Markers und als Value den Hexadezimalen-Wert dieses enthält. Neben dem SOI, EOI, APP0, DHT, DQT, SOF, SOS enthält es zusätzlich noch den Marker `0xFF00`, hier benannt als DEV.
Die Bezeichnung DEV ist eine Abkürzung für den englischen Begriff *devaluation* und bedeutet auf Deutsch *Entwertung*. Aus dem Grund, dass es sich bei `0xFF00` eigentlich nicht um einen richtigen Marker handelt, sondern das zufällige Vorkommen des Wertes `0xFF` durch `0x00` entwertet wird, wurde die Bezeichnung gewählt.

Der Marker wurde in die Marker-Liste zur Filterung aufgenommen, da ein Vorkommen dieses, Merkmal einer JPEG-Datei sein kann.
Die Suche der Marker wird durch die Methode `getIndicesOfMarkerFinds` realisiert. Alle Cluster ohne PNG-Marker des Speicherabbildes werden durch den jeweiligen Index über den `ImageHandler` geöffnet und es folgt eine Suche nach den gesamten Markern aus `Static.py`. Wird einer der Marker gefunden, wird der Index des entsprechenden Clusters einem Set angehängt. Die Verwendung eines Sets wurde an dieser Stelle gewählt, da es durch das mehrmalige Vorkommen der Marker zu Dopplungen kommen kann. Ob ein Marker mehrfach in einem Cluster vorkommt ist für die Weiterverarbeitung irrelevant. Wenn kein Marker gefunden wird, kann noch nicht ausgeschlossen werden, dass es sich um ein JPEG-Cluster handelt. Besonders in den eigentlichen Bilddaten kann ein Fehlen dieser Marker auftreten. Da ein Fehlen eines Clusters bei der Wiederherstellung ein größeres Problem bereitet, als irrelevante Cluster mit in die Reassembly-Phase zu nehmen, erfolgt eine weitere Untersuchung der Cluster.

Die Cluster die keinen PNG-Marker beinhalten werden noch einmal durch das inhaltsbasierte Carving untersucht und nach JPEG-Clustern gesucht. Eine Methode dieses Carving-Vorgehens ist die Einbindung der Entropie, dem mittleren Informationsgehalt. Der Vergleich von Entropiewerten bietet sich besonders bei der Suche nach Bilddateien an, da diese im Vergleich zu Textdateien einen hohen Entropiewert besitzen.
Die Methode `getEntropy` bekommt das bereits vorklassifizierte Set an Indexen übergeben, greift über den Index auf die Daten zu und berechnet diese anhand der Formel für die Berechnung der Entropie für jedes Cluster. Bevor die Formel angewendet werden kann, muss die Wahrscheinlichkeit des Auftretens für jedes Byte, also das Vorkommen der 256 Zeichen, berechnet werden. Dieser Wert wird durch die Länger der Daten geteilt und in die Formel eingesetzt.
Damit eine Filterung anhand des berechneten Entropiewertes erfolgen kann, muss ein Bereich in dem der Entropiewert liegen muss, festgelegt werden. In der Literatur sind Werte im Bereich von 7-8 zu finden, weshalb dieser für die Filterung gewählt wurde [62].
Die Filterung nach dieser übernimmt die Methode `filterClusterByEntropy`, welche in der `collate`-Methode aufgerufen wird. Diese Methode nimmt ebenfalls die nach den Markern gefilterten Indexe entgegen und erstellt ein set aus den Indexen, die durch die Marker oder den Entropiewert als JPEG klassifiziert wurden.
Nach diesem Vorgehen verbleibt ein Set, umgewandelt in eine Liste mit Indexen zu Clustern, in denen sich mit hoher Wahrscheinlichkeit alle JPEG-Cluster und vermutlich zusätzlich irrelevante falsch klassifizierte Cluster befinden.

Der nächste Schritt besteht darin, den Index des Header- und Footer-Cluster zu identifizieren. Durch den JPEG-Standard ist dies unkompliziert, da Start und Ende einer JPEG-Datei durch den SOI und EOI Marker eindeutig gekennzeichnet sind. Die Funktionen `getHeaderClusterIndices` und `getFooterClusterIndices` sind sehr ähnlich aufgebaut. Beide Funktionen greifen über die Liste der Indexe der klassifizierten Cluster auf die Daten zu und suchen in den Daten nach dem Vorkommen des ent-

sprechenden Markers. Einziger Unterschied beider Funktionen besteht darin, dass bei der Suche nach dem Footer die Stelle des Fundes innerhalb eines Clusters egal ist, während beim Header die ersten Byte-Werte eines Clusters mit dem SOI Marker übereinstimmen müssen. Das resultiert aus der Regel, dass der Marker des JPEG-Headers immer am Cluster-Anfang stehen muss. Der Footer hingegen kann an jeder Stelle des Clusters vorkommen, da ein Bild auch kleiner als ein Vielfaches der Clustergröße sein kann. Als Ergebnis der Suche wird eine Liste mit den Indexen von Clustern mit Footer und eine Liste mit Header-Clustern generiert.
Mit diesem Schritt ist die Collating-Phase abgeschlossen und es folgt die Reassembly-Phase.

3.1.3 Reassembly

Die Reassembly-Phase dient der Rekonstruktion und Zusammensetzung der klassifizierten Dateien. Für den Prototyp besteht die Aufgabe darin, aus den Ergebnissen der Collating-Phase die gefundenen JPEG-Dateien zu entnehmen und zusammengehörige Cluster in der korrekten Reihenfolge wieder zu verbinden. Die Reassembly-Phase läuft neben der Klasse `Valkyra` auch in den Funktionen von `decoder.py` und `CEDcalculator.py` ab. Eine Zusammenführung der einzelnen Funktionen und Methoden ist allerdings in `Valkyra`.
Als erste Aufgabe werden Segmente, eine Gruppierung von nebeneinander liegenden Clustern, gebildet. Dieser Schritt wird gemacht, da in der Literatur durch diverse Experimente die Erkenntnis erlangt wurde, dass Cluster, welche nebeneinander liegen mit einer hohen Wahrscheinlichkeit zur selben Datei gehören [19]. Da einzelne Cluster bei großen Pixelbreiten für einen Vergleich zu klein sind, wird im Verlauf mit den Segmenten gearbeitet.Für eine bessere Unterscheidung zwischen den Begriffen Cluster, Fragment und Segment dient Abbildung 3.5.

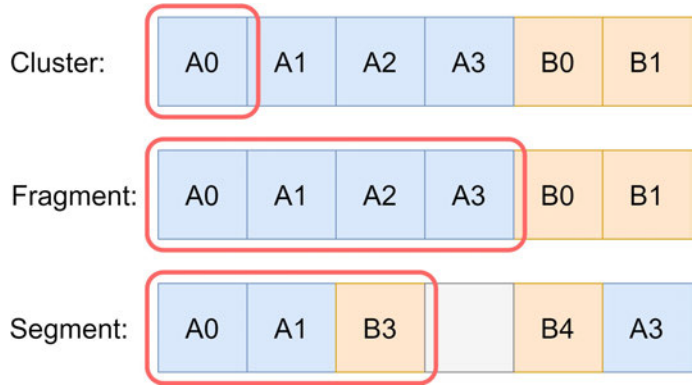


Abbildung 3.5: Schematische Darstellung der Unterscheidung der Begriffe Cluster, Fragment und Segment

Der Unterschied zwischen Fragment und Segment liegt darin, dass ein Fragment ein

Teil einer Datei ist, während ein Segment eine Zusammenfassung von mehreren nebeneinanderliegenden Clustern, unabhängig ihres Dateityps ist. Die Zusammenführung zu einem Segment kann zum Beispiel anhand von Lücken, wie in 3.5 geschehen.

Das Suchen nach dem besten Treffer zum Anhängen an ein Segment läuft nach dem Prinzip des Pixelvergleichs ab. Hierzu wird die Methode der CED-Berechnung genutzt, da es sich hierbei um einen aktuellen Ansatz handelt, welcher die Verbesserung bisher bestehender Ansätze ist [69].
Das Ablaufdiagramm 3.6 veranschaulicht die Schritte, die diese Phase durchläuft.

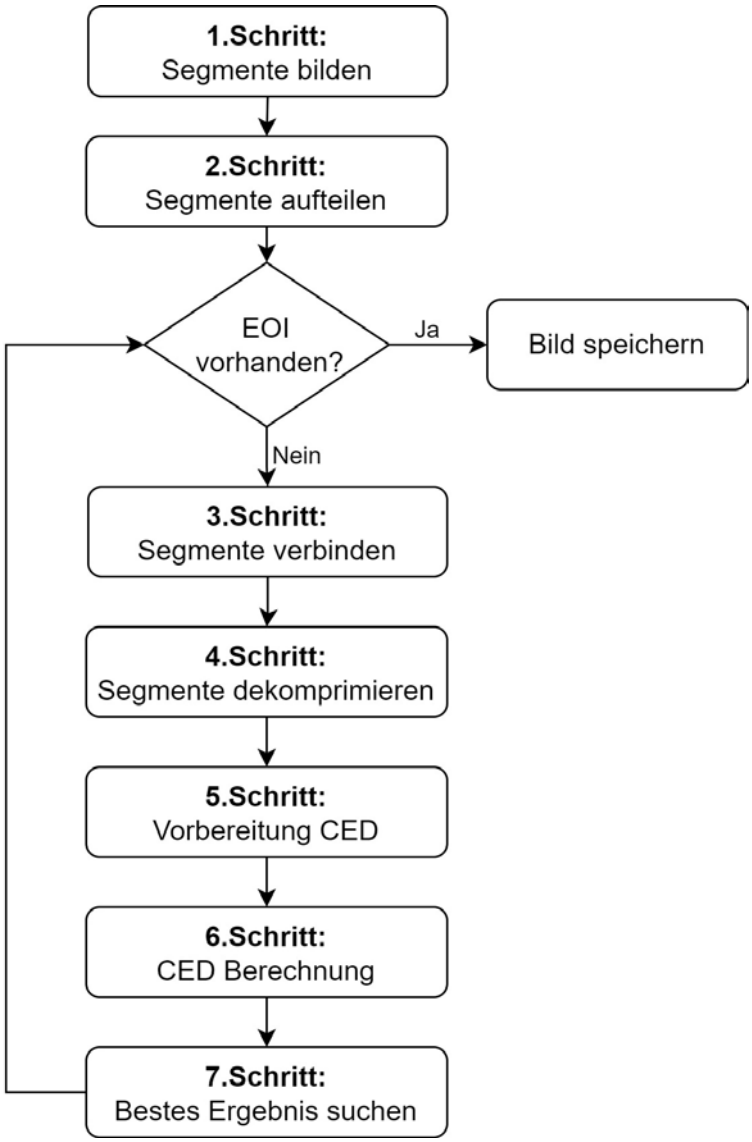


Abbildung 3.6: Ablauf der Schritte innerhalb Reassembly-Phase

1. **Segmente bilden**
   Für das Bilden von Segmenten ist die Methode `createSegments` zuständig. Die Segmente werden zuerst anhand ihrer Lücken durch die Methode `createSegmentsByGap` gebildet, indem Indexe, welche ohne Lücke aufeinanderfolgen, in einer Liste als Segment zusammengefasst werden. Beim folgenden Beispiel fehlen die Indexe 4 und 10, wodurch sich an diesen Stellen die Segmentgrenzen befinden und alle Indexe ohne Lücke als ein Segment zusammengefasst werden.

   $$0\,1\,2\,3\,|\,5\,6\,7\,8\,9\,|\,11\,12\,13$$

   Mit diesem Schritt ist die Segmentbildung noch nicht endgültig abgeschlossen. Es kann vorkommen, dass zwei JPEG-Bilder direkt aufeinanderfolgen oder ein Bild durch ein anderes unterbrochen ist. Um diese Fälle zu berücksichtigen sind die Methoden `createSegmentsByFooter` und `createSegmentsByHeader` da. Sie sind ähnlich aufgebaut und vergleichen die Übereinstimmungen der Indexe in den Listen für Header und Footer mit den bereits nach Lücken segmentierten Daten. Gibt es eine Übereinstimmung, wird im Fall eines Headers vor dem Index und beim Auftreten eines Footers nach dem Index, des Clusters in dem dieser vorkommt, abgeschnitten. Damit ist die Segmentbildung, welche in 3.7 beispielhaft abgebildet ist, abgeschlossen.

2. **Header Segment suchen**
   Nach der Segmentbildung müssen diese unterteilt werden in Segmente mit Header und solche ohne. Hierfür wird erneut nach dem Header-Marker in der Methode `getHeaderSegmentIndices` gesucht und alle Segmente, die diesen enthalten, werden an eine Liste `header_segment_indices` angehängt. Alle anderen Segment-Indexe werden in der Methode `getNonHeaderIndices` Teil der Liste `non_header_segment_indices`. Diese Unterscheidung muss erfolgen, da der Prototyp vom Header ausgehend nach einem Segment sucht, welches gemessen mit dem CED-Vergleich am besten zu dem entsprechenden Header passt.

3. **Segmente miteinander verbinden**
   Der nächste Schritt besteht darin, den Header-Segmente die Nicht-Header-Segmente nacheinander anzuhängen. Allerdings muss zuvor eine Überprüfung erfolgen, ob es sich bei dem Header-Segment bereits um ein vollständiges Bild handelt, weshalb in der Methode `searchForMatches` eine erneute Suche nach dem Footer erfolgt. Existiert dieser, wird das Bild direkt gespeichert. Wenn nicht, geht der Prototyp jeden Header in der Methode `reassembly` nacheinander durch und arbeitet diese in der Reihenfolge des Vorkommens in der Liste ab. Mit Hilfe der `mergeIndicesOfSegments` Methode werden die Indexe der Cluster des Header-Segments mit denen des Nicht-Header-Segments zusammengefügt. Anschließend werden die Daten von allen Indexen miteinander verbunden.

Bevor die Bilddaten, wie der nächste Schritt vorsieht, dekomprimiert werden können, muss übergangsweise ein Footer angehängt werden. Dies hängt damit zusammen, dass der verwendete Decoder für die Dekomprimierung die Marker des Footers vorsieht. Die Methode `appendTransitionalFooter` erledigt diesen Schritt.

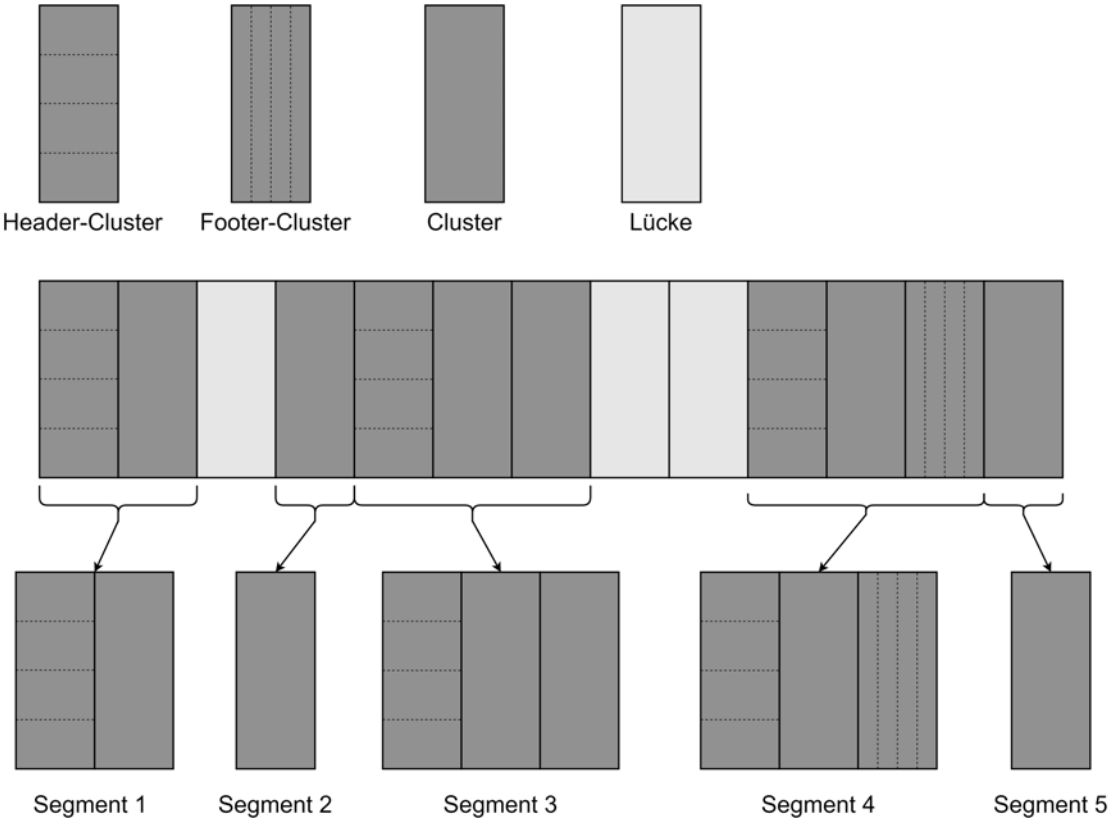


Abbildung 3.7: Prinzip zur Aufteilung der Cluster in Segmente

4. **Dekomprimieren**
   Die Daten der zusammengefügten JPEG-Cluster liegen komprimiert vor. Dies ist insofern ein Problem, da auf diese Weise kein Vergleich der Pixel erfolgen kann. Der Prototyp ist aber so konstruiert, dass er den graphbasierten Carving-Algorithmus des Pixel Vergleichs verwendet. Aus diesem Grund müssen die Daten durch die Informationen im Header dekomprimiert werden. Dieser Schritt wird durch eine bereits existierende Python Bibliothek `PIL` (*Python Imaging Library*), die Funktionen zum Dekodieren von JPEG-Bildern beinhaltet, durchgeführt. Die Bilddaten werden durch die Bibliothek nach dem JPEG-Standard dekodiert, indem die Schritte der JPEG-Kodierung rekursiv abgearbeitet werden. Falls Probleme bei der Dekodierung entstehen, was bei falscher Zusammensetzung möglich ist, wurde das Programm um eine Ausnahme ergänzt. Wenn diese eintritt, wird der Schritt übersprungen und mit dem nächsten Segment zur Dekodierung fortgesetzt.

5. **Vorbereitung zur CED-Berechnung**
   Aus dem vorherigen Schritt der Dekomprimierung erhält man die dekomprimierten Bilddaten, welche für einen Vergleich der einzelnen Pixel genutzt werden. Für den Vergleich der Pixel ist eine Kenntnis über die Bildbreite in Pixel und den Startpunkt der eigentlichen Bilddaten notwendig. Alle Vorbereitungen, sowie die CED-Berechnungen sind im `CEDcalculator` implementiert. Die Bildbreite wird im Header von JFIF-Dateien beim Start-Of-Frame gespeichert. Die Funktion `getImageWidth` sucht daher zuerst nach dem SOF Marker, geht zur Position, an der die Größe hinterlegt ist, formt sie von einem Hexa- in einen Dezimalwert um und extrahiert dann die Bildgröße. Da die Bilder durch den Decoder zurück in den RGB-Farbraum transformiert werden, muss die Bildbreite mit dem Faktor drei multipliziert werden. Dies liegt daran, dass sich ein Pixel aus den Farben rot, grün und blau zusammensetzt, welche jeweils durch ein Byte abgebildet werden.
   Die so berechnete Bildbreite wird dazu genutzt, die Bildzeilen miteinander zu vergleichen. Für den Vergleich wird der bereits vorgestellte CED-Vergleich genutzt, welcher drei Bildzeilen für die Berechnung miteinander vergleicht. Mit Hilfe einer Suche nach dem Marker für das Start-of-Image Segment und Berechnung seiner Länge, kann der Start der eigentlichen Bilddaten gefunden und die Daten extrahiert werden. Die Daten werden im Anschluss in Listenelemente der Größe einer Bildzeile gespeichert. Die Gesamtheit dieser Schritte ist in der Funktion `getRowData` umgesetzt.
   Hat das letzte Stück der Daten nicht die entsprechende Größe, wird es nicht für die Berechnung des CED-Wertes verwendet, da ein Vergleich mit der Zeile davor aufgrund unterschiedlicher Länge nicht erfolgen kann.
   Da ein Vergleich zwischen dezimalen Werten für den Anwender überischtlicher ist, als von Hexadezimalen, werden die Bildzeilen Byte für Byte in der Funktion `convertRowData` durch den Aufruf von `convertToInt` umgeformt. Mit diesem Schritt ist die Vorbereitung für die Berechnung des CED-Wertes abgeschlossen.

6. **CED-Wert ermitteln**
   Die Berechnung des CED-Wertes übernimmt die `getCEDFromData` Funktion innerhalb des `CEDcalculators`. In der Formel für diesen Wert ist festgehalten, dass der EDboundary und der EDnearby miteinander verglichen werden müssen. Hierzu müssen diese zu Beginn mit der allgemeinen Formel für den ED-Wert ausgerechnet werden. Die Funktion `getED` integriert allgemein die Formel zur Berechnung des ED-Wertes aus zwei Bildzeilen in den Programmcode. In der Funktion `getEDFromData` wird diese aufgerufen und für die übergebenen dekomprimierten Bilddaten eine Liste mit den berechneten Werten erzeugt.
   Eine weitere allgemein formulierte Funktion `getCED` wird durch die Funktion `getEDFromData` aufgerufen und die Liste mit den ED-Werten zur Berechnung übergeben. Es wird mit dem ersten Wert als EDboundary und dem zweiten Wert

als EDnearby gestartet. Anschließend wird der EDnearby zum EDboundary und der nächste ED-Wert aus der Liste als EDnearby genommen. Alle so bestimmten CED-Werte werden einer Liste angehängt. In der Funktion der `decoder.py` wird die Funktion zur Berechnung des CED-Wertes aus den Daten aufgerufen und diese ermittelten Werte, an die Stellen, wo der `decoder` aufgerufen wird, zurückgegeben.

7. **Suchen des besten Ergebnisses**
   Die Methoden `searchForMatches`, `getMeanCEDs` und `getBestMatches` verarbeiten die berechneten CEDs. Für jeden Header werden alle Segmente einmal angehängt, dekodiert und der CED-Wert berechnet. Ist dies aufgrund der inkorrekten Daten nicht möglich, wird das nächste Segment in der Liste genutzt. Wenn keine CED-Werte existieren, weil das Bild beispielsweise nicht dekomprimiert werden konnte, wird das Header-Segment ohne angehängtes Segment gespeichert. Erhält man jedoch CED-Werte für mehrere Segmente, werden zuvor die Segmente in `filterCEDs` aussortiert, bei denen mindestens ein CED-Wert größer oder gleich eins ist. Diese Filterung wird gemacht, da ein sehr hoher CED-Wert signalisiert, dass beide Segmente nicht zur selben Datei gehören. Ist der CED-Wert hoch, dann ist der Unterschied von aufeinanderfolgenden Bildzeilen groß, was bei zwei unterschiedlichen Bildern der Fall sein kann.
   Aus diesem Grund erfolgt die vorherige Filterung der CED-Werte.
   Aus den CED-Werten für jede Kombination aus Header und Segment wird der Durchschnitt bestimmt, welcher für die Ermittlung des besten Ergebnisses herangezogen wird. Die durchschnittlichen CED-Werte aller Kombinationen werden als nächstes miteinander verglichen und das Nicht-Header-Segment mit dem niedrigsten Wert wird mit dem Header-Segment verbunden. Der Index des angehängten Segments wird aus der Liste der freien Nicht-Header-Segmente entfernt, da nach dem Prinzip gearbeitet wird, dass jedes Fragment nur einmal verwendet werden kann.
   Am Schluss erfolgt eine erneute Überprüfung, ob das neue Segment einen Footer enthält. Ist dies nicht der Fall, wird ein neues Segment angehängt und alle weiteren Schritte erneut bis zum Erreichen eines Footers oder dem Fehlen weiterer passender Segmente durchgeführt. Ist jedoch ein Footer vorhanden, beginnt der letzte Schritt der Reassembly-Phase.

8. **Speichern des Bildes**
   Der letzte Schritt der Reassembly-Phase und des kompletten Prototyps besteht aus der Speicherung des Bildes. Für diesen Prozess sind die Methoden `createFolder`, `sliceAfterFooter` und `saveImage` verantwortlich. Damit die Bilder nicht im selben Ordner wie das eigentliche Programm liegen und der Nutzer eine bessere Übersicht hat, wird als erstes ein Ordner `pictures` erzeugt, sofern dieser noch nicht existiert. Damit ist der erste Schritt bereits abgeschlossen und

es folgt eine Bearbeitung aller Daten mit Footer. Da das Bild nach dem Footer-Marker endet und alle nachfolgenden Daten innerhalb des letzten Clusters nicht zum Bild gehören, erfolgt ein Wegschneiden der nicht zugehörigen Daten.
Im Anschluss müssen die Bilddaten in den angelegten Ordner geschrieben werden. Das übernimmt die Funktion `saveImage`. Zuerst wird der Dateiname aus dem aktuellen Datum und der Uhrzeit bestimmt. Die Uhrzeit wird auf die Mikrosekunde genau gewählt, damit auch tatsächlich alle Dateien gespeichert werden und es zu keiner Überschreibung aufgrund von Zeitdopplung kommt. Als Endung wird *.jpeg* festgelegt, da diese von fast allen Decodern unterstützt wird. Die Bilddaten werden in das Dokument des generierten Namens geschrieben und durch eine Ausgabe dem Nutzer signalisiert, dass eine Datei gesichert wurde.

3.2 Anwendung des Prototyps

Die Anwendung des Programms ist simpel und funktioniert über die Kommandozeile. Beim Aufruf muss zwingend der Pfad durch `-f` gefolgt vom Pfad zu der zu analysierenden Datei mitgeliefert werden. Ein weiterer Parameter, der dem Programm übergeben werden kann, aber keine Pflichteingabe darstellt, ist die Clustergröße der vorliegenden Datei/ des Datenträgerabbildes. Wenn keine Größe über `-c` in Bytes beigefügt wird, dann wird die Standardclustergröße von 4096 Bytes für den weiteren Verlauf genutzt. Ein Start des Programms kann wie folgt aussehen.

```
$ python main.py -f dateipfad\datei -c 4096
```

Neben diesen Parametern gibt es noch weitere, welche die Ausgabe der Ergebnisse beeinflussen. Eine Auflistung dieser Parameter ist aus der Tabelle 3.1 entnehmbar. Abgesehen des Pfades zur Datei sind alle anderen Parameter optional und dem Anwender überlassen.

Tabelle 3.1: Auflistung der Kommandozeilenparameter, die dem Programm übergeben werden können

Parameter	**Beschreibung**	**notwendig**
`-f`	Pfad zur Datei, die analysiert werden soll	x
`-c`	Setzen einer Clustergröße für das Speicherabbildes	
`-i`	Anzeigen der Indexe aller Cluster des Speicherabbildes	
`-e`	Anzeigen der Entropie für jedes Cluster	
`-s`	Anzeigen der JPEG-Cluster, aufgeteilt in Segmente	
`-ceds`	Anzeigen der CED-Werte für die dekodierbaren Segment-Kombinationen	
`-mceds`	Anzeigen der durchschnittlichen CED-Werte der vorgefilterten, möglichen Segmentkombinationen	

Vorab ist es wichtig, die Grenzen des Programms und die Voraussetzungen für die zu analysierenden Daten zu erwähnen. Bei den Grenzen handelt es sich um bestimmte Szenarien, die aus diversen Gründen, welche folgend erklärt werden, nicht berücksichtigt wurden.
Das Speicherabbild, von dem JPEG-Dateien wiederhergestellt werden sollen, sollte recht klein sein, da das Programm für größere Datenmengen nicht ausgelegt ist. Die Größe der Datenmenge kann an dieser Stelle nicht auf einen bestimmten Wert begrenzt werden, da die Zeit, die der Prototyp benötigt, davon abhängig ist wie viele Segmente zur Verknüpfung vorhanden sind. Außerdem kann es nur mit JFIF-Dateien umgehen. Die Begründung liegt darin, dass in EXIF-Bildern Thumbnails integriert sind, welche ebenfalls einen SOI und EOI beinhalten. Das Programm geht allerdings lediglich von jeweils einem EOI und SOI pro Bild aus. Diese Eigenschaft kann bei EXIF-Bildern zu falschen Ergebnissen führen. Zusätzlich darf es sich bei den JFIF-Dateien nur um baseline-kodierte Bilder handeln, da bei diesen das Vorkommen nur eines SOS Markers sichergestellt ist. Des Weiteren muss bei der Benutzung berücksichtigt werden, dass kein verschlüsseltes Speicherabbild genutzt wird und dem Anwender bereits die Clustergröße bekannt ist.

3.3 Testszenarien

Zur Überprüfung der Lauffähigkeit des entwickelten Programms müssen diverse Szenarien überprüft werden, welche in der Realität auftreten können. Diese Szenarien können sich sowohl in der Anzahl der fragmentierten Dateien, als auch in der Art der Dateitypen unterscheiden. Auch wenn möglichst viele Szenarien entwickelt wurden, kann nur ein Bruchteil der auftretenden Fälle abgedeckt werden. Die Szenarien dienen auch zum Testen, ob bestimmte Eigenschaften des Programms so arbeiten, wie durch die Implementierung vorgesehen. Da es sich lediglich um einen ersten Prototyp handelt, können keine real entstandenen Fragmentationsszenarien getestet werden. Dies liegt an der enormen Größe von ganzen System-Speicherabbildern, für die der Prototyp nicht ausgelegt ist. Die Szenarien dienen auch dazu, Grenzen aufzuzeigen und Einflussfaktoren für gute Ergebnisse zu ermitteln.
In Tabelle 3.2 ist eine grobe Übersicht der einzelnen Szenarien dargestellt. Eine genauere Beschreibung der Szenarien folgt in den Unterkapiteln.
Die Szenarien wurden händisch erzeugt, damit die Art der Fragmentierung beeinflussbar war. Hierfür wurden zu Beginn mit Hilfe eines Skripts die Inhalte des Speicherabbildes in seine Cluster aufgeteilt. Anschließend wurden diese Cluster neu sortiert und zusammengefügt. Für die Szenarien wurden sowohl eigene, als auch lizenzfreie Bilder verwendet. Die Auswahl der Bilder wurde aufgrund ihrer farblichen und strukturellen Unterschiedlichkeit getroffen. Bei den JPEG-Bildern, welche vom Programm rekonstruiert werden sollen, handelt es sich lediglich um Bilder des JFIF-Typs. Andere Bilder wie EXIF-Bilder, die ebenfalls den JPEG-Standard verwenden, wurden ignoriert. Des Weiteren wurden Textdateien genutzt, die zufällig erzeugt wurden.

Tabelle 3.2: Auflistung der entwickelten Szenarien mit kurzer Beschreibung

Szenario	Dateien	Beschreibung
1	Bilder 1-12	Zwölf unfragmentierte Bilder
2	Bild1, Textdatei	Textdatei zwischen zwei Bildfragmente
3	Bild1, Textdatei	Textdatei zwischen zwei ungeordneten Bildfragmenten
4	Bild1, Bild2	Bilddatei zwischen zwei Bildfragmenten eines anderen Bildes
5	Bild1, PNG-Datei	PNG-Datei zwischen zwei Bildfragmenten
6	Bild1, Textdatei	Textdatei zwischen drei Bildfragmenten
7	Bild1, Bild2	Zwei Bilder aufgeteilt in zwei Fragmente
8	Bild1, Bild2, Bild5, Bild11, Bild12, PNG-Datei, Textdatei, Worddokument, ausführbare Datei	Zwei Bilder unfragmentiert, zwei Bilder und die PNG-Datei in zwei Fragmenten, ein Bild in drei Fragmenten, unterbrochen von einer Text-, einer Word- und einer ausführbaren Datei

3.3.1 Szenario 1

Das erste Szenario dient im Besonderen der Überprüfung, ob der Prototyp innerhalb der Collating-Phase alle relevanten Cluster findet oder ob bestimmte JPEG-Cluster durch die Filter-Funktionen nicht erkannt und fälschlicherweise rausgefiltert werden. Falls dies der Fall sein sollte, sollen die Auswirkung von fehlenden relevanten Clustern auf das rekonstruierte Bild herausgestellt werden.
Zu diesem Zweck werden mehrere JPEG-Dateien dem Prototyp übergeben, ohne dass diese fragmentiert sind oder andere Daten enthalten. Es handelt sich lediglich um die eigentliche Bilddatei, von welcher im Idealfall alle Cluster als JPEG-Cluster identifiziert werden.
Bei den Bildern handelt es sich um zwölf optisch unterschiedliche Bilder, von denen alle Bilder mit Ausnahme der Bilder 9-11, welche lizenzfrei von der Webseite *Pexels* heruntergeladen wurden, selbst erzeugt wurden [71]. Die zwölf verwendeten Bilder mit ihren entsprechenden Eigenschaften sind in Abbildung 3.8 in Kleinansicht dargestellt. Der MD5-Hashwert, eine Prüfsumme zum Vergleich ob Daten identisch sind, dient für die Auswertung der Ergebnisse dazu, sicherzustellen, dass ein Bild nicht nur optisch, sondern auch die Daten vollständig korrekt rekonstruiert wurden.

Dateiname	Originalbild	Dateigröße	Clustergröße	Pixelgröße	Hashwert
Bild1.JPG		74.627 Bytes	19	640 x 480	513ec1e0a764ba6a3b104278b135c787
Bild2.JPG		54.964 Bytes	14	1501x1664	5e682a7233d24b80e7d2318c5829ea81
Bild3.JPG		128.613 Bytes	32	1713 x 1941	7044e624a130f431a1c2614d8e5ddd7b
Bild4.JPG		282.573 Bytes	69	3108 x 2586	1b0f80daad94c2deae04d206689f9cf6
Bild5.JPG		334.889 Bytes	82	2973 x 2676	fe0d1e66e95b935dda29ff6928d439db
Bild6.JPG		199.635 Bytes	49	3084 x 1902	ef06ce7aac12deb379c5b4a49c9913ac
Bild7.JPG		246.088 Bytes	61	2787 x 2151	6cff289e00eae9b94fcaad76be143433
Bild8.JPG		248.248 Bytes	61	2448 x 1764	06eaf46e037c5c0c229be4d5e2d31791
Bild9.jpg		169.385 Bytes	42	2529 x 2070	847490a50d4ebc0088fc653cb07da920
Bild10.jpg		94.080 Bytes	23	1630 x1250	0028269c05bfc29bfcc5ad90dc9e429d
Bild11.jpg		88.627 Bytes	22	1194 x 968	5df3d3a385ea9ded54c53eb21d4a2417
Bild12.jpeg		48.596 Bytes	12	721 x 437	250dca89e20eb40cc92896d0936f4b0a

Abbildung 3.8: Eigenschaften von allen Bildern, die in den Szenarien verwendet werden

3.3.2 Szenario 2

Das zweite Szenario dient dazu, den vollständigen Prototyp mit allen Phasen auf seine Korrektheit bei einem recht simplen Szenario zu überprüfen. Hierbei wird das Bild1.JPG verwendet und in zwei Fragmente aufgeteilt, welche in der korrekten Reihenfolge aufeinander folgen. Zwischen diesen Fragmenten befindet sich eine zufällig erzeugte, genau zwei Cluster große Textdatei. Über wie viele Cluster sich die Textdatei streckt ist irrelevant, jedoch sollte sie die Cluster vollständig ausfüllen. Die genaue Aufteilung der Cluster ist in Abbildung 3.9 dargestellt.

Abbildung 3.9: Zusammensetzung der Cluster des Speicherabbildes (Szenario 2)

3.3.3 Szenario 3

Um zu überprüfen, ob die Anordnung von Header- und Nicht-Header-Fragment einen Einfluss auf das Ergebnis hat, existiert Szenario 3. Es ist wie Szenario 2 aufgebaut, unterscheidet sich jedoch durch die Anordnung der beiden Fragmente. Es wird dasselbe Bild wie vorher in dieselben Fragmente wie zuvor geteilt, jedoch beginnt das Speicherabbild mit dem Nicht-Header-Fragment mit zwei Clustern Textdatei dazwischen und endet mit dem Header-Fragment. Da die Bilddatei das letzte Cluster nicht ausfüllt und ein Rutschen der Daten des Folgeclusters in das Encluster, wird dieses mit zufälligen Werten aufgefüllt. Die Anordnung lässt sich aus Abbildung 3.10 entnehmen.

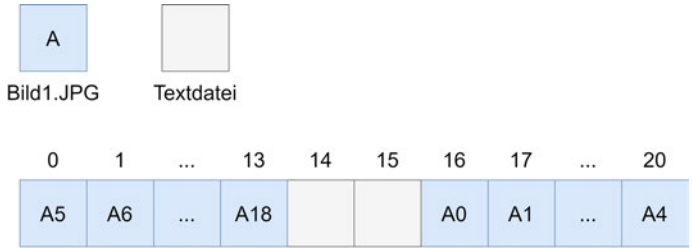


Abbildung 3.10: Zusammensetzung der Cluster des Speicherabbildes (Szenario 3)

3.3.4 Szenario 4

Das vierte Szenario dient der Überprüfung, ob der Prototyp zuverlässig eine Bilddatei zwischen einer anderen fragmentierten Bilddatei herausfiltern kann, ohne beide Informationen miteinander zu verknüpfen. Es wird geschaut, ob die Fragmente der Bilddatei zusammengesetzt werden können und der Prototyp erkennt, dass zwischen diesen Fragmenten ein vollständiges Bild liegt. Aus diesem Grund wurde Bild2 vollständig zwischen zwei Fragmente von Bild1 geschoben. Zusätzlich wurde das letzte Cluster von Bild2 aufgefüllt, damit das zweite Fragment von Bild1 in einem neuen Cluster starten kann. Die Zusammensetzung der Bilder lässt sich aus Abbildung 3.11 entnehmen.

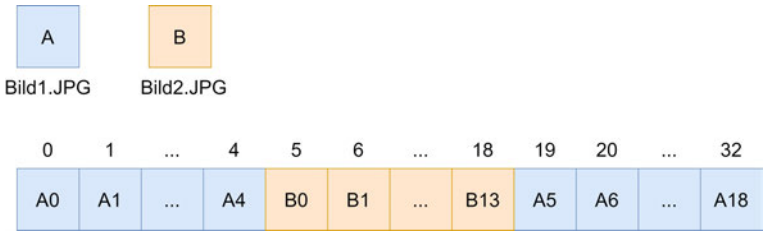


Abbildung 3.11: Zusammensetzung der Cluster der Speicherabbilder (Szenario 4)

3.3.5 Szenario 5

Das Szenario 5 soll überprüfen, ob Bilder eines anderen Dateityps in der Collating-Phase herausgefiltert werden können. Dies ist insofern interessant zu überprüfen, da Bilder, unabhängig des Kompressionsalgorithmus, im Allgemeinen ähnliche Eigenschaften besitzen. Des Weiteren ist ein Vorkommen von Bildern unterschiedlicher Dateitypen in realen Szenarien sehr wahrscheinlich, weshalb der Prototyp mit solchen Problemen umgehen sollte.Außerdem sollen die Auswirkungen auf die Reassembly-Phase geprüft werden, falls zuvor keine korrekte Klassifizierung erfolgt ist. Für das Szenario wurde das Bild 3.12 von dem Dateityp PNG verwendet und das letzte Cluster dieses aufgefüllt mit zufälligen Werten.

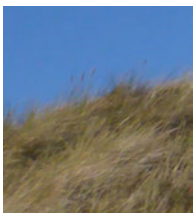


Abbildung 3.12: Bild des Dateityps PNG

Die Bildeigenschaften sind für den Verlauf irrelevant, da die Cluster im Idealfall ignoriert werden sollen. Als JPEG-Bild wurde erneut Bild1 verwendet und in zwei Fragmente, mit dem PNG-Bild dazwischen, aufgeteilt, wie in Abbildung 3.13 anschaulich dargestellt.

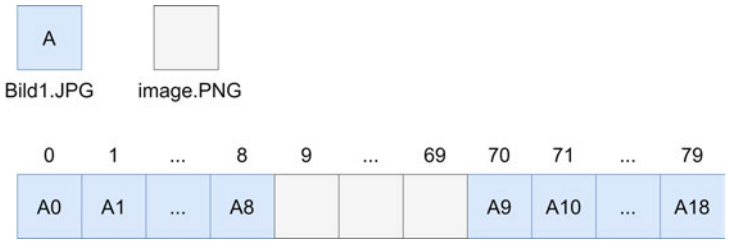


Abbildung 3.13: Zusammensetzung der Cluster des Speicherabbildes (Szenario 5)

3.3.6 Szenario 6

Das Szenario 6 soll überprüfen, wie der Prototyp mit dreifach fragmentierten Bilddaten umgeht und ob der Fragmentationspunkt eine Auswirkung auf das Ergebnis hat. Deshalb ist Szenario 6 in 6a-6e unterteilt, wie in Abbildung 3.14 dargestellt.

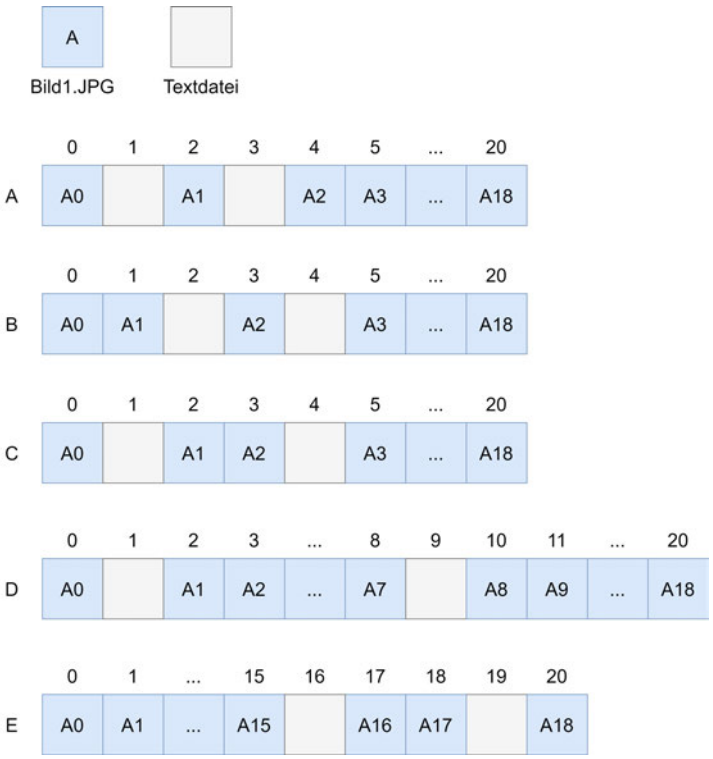


Abbildung 3.14: Zusammensetzung der Cluster der Speicherabbilder (Szenario 6)

Bei jedem dieser Unterszenarien wird die Bilddatei Bild1 genutzt, in drei Fragmente aufgeteilt und zwischen die Fragmente eine ein Cluster große Textdatei eingefügt. Der Unterschied zwischen den Ausführungen besteht in dem Fragmentierungspunkt der Bilddatei.

3.3.7 Szenario 7

Bisher wurde nur mit einer einzelnen fragmentierten Bilddatei gearbeitet, was in Szenario7 geändert wird. Dieses Szenario testet die Fähigkeit des Programms, mit mehreren Fragmenten verschiedener Bilddateien umzugehen. Zu diesem Zweck wurden Bild1 und Bild2 in jeweils zwei Fragmente geteilt und abwechselnd aneinandergehängt. Um zusätzlich zu überprüfen, ob die Position der beiden Header einen Einfluss auf das Ergebnis hat, wurden zwei Unterszenarien entwickelt. Bei jedem Unterszenario wurde die Position der beiden Header wie aus Abbildung 3.15 entnehmbar, getauscht.

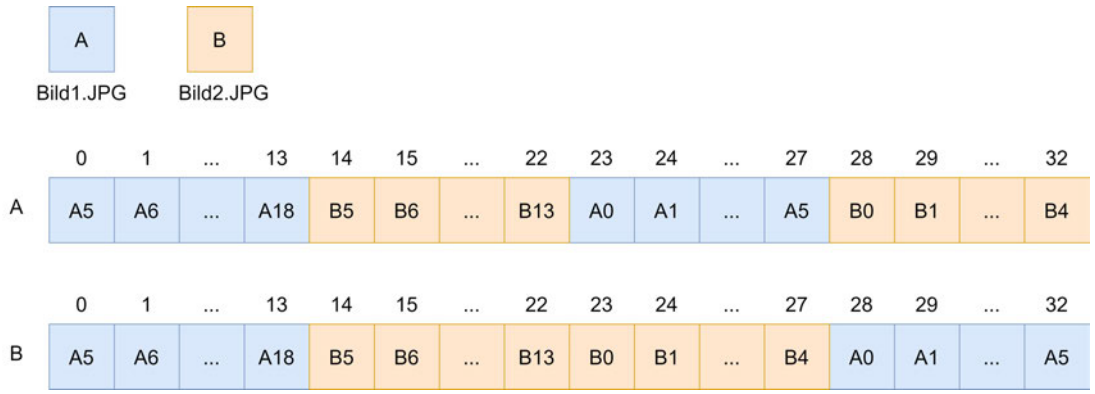


Abbildung 3.15: Zusammensetzung der Cluster des Speicherabbildes (Szenario 7)

3.3.8 Szenario 8

Die bisherigen Szenarien bestehen aus maximal zwei Bildern und sind darauf spezialisiert kleine Teilbereiche zu testen. Bei Szenario 8 sollen die verschiedenen Szenarien miteinander vermischt werden, damit ein realitätsnäheres Speicherabbild entsteht. Zu diesem Zweck wurden die Bilder 1, 2, 5, 8, 11 und 12, sowie eine Textdatei, ein Word-Dokument, eine ausführbare Datei und das PNG-Bild aus Szenario 5 verwendet. Die Bilder 1, 8, und 12 wurden unfragmentiert in diesem Szenario abgelegt, während Bild5 und Bild11 in zwei Fragmente und Bild2 in drei Fragmente aufgeteilt wurden. Auch das PNG-Bild wurde in diesem Szenario in zwei Fragmente aufgeteilt, während die anderen übrigen Dateien an einem Stück in das Speicherabbild eingefügt wurden. Von der Textdatei und dem ausführbaren Programm wurde jeweils nur ein Cluster genommen, während das Worddokument vollständig, bestehend aus drei Clustern, verwendet wurde. Da dieses das letzte Cluster nicht vollständig aufgefüllt hat, wurden die fehlenden Stellen aufgefüllt. Die Aufteilung dieser ist aus Abbildung 3.16 entnehmbar.

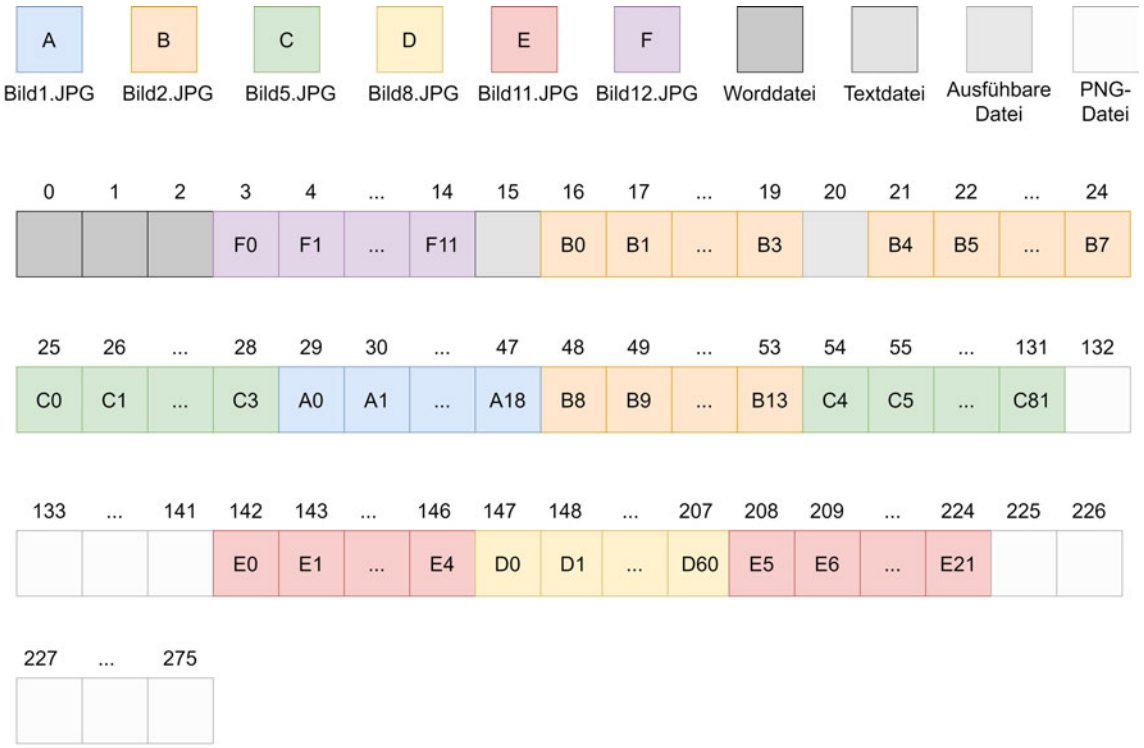


Abbildung 3.16: Zusammensetzung der Cluster des Speicherabbildes (Szenario 8)

3.4 Evaluationsstrategie

Für die Beurteilung der Ergebnisse, die das Programm nach Durchlauf der Testszenarien liefert, musste ein Prinzip entwickelt werden, welches eine objektive Bewertung ermöglicht. Dabei wurden nicht nur die Endergebnisse, sondern auch die Zwischenschritte bewertet. Zu Beginn wurde geschaut, welche Bildcluster richtig als solche klassifiziert, welche vergessen und welche falsch eingeordnet wurden. Hierbei wurde die Collating-Phase als eigenständiger Prozess beurteilt. Zu Beginn wurde jedes Szenario alleinstehend betrachtet und eine Bewertung anhand der erhaltenen Ergebnisse ermittelt. Anschließend wurde aus den einzelnen ein gesamtes Resultat für die Effektivität der Collating-Phase abgeleitet. Die Bewertung der Klassifizierungs-Ergebnisse erfolgte auf Grundlage binärer Klassifikatoren, welche insbesondere aus dem Maschinellen Lernen bekannt sind. Neben den allgemeinen Klassifikatoren, welche in Graphik 3.17 speziell für den aktuellen Zweck eingeordnet sind, wurde im Besonderen auf die Sensitivität, auch Richtig-Positiv-Rate genannt, eingegangen.

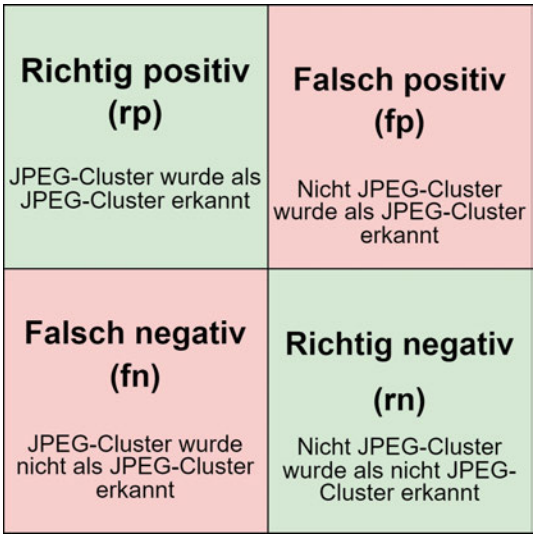


Abbildung 3.17: Konfusionsmatrix angepasst für die Collating-Phase des Prototyps

Auf den aktuellen Fall bezogen misst die Sensitivität die Fähigkeit des Programms, Cluster zu erkennen, bei denen es sich tatsächlich um JPEG-Cluster handelt. Mit der angegebenen Formel lässt sich diese bestimmen [72].

$$Sensitivität = \frac{rp}{rp + fn} \tag{3.1}$$

Des Weiteren wird die Spezifität, die Richtig-Negativ-Rate, betrachtet, um den Anteil der korrekt als nicht-JPEG-Cluster bestimmten Cluster ausfindig zu machen [72].

$$Spezifität = \frac{rn}{rn + fp} \tag{3.2}$$

Anhand dieser Formeln kann die Funktionalität der Collating-Phase beurteilt werden. Von der Korrektheit dieser Phase ist das Ergebnis in seiner Gesamtheit abhängig. Werden JPEG-Cluster nicht als diese erkannt, fehlen sie im Reassembly-Schritt. Weniger drastisch wirkt sich die Falsch-Positiv-Rate auf den weiteren Verlauf aus, da nicht passende Cluster in der Reassembly-Phase bei unzutreffenden Voraussetzungen weiterhin entfernt werden können. Dies sollte möglichst vermieden werden.

Die Evaluation der Gesamtergebnisse, also die der extrahierten JPEG-Dateien, welche sowohl fragmentiert, als auch unfragmentiert vorliegen, wird durch eine Wiederherstellungsrate, welche die prozentuale Anzahl der wiederhergestellten Daten angibt, berechnet. Der Schwierigkeitsgrad der einzelnen Szeanrien und die Art der Fragmentierung finden keine Berücksichtigung bei der Bewertung der Ergebnisse. Der Grund für diese Entscheidung liegt darin, dass bei einem realen System nicht auf die Komplexität der Fragmentierung geschaut wird, sondern am Ende lediglich das Ergebnis zählt.

4 Ergebnisse

In dem folgenden Kapitel werden die Ergebnisse dargelegt, welche der Prototyp aus den Testszenarien generiert. Für die einheitliche Bewertung der Ergebnisse wurde auf die zuvor entwickelte Evaluationsstrategie zurückgegriffen. Bei der Generierung der Ergebnisse wurde zwischen Collating- und Reassembly-Phase unterschieden. Eine Auflistung der Ergebnisse ist in 4.1 abgebildet und eine detailliertere Betrachtung der Ergebnisse pro Szenario erfolgt im Anschluss.

Tabelle 4.1: Ergebnisse des Prototyps für alle Szenarien

	Klassifizierte Cluster				
Szenario	rp	rn	fp	fn	**Korrekt rekonstruiert**
1	337	\	\	149	4/12
2	19	2	0	0	1/1
3	19	2	0	0	1/1
4	33	\	\	0	2/2
5	19	2	59	0	1/1
6	19	2	0	0	2/5
7	33	\	\	0	4/4
8	184	5	61	26	2/6

4.1 Szenario 1

Das erste Szenario dient einer Überprüfung, ob der Prototyp alle JPEG-Cluster richtig als solche erkennt. Da lediglich relevante Cluster in dem Szenario verwendet wurden, kann nur die Sensitivität und nicht die Spezifität berechnet werden.

Zu Beginn erfolgt eine Beurteilung des Klassifizierungsalgorithmus des Prototyps. Hierzu wird der Fokus auf die Anzahl der erkannten Cluster aus den eigentlich vorhandenen relevanten Clustern gelegt. Betrachtet man die Anzahl der korrekt klassifizierten Cluster aus Tabelle 4.2, konnten bei vier Bildern alle Cluster als JPEG-Cluster identifiziert werden.
Bei den anderen Bildern wurden nicht alle Cluster erkannt und die Anzahl der gefundenen Cluster schwankt. Beispielsweise wurden bei Bild4 und Bild6 weniger als die Hälfte der Bildcluster auch als solche durch das Programm erkannt. Neben diesen Informationen findet sich in der Tabelle zusätzlich noch eine Auflistung der Clusterindexe der Cluster, welche als JPEG klassifiziert wurden. Hierbei symbolisieren die eckigen Klammern Segmente in welche der Prototyp die Cluster aufteilt.

Tabelle 4.2: Ergebnisse der Collating-Phase des Prototyps (Szenario 1)

Dateiname	JPEG-klassifizierte Cluster, aufgeteilt in Segmente	Korrekt klassifziert
Bild1.JPG	[0-18]	19/19
Bild2.JPG	[0-13]	14/14
Bild3.JPG	[0] [2] [6-31]	28/32
Bild4.JPG	[0] [20] [31] [34-35] [40-43] [45] [47] [49] [52] [54] [68]	15/69
Bild5.JPG	[0] [5] [8-24] [27-49] [51-54] [56-61] [63] [75-79] [81]	59/82
Bild6.JPG	[0] [16] [29] [31] [41-43] [45-46] [48]	10/49
Bild7.JPG	[0] [10-13] [15-60]	51/61
Bild8.JPG	[0] [3-53] [55-60]	58/61
Bild9.jpg	[0-4] [6-7] [10-26] [31-36] [39-41]	33/42
Bild10.jpg	[0] [2-3] [5-13] [16] [18-19] [22]	16/23
Bild11.jpg	[0-21]	22/22
Bild12.jpeg	[0-11]	12/12
GESAMT		337/486

Für eine allgemeine Beurteilung der Collating-Phase dieses Szenarios folgt eine Berechnung der Sensitivität. Aus dem Einsetzen der Ergebnisse für alle Bilder resultiert eine Sensitivität von 0,6934, was rund 69% entspricht.

$$rp = 337$$
$$fn = 149$$
$$Sensitivität = \frac{337}{337+149} = 0,6934 = 69,34\%$$

Aus dem Ergebnis lässt sich ableiten, dass über die Hälfte der relevanten Cluster, Cluster des JPEG-Typs, auch als diese erkannt wurden.

Nach den Ergebnissen der Collating-Phase folgt eine Beurteilung der Ergebnisse der Reassembly-Phase. Da die Ergebnisse von der Klassifzierung der Collating-Phase abhängig sind, ist eine eigenständige Beurteilung der Ergebnisse nicht möglich.
Aus Graphik 4.1 lässt sich ableiten, dass die Bilder, bei denen alle Cluster korrekt klassifiziert wurden, auch vollständig rekonstruiert werden konnten. Ein Abgleich des Hashwertes dieser Bilder belegt die optische Feststellung, dass die Ausgangsbilder identisch zu den Eingangsbildern sind. Die Bilder, um die es sich hierbei handelt, sind Bild1, Bild2, Bild11 und Bild12.

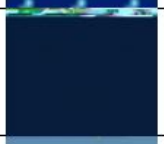
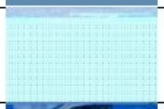
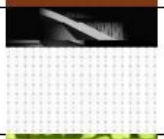
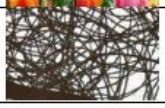
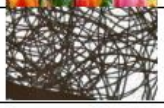
Dateiname	Originalbild	Rekonstruiertes Bild	Hashwert Originalbild	Hashwert Ergebnis
Bild1.JPG			513ec1e0a764ba6a3b104278b135c787	513ec1e0a764ba6a3b104278b135c787
Bild2.JPG			5e682a7233d24b80e7d2318c5829ea81	5e682a7233d24b80e7d2318c5829ea81
Bild3.JPG			7044e624a130f431a1c2614d8e5ddd7b	216c1c2c0216290ba96a5f9c2b7756b6
Bild4.JPG			1b0f80daad94c2deae04d206689f9cf6	46d7d3f5e61ba0318a94f6ae368eea08
Bild5.JPG			fe0d1e66e95b935dda29ff6928d439db	bc13b6948c22885f192d3fa5f481a085
Bild6.JPG			ef06ce7aac12deb379c5b4a49c9913ac	da1a654211b5dca431427d0c3fe5c281
Bild7.JPG			6cff289e00eae9b94fcaad76be143433	21ac5dae8d1f6d6e09c560ae9cb33787
Bild8.JPG			06eaf46e037c5c0c229be4d5e2d31791	c86f3ac6b3e780d3d4b8f0c46d3bbe9c
Bild9.jpg			847490a50d4ebc0088fc653cb07da920	26bc6552b829e54e9b38eb3d3c66c544
Bild10.jpg			0028269c05bfc29bfcc5ad90dc9e429d	f45b7124f05eeb3cd691693390882ae3
Bild11.jpg			5df3d3a385ea9ded54c53eb21d4a2417	5df3d3a385ea9ded54c53eb21d4a2417
Bild12.jpeg			250dca89e20eb40cc92896d0936f4b0a	250dca89e20eb40cc92896d0936f4b0a

Abbildung 4.1: Gegenüberstellung der Original- und rekonstruierten Bilder mit jeweiligem Hashwert (Szenario 1)

Die optische Begutachtung der übrigen Bilder ergibt, dass mindestens am oberen Rand der Bilder einige Pixel des Originalbildes zu erkennen sind.
Die Auseinandersetzung mit den Segmenten, ergibt, dass alle Bilder beim Speichern mit dem ersten, also dem Headersegment, kein Problem hatten.
In der Tabelle 4.2 sind zu jedem analysierten Bild von Szenario 1 die einzelnen Segmente angegeben, welche nach der Collating-Phase gebildet werden und mit denen im Verlauf gearbeitet wird. Hierzu sind die Cluster, welche ein Segment bilden, und abschließend der Index der Segmente, welche durch die Reassembly-Phase zusammengefügt wurden, angegeben.

Tabelle 4.3: Auflistung der Cluster aus denen, die der Prototyp zum Zusammensetzen nutzt (Szenario 1)

Dateiname	**Cluster des Endergebnisses, aufgeteilt in Segmente**
Bild1.JPG	[0-18]
Bild2.JPG	[0-13]
Bild3.JPG	[0] [2] [6-31]
Bild4.JPG	[0] [34-35] [54] [68]
Bild5.JPG	[0]
Bild6.JPG	[0] [41-43] [31] [16] [45-46] [29] [48]
Bild7.JPG	[0] [10-13]
Bild8.JPG	[0]
Bild9.JPG	[0-4] [31-36] [39-41]
Bild10.JPG	[0]
Bild11.jpg	[0-21]
Bild12.jpeg	[0-11]

Es folgt eine Übersicht der Ergebnisse der Bilder, welche nicht korrekt rekonstruiert werden konnten. Beginnt man mit der Betrachtung der Anordnung der klassifizierten Cluster bei Bild3, lässt sich aus der Tabelle entnehmen, dass trotz fehlender Cluster die drei Segmente mit den vorhandenen Clustern richtig angeordnet und alle für das Endergebnis verwendet wurden.
Bei der Auseinandersetzung mit dem folgenden Bild, Bild4, ist dies ebenfalls der Fall. Hier besteht das Endergebnis aus vier Segmenten, welche in aufsteigender Reihenfolge aufeinander folgen. Des Weiteren wurden von den verfügbaren Clustern nochmal zehn Cluster in der Reassembly-Phase aussortiert, wodurch das Ergebnis-Bild aus fünf von ursprünglich 69 Clustern besteht.
Bei Bild5 wurde lediglich ein Segment, bestehend aus dem Header-Cluster, für die Rekonstruktion genutzt. Insgesamt wurden durch die Reassembly-Phase 57 Cluster aussortiert, welche sich aus neun Segmenten zusammensetzen.
Im Gegensatz dazu wurden bei Bild6 alle der verfügbaren Cluster für das Endergebnis verwendet, jedoch ist die Reihenfolge inkorrekt, in der diese sieben Segmente angeordnet wurden.

Das Ergebnis für das nächste Bild, Bild7, zeigt, dass die zusammengesetzten Segmente sich zwar in der korrekten Reihenfolge befinden, jedoch wieder eine Vielzahl der als JPEG klassifizierten Cluster verworfen wurden. Von möglichen 51 Clustern konnten nur fünf miteinander verbunden werden. Da es sich um zwei Segmente handelt, wurde nur eins dem Header-Segment als passend zugeordnet.
Anhand der Informationen aus der Tabelle wird deutlich, dass das Ausgangsbild von Bild8 nur noch aus einem Segment, bestehend aus einem Cluster, dem Headercluster, repräsentiert wird. Stellt man dieses Ergebnis den klassifizierten Clustern, welche für die Rekonstruktion hätten verwendet werden können, gegenüber, wurden durch den Reassembly-Algorithmus noch einmal 57 Cluster verworfen.
Als nächstes folgt eine Auseinandersetzung mit dem Ergebnis von Bild9. Dieses besteht aus drei Segmenten, welche sich auf 14 Cluster aufteilen, die in einer richtigen Reihenfolge aneinandergefügt wurden. Zwei Segmente konnten nicht verwendet werden und wurden vom Prototyp aussortiert, wodurch erneut eine Reduktion von 19 Clustern eintritt.
Eine große Reduktion durch die Reassembly-Phase zeigt auch das als letztes zu betrachtende Bild, Bild10. Als Ergebnis der Collating-Filterung bleiben von 33 Clustern noch 16 zur Rekonstruktion, wovon lediglich das Header-Cluster wiederhergestellt wurde. Die anderen 15 Cluster, aufgeteilt in fünf Segmente finden keine Verwendung.
Fasst man diese Ergebnisse zusammen, können nur 33,33% der Bilder wiederhergestellt werden.

$$Wiederherstellungsrate = \frac{4}{12} = 0,3333 = 33,33\%$$

Das Ergebnis zeigt, dass rund ein Drittel der Bilder auch ohne Fragmentierung rekonstruiert werden können.

4.2 Szenario 2

Im Gegensatz zu Szenario 1 berücksichtigt Szenario 2 die Spezifität durch eine Textdatei, bestehend aus zwei Clustern. Der Tabelle 4.4 ist zu entnehmen, dass alle Cluster des Bildes1, wie bereits im Szenario zuvor, korrekt als JPEG-Cluster erkannt wurden.

Tabelle 4.4: Gegenüberstellung der korrekten Ergebnisse der Klassifizierung und denen des Prototyps (Szenario 2)

	JPEG-Cluster	**Keine-JPEG Cluster**
Ergebnis Prototyp	0, 1, 2, 3, 4, 7, 8, 9, 10, 11, 12, 13, 14, 15 16, 17, 18, 19, 20	5, 6
Korrektes Ergebnis	0, 1, 2, 3, 4, 7, 8, 9, 10, 11, 12, 13, 14, 15 16, 17, 18, 19, 20	5, 6

Für die Sensitivität heißt das, dass diese im Fall von Szenario 2 bei 100% liegt.

$$rp = 19$$
$$fn = 0$$
$$Sensitivität = \frac{19}{19+0} = 1 = 100\%$$

Die Ergebnisse aus Tabelle 4.4 der Collating-Phase zeigen auch, dass die Textdatei, welche sich über zwei Cluster erstreckt, entfernt wurde, indem deren Cluster für die Weiterverarbeitung innerhalb der Reassembly-Phase als irrelevant verworfen wurden. Somit ergibt sich eine Spezifität von ebenfalls 100%.

$$rn = 2$$
$$fp = 0$$
$$Spezifität = \frac{2}{2+0} = 1 = 100\%$$

Des Weiteren muss das Ergebnis der Reassembly-Phase betrachtet werden, da das Bild in zwei Fragmenten vorlag, unterbrochen von der Textdatei. Wie in Abbildung 4.2 zu erkennen ist, ist das rekonstruierte Bild optisch betrachtet mit dem Originalbild identisch. Vergleicht man die Hashwerte beider Bilder in der Abbildung, wird die optische Einschätzung bestätigt, dass die bifragmentierte Datei korrekt wiederhergestellt werden konnte.

Dateiname	Rekonstruiertes Bild	Hashwert Originalbild	Hashwert Ergebnis
Bild1.JPG		513ec1e0a764ba6a3b104278b135c787	513ec1e0a764ba6a3b104278b135c787

Abbildung 4.2: Gegenüberstellung des Original- und rekonstruierten Bildes mit jeweiligem Hashwert (Szenario 2)

4.3 Szenario 3

Das dritte Szenario ist fast identisch zum zweiten Szenario. Es unterscheidet sich lediglich durch die der Position des Headers. Da keine Änderung an den verwendeten Clustern vorgenommen wurde, bleibt das Ergebnis der Collating-Phase identisch und kann ebenfalls aus Tabelle 4.4 entnommen werden. Die Betrachtung des rekonstruierten Bildes mit Vergleich der Hashwerte in Abbildung 4.3 zeigt auch allgemein, dass das Ergebnis mit Szenario 2 deckungsgleich ist. Das Bild wird trotz veränderter Headerposition korrekt wiederhergestellt.

Dateiname	Rekonstruiertes Bild	Hashwert Originalbild	Hashwert Ergebnis
Bild1.JPG		513ec1e0a764ba6a3b1042 78b135c787	513ec1e0a764ba6a3b1042 78b135c787

Abbildung 4.3: Gegenüberstellung des Original- und rekonstruierten Bildes mit jeweiligem Hashwert (Szenario 3)

4.4 Szenario 4

Das Speicherabbild des vierten Szenarios besteht aus 33 Clustern, welche lediglich JPEG-Cluster sind. Als Ergebnis der Collating-Phase liefert der Prototyp alle Cluster als JPEG-Cluster zurück, was sich mit dem korrekten Ergebnis deckt, wie aus Tabelle 4.5 entnehmbar ist.

Tabelle 4.5: Gegenüberstellung der korrekten Ergebnisse der Klassifizierung und derjenigen des Prototyps (Szenario 4)

	JPEG-Cluster	Keine JPEG-Cluster
Ergebnis Prototyp	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32	/
Korrektes Ergebnis	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32	/

Durch das Erkennen aller JPEG-Cluster ergibt sich eine Sensitivität von 100%.

$$rp = 33$$
$$fn = 0$$
$$Sensitivität = \frac{33}{33+0} = 1 = 100\%$$

Eine Spezifität kann an dieser Stelle nicht ermittelt werden, da das Abbild nur aus JPEG-Clustern besteht und somit keine irrelevanten Cluster existieren, welche aussortiert werden müssen.

Wie in Abbildung 4.4 zu erkennen, wurden beide Bilder optisch korrekt wiederhergestellt, was ein Abgleich der Hashwerte belegt.

Dateiname	Rekonstruiertes Bild	Hashwert Originalbild	Hashwert Ergebnis
Bild1.JPG		513ec1e0a764ba6a3b104278b135c787	513ec1e0a764ba6a3b104278b135c787
Bild2.JPG		5e682a7233d24b80e7d2318c5829ea81	5e682a7233d24b80e7d2318c5829ea81

Abbildung 4.4: Gegenüberstellung der Original- und rekonstruierten Bilder mit jeweiligem Hashwert (Szenario 4)

4.5 Szenario 5

Die Ergebnisse des Szeanrios 5 der Collating-Phase lassen sich aus Tabelle 4.6 entnehmen. Von insgesamt 80 vorhandenen Clustern wurden zwei Cluster verworfen und nicht als JPEG-Cluster klassifiziert. In diesem Szenario sind allerdings lediglich 19 Cluster wirklich JPEG-Cluster. Beide aussortierten Cluster sind korrekt aussortiert worden und kein relevantes Cluster verworfen worden.

Tabelle 4.6: Gegenüberstellung der korrekten Ergebnisse der Klassifizierung und denen des Prototyps (Szenario 5)

	JPEG-Cluster	Keine JPEG-Cluster
Ergebnis Prototyp	0-8, 10-68, 70-79	9, 69
Korrektes Ergebnis	0-8, 70-79	9-69

Aus diesen Ergebnissen ergibt sich für die Sensitivität in diesem Szenario eine Sensitivität von 100%.

$$rp = 19$$
$$fn = 0$$
$$Sensitivität = \frac{19}{19+8} = 1 = 100\%$$

Im Gegensatz dazu ist die Spezifität, die man nach der Berechnung erhält, mit 3,28% sehr niedrig.

$$rn = 2$$
$$fp = 59$$
$$Spezifität = \frac{2}{2+59} = 0,0328 = 3,28\%$$

Als nächstes erfolgt eine Betrachtung der Endergebnisse nach Durchlaufen der

Reassembly-Phase. Das rekonstruierte Bild in Abbildung 4.5, zeigt sowohl optisch, als auch durch den Hashwert, dass das korrekte Segment der diversen möglichen Segmente angehängt wurde.

Dateiname	Rekonstruiertes Bild	Hashwert Originalbild	Hashwert Ergebnis
Bild1.JPG		513ec1e0a764ba6a3b104278b135c787	513ec1e0a764ba6a3b104278b135c787

Abbildung 4.5: Gegenüberstellung des Original- und rekonstruierten Bildes mit jeweiligem Hashwert (Szenario 1)

4.6 Szenario 6

Zu Beginn werden von Szenario 6 die Ergebnisse der Collating-Phase betrachtet. Die Klassifizierung der insgesamt 21 Cluster ergibt, wie aus Tabelle 4.7, beispielhaft für Szenario 6a dargestellt, entnehmbar ist, ein Ergebnis von 19 relevanten Clustern.

Tabelle 4.7:

	JPEG-Cluster	Keine JPEG-Cluster
Ergebnis Prototyp	0, 2, 4-20	1, 3
Korrektes Ergebnis	0, 2, 4-20	1, 3

Die beiden Cluster mit Textdaten wurden herausgefiltert, während alle JPEG-Cluster als solche erkannt wurden. Sowohl für Sensitivität als auch für die Spezifität heißt das in diesem Szenario ein Wert von 100%.

$$rp = 19$$
$$fn = 0$$
$$Sensitivität = \frac{19}{19+8} = 1 = 100\%$$

$$rn = 2$$
$$fp = 0$$
$$Spezifität = \frac{2}{2+0} = 1 = 100\%$$

Dieses Ergebnis ändert sich durch die Verschiebung des Fragmentationspunktes nicht,

da die Position der einzelnen Cluster keinen Einfluss auf die Filter hat, die zur Klassifizierung genutzt werden.
Allerdings dienen die verschobenen Fragmentationspunkte dazu, den Einfluss auf die Zusammensetzung der einzelnen Fragmente zu betrachten. Bei der optischen Betrachtung der Ausgangsbilder in 4.6 ist zu erkennen, dass zwei der fünf Bilder korrekt wiederhergestellt wurden, während die anderen drei das Bild zwar darstellen, aber farbliche oder räumliche Verzerrungen zu erkennen sind.

Szenario	Rekonstruiertes Bild	Hashwert Originalbild	Hashwert Ergebnis
a)		513ec1e0a764ba6a3b104278b135c787	a5861a43da6058c5b8b06d66dfc29925
b)			99a2af3ef1410486fdd89d3c20ff2152
c)			168e931e66f7d220379298bf9c3afbd9
d)			513ec1e0a764ba6a3b104278b135c787
e)			513ec1e0a764ba6a3b104278b135c787

Abbildung 4.6: Gegenüberstellung der Original- und rekonstruierten Bilder mit jeweiligem Hashwert (Szenario 6)

Die Segmente, aus denen sich die Bilder zusammensetzen, zeigen in Tabelle 4.8, dass bei allen drei inkorrekt rekonstruierten Bildern ein Segment weggelassen wurde.

Tabelle 4.8: Gegenüberstellung der Cluster aus denen sich die Bilder zusammensetzen und denen, die der Prototyp zum Zusammensetzen nutzt (Szenario 6)

Szenario	Korrekte Cluster des Bildes, aufgeteilt in Segmente	Cluster des Endergebnisses, aufgeteilt in Segmente
a	[0] [2] [4-20]	[0] [4-20]
b	[0-1] [3] [5-20]	[0-1] [5-20]
c	[0] [2-3] [5-20]	[0] [5-20]
d	[0], [2- 8], [10- 20]	[0], [2- 8], [10- 20]
e	[0-15], [17-18], [20]	[0-15], [17-18], [20]

Für die Wiederherstellungsrate ergibt sich aus diesem Ergebnis eine Wahrscheinlichkeit von 40%, dass ein JPEG-Bild in diesem Szenario wiederhergestellt werden kann.

$$Wiederherstellungsrate = \frac{2}{5} = 0,4 = 40\%$$

4.7 Szenario 7

Die Ergebnisse von Szenario 7 sind in Tabelle 4.9 dargestellt und ergeben, dass alle Cluster als JPEG-Cluster erkannt wurden. Da dieses Szenario lediglich aus JPEG-Dateien besteht, lässt sich daraus schließen, dass alle relevanten Cluster korrekt erkannt wurden. Dieses Ergebnis war zu erwarten, da bereits in Szenario 1 bei genau diesen Bilddateien alle Cluster erkannt wurden.

Tabelle 4.9: Gegenüberstellung der korrekten Ergebnisse der Klassifizierung und derjenigen des Prototyps (Szenario 7)

	JPEG-Cluster	**Keine JPEG-Cluster**
Ergebnis Prototyp	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32	/
Korrektes Ergebnis	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32	/

Durch das Vorkommen von lediglich relevanten Clustern, kann nur die Sensitivität bestimmt werden, welche 100% ist.

$$rp = 33$$
$$fn = 0$$
$$Sensitivität = \frac{33}{33+0} = 1 = 100\%$$

Sieht man sich die rekonstruierten Bilder in Abbildung 4.7 an, wurden beide Bilder in diesem Szenario sowohl mit Header-Segment von Bild1, als auch mit dem Header von Bild2 an erster Stelle, korrekt rekonstruiert. Die optische Einschätzung der Richtigkeit der Ergebnisse wird durch einen Hashwertabgleich zusätzlich belegt.

Dateiname	Rekonstruiertes Bild	Hashwert Originalbild	Hashwert Ergebnis
Bild1.JPG		513ec1e0a764ba6a3b104278b135c787	513ec1e0a764ba6a3b104278b135c787
Bild2.JPG		5e682a7233d24b80e7d2318c5829ea81	5e682a7233d24b80e7d2318c5829ea81

Abbildung 4.7: Gegenüberstellung der Original- und rekonstruierten Bilder mit jeweiligem Hashwert (Szenario 7)

Für die Berechnung der Wiederherstellungsrate ergibt dieses Szenario einen Wert von 1 und somit 100%.

$$Wiederherstellungsrate = \frac{4}{4} = 1 = 100\%$$

4.8 Szenario 8

Das Szenario 8 stellt eine Kombination der vorherigen Szenarien dar. Die Ergebnisse der Collating-Phase sind in Tabelle 4.10 abgebildet. Insgesamt besteht das Speicherabbild aus 276 Clustern, wovon es sich bei 210 Clustern um JPEG-Cluster handelt. Die Ergebnisse zeigen, dass von diesen 210 Clustern 184 korrekt als JPEG klassifiziert wurden und 26 fälschlicherweise durch den Algorithmus verworfen werden. Von den insgesamt 66 irrelevanten Clustern wurden lediglich fünf verworfen, welche ein Cluster der Worddatei, die Textdatei, ein ausführbares Programm und die Cluster mit den SOI und EOI Markern der PNG-Datei sind.

Tabelle 4.10: Gegenüberstellung der korrekten Ergebnisse der Klassifizierung und derjenigen des Prototyps (Szenario 8)

	JPEG-Cluster	Keine JPEG-Cluster
Ergebnis Prototyp	1-14, 16-19, 21- 25, 29-53, 55, 58-74, 77-99, 101-10104, 106-111, 113, 125-129, 131, 133-147, 150- 200, 202-274	0, 15, 20, 26-28, 54, 75-76, 100, 105, 112, 114-124, 130, 132, 148-149, 201, 275
Korrektes Ergebnis	3-14, 16-19, 21- 131, 142-224	0-2, 15, 20, 132-141, 225-275

Aus diesen Ergebnissen errechnet sich eine Sensitivität von 87,62%.

$$rp = 184$$
$$fn = 26$$
$$Sensitivität = \frac{184}{184+26} = 0,8762 = 87,62\%$$

Da dieses Szenario sowohl aus JPEG-Dateien, als auch Dateien anderer Dateitypen besteht, kann eine Spezifität bestimmt werden. Diese liegt bei 7,58%.

$$rn = 5$$
$$fp = 61$$
$$Spezifität = \frac{5}{5+61} = 0,0758 = 7,58\%$$

Die Betrachtung der rekonstruierten Bilder, welche in Abbildung 4.8 den Originalbildern gegenüber gestellt sind, ergibt eine optische Identität der Bilder1 und 12. Diese Bilder lagen unfragmentiert im Speicherabbild vor. Die anderen fünf Bilder, die aus mindestens zwei Fragmenten bestanden, konnten nicht korrekt wieder rekonstruiert werden. Dadurch, dass nur zwei von sechs Bildern vollständig rekonstruiert wurden, liegt die Wiederherstellungsrate in diesem Szenario bei 33,33%.

$$Wiederherstellungsrate = \frac{2}{6} = 0,3333 = 33,33\%$$

Setzt man sich mit den Clustern und Segmenten in Tabelle 4.11 auseinander aus denen die Ergebnisbilder erzeugt wurden, ist zu erkennen, dass bei allen Bildern mindestens ein Segment gefunden wurde.

Tabelle 4.11: Gegenüberstellung der Cluster aus denen sich die Bilder zusammensetzen und denen, die der Prototyp zum Zusammensetzen nutzt (Szenario 8)

Dateiname	**Korrekte Cluster des Bildes, aufgeteilt in Segmente**	**Cluster des Endergebnisses, aufgeteilt in Segmente**
Bild1.JPG	[29-47]	[29-47]
Bild2.JPG	[16-19] [21-24] [48-53]	[16-19] [208-224]
Bild5.JPG	[25-28] [54-131]	[25]
Bild8.JPG	[147-207]	[147]
Bild11.JPG	[142-146] [208-224]	[142-146] [150-200] [1-2] [21-24] [48-53]
Bild12.JPG	[3-14]	[3-14]

Bei diesem Segment handelt es sich um das, welches das Cluster mit dem Header beinhaltet. Bei Bild5 und Bild8 besteht das Endergebnis nur aus diesem Segment und Cluster, während Bild2 an das Header-Segment die Bilddaten von Bild11 angehängt wurden. Im Gegensatz dazu wurden Bild11 fälschlicherweise PNG-Daten, die beiden falsch klassifizierten Word-Cluster und die Bilddaten von Bild2 zugeordnet.

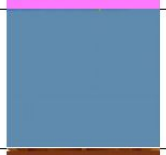
Dateiname	Originalbild	Rekonstruiertes Bild	Hashwert Originalbild	Hashwert Ergebnis
Bild1.JPG			513ec1e0a764ba6a3b104278 b135c787	513ec1e0a764ba6a3b104278 b135c787
Bild2.JPG			5e682a7233d24b80e7d2318c 5829ea81	e50506181c8b0d0c718d5a4b 8d3ef98e
Bild5.JPG			fe0d1e66e95b935dda29ff6928 d439db	bc13b6948c22885f192d3fa5f4 81a085
Bild8.JPG			06eaf46e037c5c0c229be4d5e 2d31791	c86f3ac6b3e780d3d4b8f0c46 d3bbe9c
Bild11.jpg			5df3d3a385ea9ded54c53eb21 d4a2417	b3be9683965b85deff6750bcd 8031838
Bild12.jpeg			250dca89e20eb40cc92896d09 36f4b0a	250dca89e20eb40cc92896d0 936f4b0a

Abbildung 4.8: Gegenüberstellung der Original- und rekonstruierten Bilder mit jeweiligem Hashwert (Szenario 8)

5 Diskussion

Im Rahmen der vorliegenden Masterarbeit wurde eine prototypische Implementierung eines intelligenten Carving Algorithmus zur Rekonstruktion fragmentierter JPEG-Dateien erzeugt. Mit Hilfe von eigens erstellten Testszenarien wurden Ergebnisses zum Evaluieren dieses Prototyps erzielt, auf die in diesem Kapitel eingegangen wird. Anhand der einzelnen Szenarien werden die Ergebnisse interpretiert, Gründe für deren Erhalt erarbeitet und Fehlerquellen herausgearbeitet.

5.1 Szenario 1

Die Ergebnisse des ersten Szenarios zeigen, dass lediglich bei vier von zwölf Bildern alle Cluster richtig als JPEG-Cluster klassifiziert wurden.
Um nachzuvollziehen, wieso einige Cluster nicht klassifiziert werden, ist es wichtig, sich die Cluster und das entsprechende Bild dazu anzuschauen. Da der Collating-Algorithmus bestimmte Filtermethoden aufweist, muss geschaut werden, ob durch diese die JPEG-Cluster herausgefiltert werden. Zu Beginn der Collating-Phase wird nach typischen JPEG-Markern gefiltert. Werden bestimmte Marker anderer Dateitypen gefunden, wird das Cluster verworfen, sind JPEG-Marker in den übrigen Clustern vorhanden, werden diese direkt als JPEG zugehörig klassifiziert. Ebenso läuft es auch mit der Entropie ab. Liegt diese zwischen 7 und 8, handelt es sich laut Algorithmus um ein JPEG-Fragment.
Um herauszufinden, wieso es zur Falsch-Klassifizierung kommt, müssen die verworfenen Cluster betrachtet werden. Dies wird im Folgenden beispielhaft für Bild3 gemacht und die Begründung kann für die anderen Bilder mit fehlenden Clustern übernommen werden. Der JPEG-Header von Bild3 befindet sich komplett im ersten Cluster. Zusätzlich dazu enthält dieses noch Bilddaten. Die folgenden Cluster enthalten somit, bis auf das letzte Cluster, nur die eigentlichen Bilddaten. Aus den Ergebnissen kann man entnehmen, dass die Cluster 1, 3, 4, 5 nicht als JPEG-Cluster erkannt werden. Sucht man in diesen nach den EOI und SOI Markern von PNG-Dateien, wo ein Vorhandensein dazu führt, dass die Cluster direkt verworfen werden, lässt sich kein Treffer ermitteln. Das bedeutet, dass diese Eigenschaft nicht dazu geführt hat, dass eine Falschklassifizierung eingetreten ist.
Bei der Suche nach den definierten JPEG-Markern gibt es in den genannten Clustern keinen Treffer. Das bedeutet, dass eine Klassifizierung als JPEG-Cluster lediglich noch über die Entropie geschehen kann.

Die Entropiewerte dieser Cluster betragen:

$$4,145413581581885$$
$$5,419502682315135$$
$$5,892986648324583$$
$$6,070188487616102$$

Diese Werte liegen unter dem Bereich nach dem die Entropie gefiltert wird, weshalb sie nicht zur Liste der relevanten Cluster zugefügt werden. Die erhaltenen Entropiewerte sind niedriger als die von durchschnittlichen Bilddateien. Da die Entropie die Wahrscheinlichkeit des Vorkommens bestimmter Zeichen angibt, ist bei einer niedrigen Entropie zu erwarten, dass sich Zeichen häufig wiederholen. Betrachtet man Cluster 1 im Hexeditor, wie in 5.1 geschehen, fällt auf, dass sich zum Beispiel die Byte-Werte `0x05`, `0x00`, `0x14`, `0x40` häufig wiederholen. Diese Wiederholung ist eine Begründung für die niedrige Entropie.

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Dekodierter Text
00000000	00	50	01	40	05	00	14	00	50	01	40	05	00	14	00	02	.P.@....P.@.....
00000010	81	85	00	2D	00	14	00	50	20	A0	61	40	05	00	14	00	….-...P a@....
00000020	50	01	40	05	00	14	08	28	18	50	01	40	05	00	14	00	P.@....(.P.@....
00000030	50	01	40	05	00	14	00	50	20	A0	02	80	0A	06	14	00	P.@....P .€....
00000040	50	01	40	05	00	14	00	50	20	A0	02	80	0A	00	28	00	P.@....P .€..(.
00000050	A0	02	81	85	00	14	00	50	20	A0	02	81	85	00	14	00	..….…P ..…....
00000060	50	01	40	82	80	0A	00	28	18	50	20	A0	02	80	0A	06	P.@‚€..(.P .€..
00000070	14	00	50	01	40	05	00	14	00	50	01	40	05	00	14	00	..P.@....P.@....
00000080	50	20	A0	02	81	85	00	14	00	50	20	A0	02	80	0A	00	P ..….…P .€..

Abbildung 5.1: Hexadezimaler Ausschnitt eines Clusters von Bild3.JPG geöffnet mit HxD

Besonders auffällig bei diesem und anderen Bildern, bei denen JPEG-Cluster nicht erkannt wurden, ist, dass es sich in den meisten Fällen um mehrere, aufeinanderfolgende Cluster handelt.
Bei Bild3 wurden die Cluster auf den Header folgend aufgrund fehlender Marker und niedriger Entropie nicht erkannt. Fügt man Header und lediglich die nicht klassifizierten Cluster zusammen und dekodiert diese, ergibt sich das Bild aus Graphik 5.2. Betrachtet man die ersten Bildzeilen, welche dekomprimiert wurden, ist bis auf wenige Bereiche, in denen der Leuchtturm zu finden ist, ein recht einheitlicher blauer Himmel zu erkennen. Diese optische fehlende Diversität könnte dazu geführt haben, dass die Byte-Werte der entsprechenden Cluster ebenfalls nur geringe Unterschiede aufweisen.

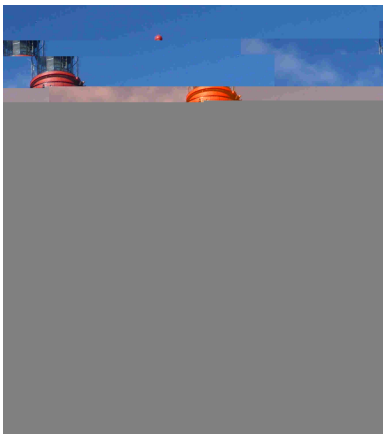


Abbildung 5.2: Dekomprimiertes Bild der nicht erkannten JPEG-Cluster von Bild3

Besonders ist der Einfluss von gleichen Flächen im Bild anhand von Bild6 erkennbar. Die Entropiewerte der ersten Cluster sind auffallend niedrig, wie man Abbildung 5.3 entnehmen kann.

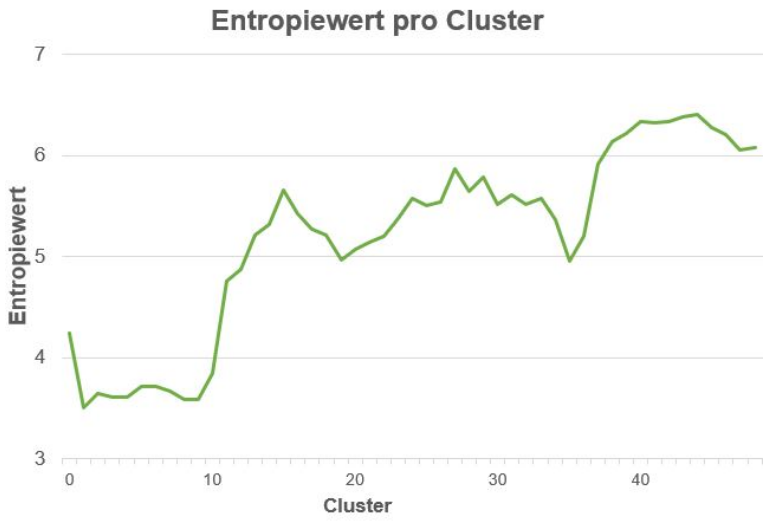


Abbildung 5.3: Graphische Darstellung der Entropiewerte für jedes Cluster von Bild6

Vergleicht man im Zusammenhang dazu die Byte-Werte, geöffnet mit einem Hexeditor, in Abbildung 5.4 fällt eine häufige Wiederholung von bestimmten Byte-Werten auf, wodurch eine niedrige Entropie hervorgerufen wird.

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Dekodierter Text
000002A0	00	14	00	50	01	40	05	00	14	00	50	01	40	05	00	14	...P.@....P.@...
000002B0	00	50	01	40	05	00	14	00	50	01	40	05	00	14	00	50	.P.@....P.@....P
000002C0	01	40	05	00	14	00	50	01	40	05	00	14	00	50	01	40	.@....P.@....P.@
000002D0	05	00	14	00	50	01	40	05	00	14	00	50	01	40	05	00	....P.@....P.@..
000002E0	14	00	50	01	40	05	00	14	00	50	01	40	05	00	14	00	..P.@....P.@....
000002F0	50	01	40	05	00	14	00	50	01	40	05	00	14	00	50	01	P.@....P.@....P.
00000300	40	09	40	82	80	0A	00	28	00	A0	02	80	0A	00	29	80	@.@,€..(. .€..)€

Abbildung 5.4: Hexadezimaler Ausschnitt eines Clusters von Bild6.JPG geöffnet mit HxD

Um die Aussage zu stützen, dass das Auftreten von sehr ähnlichen Bildzeilen die wiederholenden Byte-Werte hervorruft und nicht Ergebnis der Kompression ist, werden zu erst einmal in Abbildung 5.5 die ersten Bildzeilen betrachtet.

Abbildung 5.5: Ausschnitt der ersten Bildzeilen von Bild6.JPG

Bereits bei der optischen Betrachtung sind die ersten Bildzeilen fast identisch blau gefärbt. In Abbilung 5.6 wurde die erste Bildzeile des dekomprimierten Bildes ausgeschnitten und in einem Hexeditor geöffnet, was die Byte-Werte für die RGB-Werte jedes Pixels der ersten Bildzeile zeigt.

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Dekodierter Text
00000000	F4	3A	28	A2	BD	83	CD	8C	AE	58	A2	AB	D5	8A	0A	0A	ô:(¢½ƒÍŒ®X¢«ÕŠ..
00000010	28	A2	80	0A	28	A2	80	0A	28	AA	F4	01	62	8A	28	A0	(¢€.(¢€.(ªô.bŠ(
00000020	02	8A	28	A0	02	8A	28	A0	02	8A	28	A0	02	8A	28	A0	.Š(.Š(.Š(.Š(
00000030	02	8A	28	A0	02	8A	28	A0	02	8A	28	A0	02	8A	28	A0	.Š(.Š(.Š(.Š(
00000040	02	8A	28	A0	02	8A	28	A0	02	8A	28	A0	02	8A	28	A0	.Š(.Š(.Š(.Š(
00000050	02	8A	28	A0	02	8A	28	A0	02	8A	28	A0	02	8A	28	A0	.Š(.Š(.Š(.Š(
00000060	02	8A	28	A0	02	8A	28	A0	02	8A	28	A0	98	CA	E1	45	.Š(.Š(.Š(˜ÊáE

Abbildung 5.6: Hexadezimaler Ausschnitt eines Clusters des dekomprimierten Bild6.JPG geöffnet mit HxD

Anhand des Ausschnitts bestätigt sich, dass häufige Wiederholungen der Byte-Werte auftreten. Beispielhaft ist in Abbilung 5.7 der sich wiederholende Pixel mit den Byte-Werten `0x0A28A2` als die Farbe dargestellt, die er repräsentiert.

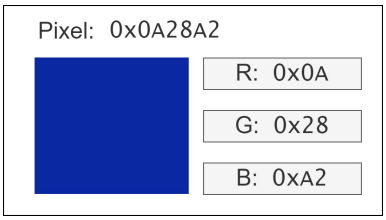


Abbildung 5.7: Visuelle Darstellung der Byte-Werte eines Pixels von Bild6 im RGB-Fabraum

Wendet man seinen Fokus wieder auf die Gesamtheit der Bilder, kann das fälschliche Herausfiltern relevanter Cluster mit dem Fehlen der JPEG typischen Marker begründet werden. Grund hierfür ist, dass bei allen Bildern der Header vollständig im ersten Cluster liegt und der einzige Marker, welcher in den Bilddaten auftauchen kann `0xFF00` ist. Lediglich im letzten Cluster kann zusätzlich der Footer `0xFFD9` enthalten sein. Durch die Reduktion der Marker in den Bilddaten-Clustern auf lediglich einen, sinkt auch die Wahrscheinlichkeit, dass dieser auftaucht.

Eine weitere Möglichkeit, als JPEG-Cluster klassifiziert zu werden, ist, wie bereits erwähnt, über die Entropie möglich. Stellt man einmal die Entropiewerte aller Bilder in Abbildung 5.8 gegenüber, ist ersichtlich, dass nur wenige Cluster weniger Bilder in dem Filterbereich von 7-8 (hier grün markiert) liegen.

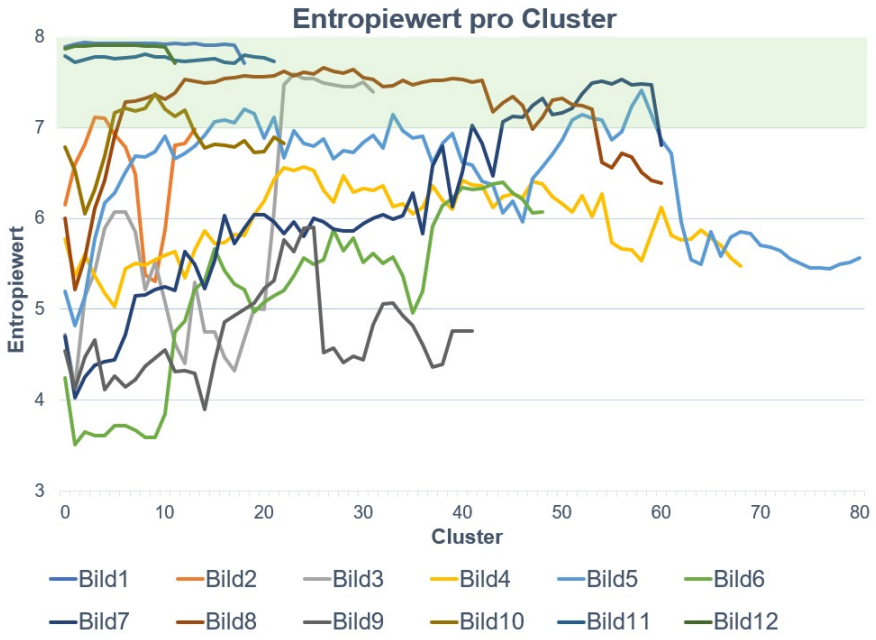


Abbildung 5.8: Graphische Darstellung der Entropiewerte für jedes Cluster von den Bildern 1-12

Aus diesem Grund wird nur ein geringer Teil der Cluster durch die Filterung nach der Entropie als JPEG-Cluster klassifiziert.Aus der Übersicht der Bildeigenschaften 3.8 lässt sich eine Korrelation zwischen der Effizienz der Collating-Phase und der Größe des Bildes zu erkennen. Bei Bildern, welche aus einer geringen Anzahl von Clustern bestehen, ist die Wahrscheinlichkeit höher, dass bei diesen auch alle JPEG-Cluster als solche klassifiziert werden.

Wendet man den Fokus von den Collating-Ergebnissen auf die der Reassembly-Phase, lassen sich optisch wenige Informationen aus den rekonstruierten Bildern ermitteln. Alle Bilder, bei denen alle Cluster korrekt klassifiziert wurden, konnten auch vollständig wiederhergestellt werden. Dies ist dadurch begründet, dass der Prototyp die Cluster dieser Bilder als ein Segment zusammenfügt, da keine Cluster zwischen ihnen fehlen. Ein Segment wird durch das Programm, sofern ein Header vorhanden ist und ein Footer vorkommt, als einzelnes Bild gespeichert.
Da bei den anderen Bildern Cluster nicht erkannt wurden, entstehen auch mehrere Segmente wie in Tabelle 4.2 dargestellt ist. Betrachtet man beispielhaft Bild3, existieren drei Segmente, wovon eins den Header enthält. Die anderen Segmente wurden in der korrekten Reihenfolge angehängt, da der CED-Vergleich zwischen Segment1 und Segment2 einen niedrigeren durchschnittlichen CED-Wert als Segment2 und Segment3 besitzt.

$$CED - Wert\ Segment1\ +\ Segment2 = 0,03841419514781436$$
$$CED - Wert\ Segment1\ +\ Segment3 = 0,1248947236050236$$

Bei dem zweiten Durchlauf, um ein passendes Segment zum Anhängen zu finden, bleibt daher nur noch eins übrig, welches angehängt wird, da keiner der CED-Werte über 1,0 steigt. Aus dem finalen Bild siehe Abbildung 4.1, welches man abschließend erhält, lassen sich in groben Zügen noch Eigenschaften des ursprünglichen Bildes erkennen. Der Leuchtturm ist beispielsweise in seiner Grundstruktur identifizierbar, jedoch durch Farbe und Verschiebung verändert. Dies ist immer da der Fall, wo nur wenige Cluster fehlen und die vorhandenen in richtiger Reihenfolge angeordnet wurden.
Betrachtet man im Anschluss wieder Bild6, welches bereits in der Collating-Phase durch wiederholende Farbwerte aufgefallen ist, lässt sich diese Eigenschaft durch die CED-Werte erneut belegen.

Schaut man sich die durchschnittlichen CED-Werte von Kombinationen zwischen dem Header und einigen Segmenten an, sind diese sehr niedrig und liegen extrem nahe beieinander.

$$0,044631279215475404$$
$$0,063897446135708910$$
$$0,070721738650808510$$
$$0,052463496871142000$$
$$0,050216906649844090$$
$$0,061549040342109285$$
$$0,047724337210148510$$

Durch die Ähnlichkeit der einzelnen Bildzeilen ist es für den Prototyp schwer, das korrekte Segment zu finden, da alle, gemessen an dem Farbunterschied, gut passen würden. In Tabelle 4.3 sind die Segmente zu erkennen, aus denen sich letztlich das Endergebnis zusammensetzt. Dadurch, dass das vierte Segment bereits eigentlich der letzte Cluster des Originalbildes ist, taucht in diesem Cluster der Marker des Footers auf. Dies führt dazu, dass das Bild automatisch gespeichert wird und kein weiteres Segment mehr nach diesem folgt. Gleiches ist bei Bild6 und Bild9 zu erkennen.
Die Ergebnisse von Bild5, Bild10 und Bild8 bestehen nur aus dem Header-Segment, was durch den CED-Wert zu begründen ist. Bei allen Kombinationen von Headersegment und allen anderen Segmenten übersteigt mindestens ein CED-Wert die Grenze von 1, weshalb kein Segment zum Anhängen infrage kommt.
Dekomprimiert man alle Cluster, die als JPEG-Cluster erkannt wurden, beispielhaft für Bild10, erhält man das Ergebnis aus Abbildung 5.9.

Abbildung 5.9: Dekomprimierung der als JPEG erkannte Cluster von Bild10

Durch die fehlenden Cluster lassen sich viele harte Kanten innerhalb des Bildes erkennen, die durch die starken Farbunterschiede der Bildzeilen entstehen. Diese Unterschiede erklären das Vorkommen von CED-Werten über 1.
Aus diesen genannten Gründen kommt es zur fehlerhaften Klassifizierung der Cluster und inkorrekten Rekonstruktion der Bilder.

5.2 Szenario 2

Die Ergebnisse aus dem zweiten Szenario zeigen, dass in diesem speziellen Fall der Prototyp einwandfrei arbeitet. Alle JPEG-Cluster werden gefunden und die nicht-JPEG-Cluster werden bereits in der Collating-Phase verworfen.
Dass alle JPEG-Cluster als diese identifiziert worden sind, ist mit der Wahl des Bildes zu begründen. Wie bereits für dasselbe Bild in Szenario 1 festgestellt, ist in jedem Bildcluster mindestens ein Marker vorhanden, nach welchem eine Filterung im Prototyp erfolgt. Des Weiteren liegt der Entropiewert der Cluster des Bildes im Bereich der Grenze für die Filterung. Neben diesen Eigenschaften stellt auch die Gegenüberstellung von JPEG- und Textdatei-Clustern ein erfolgversprechendes Szenario dar. Dies ist damit zu begründen, dass Text- und Bilddateien anhand mehrerer Eigenschaften sehr große Unterschiede aufweisen.
Beispielsweise handelt es sich bei Textdateien um ASCII-interpretierbare Zeichen und kein regulärer Buchstabe oder Zahl wird durch den für den JPEG-Standard relevanten Hexadezimalen-Wert `0xFF` dargestellt, nach welchem in der Collating-Phase gefiltert wird. Auch in der Textdatei in diesem Szenario ist kein JPEG-üblicher Marker zu finden. Außerdem liegt der Entropiewert der Textdatei mit 5,9 weit unter der Filtergrenze. Der niedrige Entropiewert ist ebenfalls mit der Einschränkung auf die wenigen ASCII-interpretierbaren Zeichen zu begründen, da so die Wahrscheinlichkeit des Auftretens eines Zeichens höher ist.

Durch die fehlerfreie Vorarbeit der Collating-Phase werden die Herausforderungen an die Reassembly-Phase auf das Nötigste reduziert.
Da es sich lediglich um zwei Fragmente handelt, kann es zu keiner falschen Verknüpfung der Fragmente, sondern lediglich zu einem Weglassen des zweiten Fragments kommen. Dies ist allerdings nicht der Fall, da kein CED-Wert größer oder gleich der festgelegten 1.0 Grenze liegt. Auch Graphik 5.10 zeigt nur kleine Entropiewerte ohne große Abweichungen.
Dass es keine großen Ausschläge beim CED-Wert gibt, ist ebenfalls durch die Bildauswahl zu begründen. Es gibt keine harten Farb- oder Formübergänge an den Bildzeilengrenzen, die den CED-Wert durch extreme Pixelunterschiede hochsetzen. Würde beispielsweise das Bild durch horizontale Streifen genau an den Bildgrenzen unterbrochen werden, würde der CED-Vergleich zu keinem erfolgreichen Ergebnis zur Rekonstruktion führen.

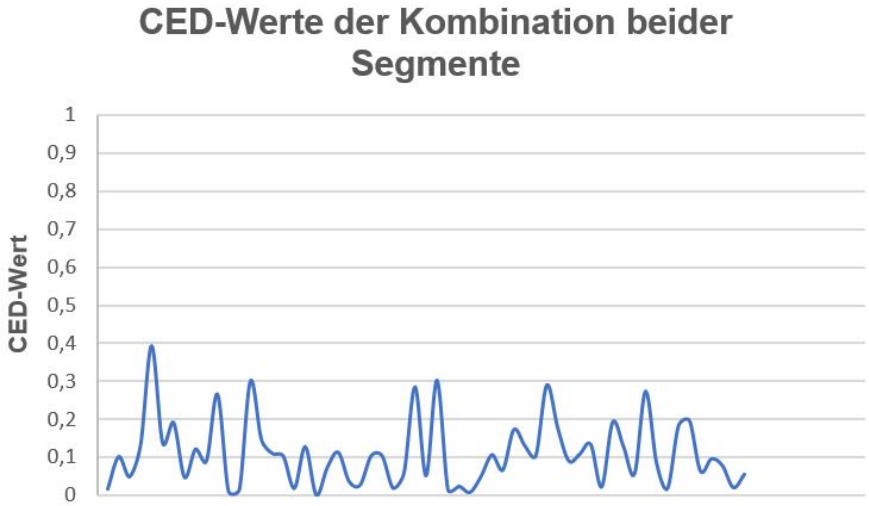


Abbildung 5.10: Graphische Darstellung der CED-Werte der beiden verbundenen Fragmente von Bild1

5.3 Szenario 3

Die Ergebnisse aus dem dritten Szenario zeigen, dass das Vorkommen der Bilddaten vor dem zugehörigen JPEG-Header keinen Einfluss auf das Ergebnis hat. Da die Cluster noch genau identisch mit denen im Szenario 2 sind und somit die relevanten JPEG-Cluster alle Marker aufweisen und die Entropie im Filter liegt, tritt das identische Ergebnis ein. Außerdem konnte das Bild, wie in Szenario 2, in korrekter Reihenfolge und vollständig wiederhergestellt werden. Dass die Veränderung der Header-Position keinen Einfluss auf das Ergebnis hat, lässt sich mit dem Aufbau des Programms begründen. Die als JPEG klassifizierten Cluster werden in Segmente aufgeteilt und diese in Header und Nicht-Header-Segmente aufgeteilt. Anschließend wird vom Header ausgehend nach einem passenden Fragment der Nicht-Header-Segmente gesucht. Das Programm berücksichtigt also nicht die Reihenfolge der Fragmente eines einzelnen Bildes, was die Ergebnisse dieses Szenarios zeigen.

5.4 Szenario 4

Die Ergebnisse des vierten Szenarios sind ebenfalls sowohl nach der Collating-Phase, als auch der Reassembly-Phase korrekt. Dass alle Cluster erkannt wurden, liegt an den verwendeten Bildern, welche wie in Szenario 1 bereits festgestellt, die Voraussetzung oder Entropie erfüllen.
Die korrekte Zusammensetzung der Bilder ist mit dem Algorithmus zum Einteilen der Segmente zu begründen. Neben der Gruppierung von Clustern zu einem Segment nach einer Lücke, wird ebenfalls erfolgreich vor Auftreten eines Headers und nach dem Fund

eines Footers geteilt. Somit befindet sich das vollständig vorliegende Bild in einem einzelnen Segment und wird direkt gespeichert. Im Gegensatz dazu muss für den Header von Bild1 das passende Fragment gefunden werde, was fehlerfrei funktioniert. Dies ist hier wie in Szenario 2 mit dem recht einheitlichen CED zu begründen.

5.5 Szenario 5

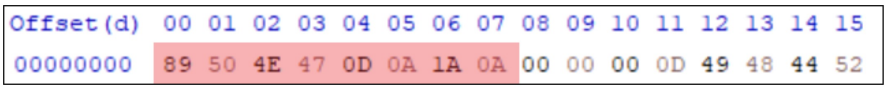
Die Ergebnisse für den Collating-Algorithmus angewendet auf das Szenario 5 zeigen, dass eine gute Sensitivität nicht im Zusammenhang mit einer guten Spezifität steht. Die hohe Sensitivität und niedrige Spezifität belegen den Fokus, welcher beim Entwickeln des Algorithmus gesetzt wurde. Es sollten möglichst alle JPEG-Cluster als diese erkannt werden, dafür wurde bei der Spezifität ein schlechter Wert in Kauf genommen. Das Ergebnis dieses Szenarios belegt dieses verfolgte Prinzip.
Betrachtet man die klassifizierten Cluster, sind sowohl in den Clustern von Bild1, als auch dem nicht JPEG-Bild JPEG typische Marker zu finden. Auch die Entropie beider Bilddateien liegt bei fast jedem Cluster bei 7,9 und damit im Referenzbereich. Lediglich das letzte Cluster des PNG-Bildes fällt durch einen Entropiewert von 4,6 aus dem Raster. Dass dieser Wert niedrig ausfällt, kann dadurch begründet werden, dass es mit Werten aufgefüllt werden musste, welche den Entropiewert natürlich beeinflussen.
Ein Vergleich der Entropiewerte mit aufgefüllten Werten des entsprechenden Clusters und ohne Auffüllung erhält man folgende Ergebnisse:

$$Entropiewert\ des\ aufgefüllten\ Clusters = 4,608091707949116$$
$$Entropiewert\ des\ Originalclusters = 7,893090110547683$$

Diese zeigen, dass Werte innerhalb des Clusters, die nicht zur Datei gehören, einen immensen Einfluss auf den Entropiewert haben können. Dieser Fall tritt ein, wenn das Cluster, welches nicht ganz gefüllt wird, vor der Beschreibung noch leer war oder eine andere Datei, zum Beispiel Textdatei, nur zum Teil überschrieben wurde.
Im Programm wird dieser Fall berücksichtigt, indem JPEG-Dateien, bei denen ein Marker vorkommt, was im Endcluster durch den Footer immer der Fall ist, stets als relevant angesehen werden und die Entropie hierbei nicht berücksichtigt wird. Jedoch könnte die Entropie-Beeinflussung ein Problem für teilweise überschriebene Dateien darstellen. Unvollständige Bilder werden beim Prototyp aber bisher nicht berücksichtigt.
Der ähnliche Entropiewert und der Fund der JPEG-Marker in einer nicht JPEG-Datei ist auf die Eigenschaften von Bilddaten zurückzuführen. Diese sind ungeordnet und es gibt keine Begrenzung an Byte-Werten, die auftreten können.
Die Ergebnisse der Klassifizierung zeigen, dass zwei Cluster von der PNG-Datei korrekt aussortiert wurden. Wenn man sich die Position dieser Cluster anschaut, wird deutlich, dass es sich um Start- und Endcluster der PNG-Datei handelt. In diesen befinden sich die jeweiligen Marker, wie der Ausschnitt in Abbildung 5.11 zeigt. In der Collating-Phase werden durch den Prototyp alle Cluster direkt aussortiert, die diese Marker enthalten,

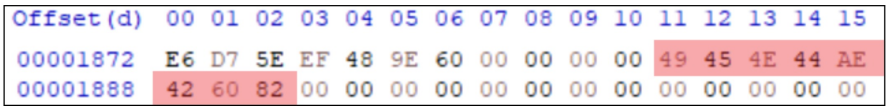
was erklärt, warum die Cluster nicht als JPEG klassifiziert wurden.

Ausschnitt aus Cluster 9:



Offset(d)	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
00000000	89	50	4E	47	0D	0A	1A	0A	00	00	00	0D	49	48	44	52

Ausschnitt aus Cluster 69:



Offset(d)	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
00001872	E6	D7	5E	EF	48	9E	60	00	00	00	00	49	45	4E	44	AE
00001888	42	60	82	00	00	00	00	00	00	00	00	00	00	00	00	00

Abbildung 5.11: Hexadezimaler Ausschnitt der PNG-Marker innerhalb der verworfenen Cluster geöffnet mit HxD

Die Gesamtergebnisse nach Durchlauf der Reassembly-Phase zeigen, dass eine niedrige Spezifität und das falsch-positiv Klassifizieren in diesem speziellen Szenario keinen Einfluss auf das Ergebnis der Reassembly-Phase hat. Das Bild wird trotz der Vielzahl an irrelevanten Clustern komplett rekonstruiert. Für die Verknüpfung des Header-Segments stehen insgesamt neun Segmente zur Verfügung, von denen Prototyp das passende Segment heraussucht. Die Menge an Segmenten lässt sich auf das Vorkommen des JPEG-Markers für das EOI zurückführen. In den PNG-Bilddateien kommt dieser zufällig an mehreren Stellen vor, was dazu führt, dass nach dem Cluster mit diesem Marker das Segment abgeschnitten wird.
Betrachtet man im Anschluss die möglichen Kombinationen genauer, lässt sich herausstellen, dass sieben bereits vor dem Ermitteln der CED-Werte wegfallen. Der genutzte Decoder kann die Kombinationen dieser nicht dekomprimieren, weshalb sie automatisch verworfen werden. Ein Fehler beim Dekomprimieren ist als Fehler innerhalb der Daten zu werten, was in diesem Fall gleichzusetzen mit einem nicht zusammengehörenden Segment ist.
Die anderen beiden Kombinationen können dekomprimiert werden und somit auch die CED-Werte berechnet werden. Betrachtet man die CED-Werte, erkennt man, dass bei der Kombination mit dem PNG-Segment ein CED-Wert mit 4,51 über dem Filterwert liegt, was die Aussortierung dieses Segments erklärt. Am Schluss bleibt aus diesem Grund nur noch das wirklich zugehörige Segment übrig, weshalb das Bild korrekt rekonstruiert wird.

Dieses Szenario zeigt, dass falsch als JPEG klassifizierte Cluster nicht unbedingt einen Einfluss auf die Reassembly-Phase haben müssen. Ein Fehlen eines relevanten Clusters, welches durch die Collating-Phase aussortiert wurde, würde in diesem Szenario größere Auswirkungen als die Hinzunahme von irrelevanten Clustern auf das Ergebnis haben.

5.6 Szenario 6

Die Ergebnisse des Szenarios zeigen erneut, dass das Filtern von Textdateien für den Prototyp kein Problem darstellt. Die Gründe für dieses Ergebnis sind identisch mit denen aus Szenario 2, weshalb darauf nicht erneut genauer eingegangen wird.
Der Fokus dieses Szenarios lag auch auf dem Einfluss der Position des Fragmentationspunkts als auf dem Zusammenfügen der Bildsegmente. Da die Ausgangsbilder, welche man nach Ausführung des Prototyps bei den Szenarien von Szenario 6 erhält, sowohl korrekt als auch inkorrekt zusammengesetzt wurden, wird deutlich, dass ein Zusammenhang zwischen Fragmentationspunkt und einem erfolgreichen Ergebnis besteht.
Bei allen falsch wiederhergestellten Bildern fehlt das erste Segment nach dem Header. Betrachtet man die CED-Werte bei der Kombination aus dem Header-Segment und dem ersten Nicht-Header-Segment, ist bei den CED-Werten am Ende des Nicht-Header-Segments in Abbildung 5.12 ein hoher Ausschlag zu erkennen. Mit 4,8 und 3,8 sind die CED-Werte von der Kombination aus Headercluster und dem Folgecluster sehr hoch. Da im Algorithmus des Prototyps Segmente als möglichen Treffer aussortiert werden sobald einer der CED-Werte über 1 liegt, lässt sich nachvollziehen, wieso das eigentlich richtige Segment aussortiert wird.

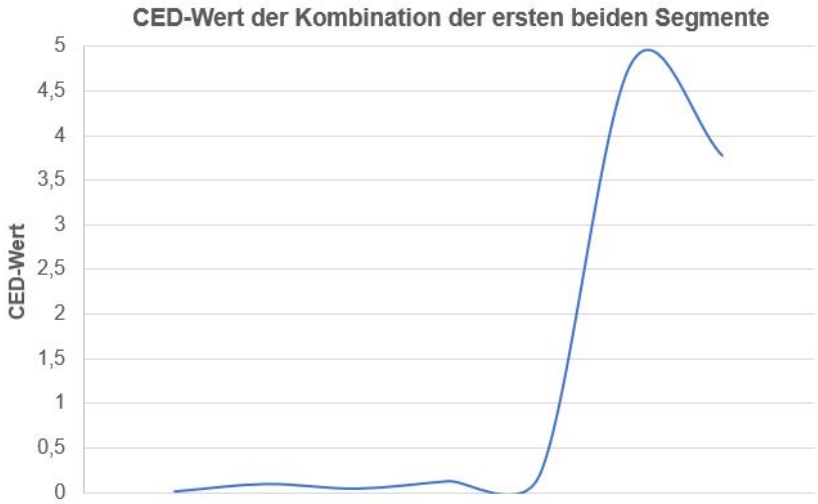


Abbildung 5.12: Graphische Darstellung der CED-Werte der Kombination vom Header-Segment und dem ersten Segment

Ein Wert über eins ist ein Indiz dafür, dass zwei Segmente nicht zueinander gehören, da der Unterschied zwischen den RGB-Werten der Pixel der Bildzeilen groß ist. Je höher der CED oder ED-Wert, desto größer ist auch der Farbunterschied zwischen zwei Bildzeilen.Eine Erklärung, wieso bei eigentlich zusammengehörenden Bildfragmenten einen großen Unterschied bei den RGB-Werten haben, lässt sich durch die Betrachtung der Bildbreite und Clustergröße erklären. Die Bildbreite für Bild1 beträgt 640 Pixel

was mit dem Wert 3 (ein Byte für jeden RGB-Wert) multipliziert 1920 Bytes ergibt. Zum Extrahieren einer Bildzeile müssen also aus dem dekomprimierten Bild für eine Berechnung des CED-Wertes drei Bildzeilen, somit 5.760 extrahiert werden. Dekomprimiert man das Header-Segment, in diesem Fall bestehend aus einem Cluster, kommt es ab Offset 5026_{10}, wie in Abbildung 5.13 erkennbar, zur Wiederholung von immer denselben Byte-Werten.

Offset(d)	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	Dekodierter Text
00004992	FF	00	1E	A7	44	BB	9A	55	54	DC	DF	F2	CB	E6	DA	BF	ÿ..§D»šUTÜßòËæÚ¿
00005008	ED	55	BA	2A	DE	7F	7F	F9	77	F8	FF	00	C0	39	BE	A4	íUº*Þ..ùwøÿ.À9¾¤
00005024	9E	EC	28	A2	8A	F9	D3	B8	28	A2	8A	00	28	A2	8A	00	žì(¢ŠùÓ¸(¢Š.(¢Š.
00005040	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	(¢Š.(¢Š.(¢Š.(¢Š.
00005056	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	(¢Š.(¢Š.(¢Š.(¢Š.
00005072	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	(¢Š.(¢Š.(¢Š.(¢Š.

Abbildung 5.13: Hexadezimaler Ausschnitt des dekomprimierten Header-Clusters von Bild1 geöffnet mit HxD

Dieses Phänomen ist bis Ende der Datei zu erkennen. Es lässt darauf schließen, dass an dieser Position Informationen aus dem darauf folgenden Cluster fehlen, um die Bildzeilen dekomprimieren zu können. Anhand des dekomprimierten Bildes in Abbildung 5.14 lässt sich erkennen, dass innerhalb der Bildzeile Bilddaten fehlen.

Abbildung 5.14: Dekomprimiertes Header-Segment von Bild1

Da 5.760 Bytes, wie zuvor berechnet, für die Berechnung eines CED-Wertes benötigt werden, wird das Ergebnis durch die sich wiederholenden Byte-Werte beeinflusst. Es werden also bei diesem Beispielbild zur Berechnung des ersten CED-Wertes zwei Cluster benötigt. Da lediglich der Unterschied der RGB-Werte zwischen den Bildgrenzen betrachtet werden muss, würde hier der erste CED-Wert ausreichen. Vergleicht man den ersten CED-Wert des eigentlich passenden Segments mit dem angehängten,

$$CED\ zwischen\ Header\ und\ Non-Header-Segment1 = 0,017882131292113357$$
$$CED\ zwischen\ Header\ und\ Non-Header-Segment2 = 0,05334237671626063$$

ist klar zu erkennen, dass der CED-Wert des ersten Segments geringer und die Bilddaten dieser beiden Segmente somit ähnlicher sind.

Da der Prototyp allerdings so entwickelt wurde, dass alle CED-Werte für das Finden nach dem am besten passenden Segment verwendet werden, kommt es in diesem Fall zum Verbinden des falschen Fragments.
Die höheren CED-Werte für die letzten Bildzeilen des eigentlich korrekten Segments lassen sich durch die zuvor festgestellten Gründe erklären. Am Ende des Segments fehlen Informationen vom folgenden Segment oder Cluster, um die Bildzeile mit den einzelnen Pixeln korrekt zu dekomprimieren. Dadurch kommt es zu Wiederholungen von wenigen Byte-Werten, die den CED-Wert beeinflussen. Bei Szenario 6a-c besteht das zweite Segment aus maximal zwei Clustern, während dieses bei Szenario 6d auf sieben Cluster vergrößert wurde. Wie man anhand des Ergebnisses sieht, bewirkt das größere Segment, dass ein korrektes Endergebnis durch eine richtige Verknüpfung der Segmente entsteht.

Bei Szenario 6e sollte zusätzlich noch überprüft werden, ob wenige Cluster auch beim Footer-Segment problematisch werden können. Das Ergebnis zeigt jedoch, dass hierbei keine Probleme entstehen, was sich dadurch erklären lässt, dass die Bilddaten in diesem Segment enden und daher keine falschen CED-Werte aufgrund fehlender Daten entstehen können.
Dieses Szenario zeigt einen Einfluss des Fragmentierungspunkts und der Größe der Segmente auf das Endergebnis durch den Prototyp.

5.7 Szenario 7

Die Begründung für die fehlerfreie Klassifizierung der gesamten Cluster des Szenario 7 lässt sich durch die Bildauswahl begründen, welche in Szenario 1 genauer diskutiert wurde.
Der eigentliche Fokus dieses Szenarios bestand in den Ergebnissen der Reassembly-Phase, da eine Überprüfung erfolgen sollte, ob die Reihenfolge der Header einen Einfluss auf das Ergebnis des Prototyps hat. Aus diesem Grund steht bei diesem Szenario die Beurteilung der Reassembly-Phase im Vordergrund. Die Ergebnisse zeigen, dass die Ergebnisse trotz Positionstausch der Header weiterhin richtig sind. Betrachtet man die CED-Werte der verschiedenen Kombinationen, angefangen mit dem Header Segment von Bild1, liegen die Werte zwar sehr nah beieinander, jedoch ist der durchschnittliche CED-Wert des zugehörigen Fragments minimal kleiner.

$$Header\ Bild1\ +\ Footer\ Bild1 = 0,1107490594700917$$
$$Header\ Bild1\ +\ Footer\ Bild2 = 0,13827937391497624$$

Die Kombination des Header-Segments von Bild2 und beiden verfügbaren Footer-Segmenten liefert ein ähnliches Ergebnis.

$$Header\ Bild2\ +\ Footer\ Bild1 = 0,09319427043185201$$
$$Header\ Bild2\ +\ Footer\ Bild2 = 0,0727642416129355$$

Vergleicht man die CED-Werte der Kombinationen des Header-Segments von Bild1, sowohl mit dem Endfragment von Bild1, als auch mit dem Endfragment von Bild2 in Abbildung 5.15, liegen die Werte sehr nah beieinander und kein Wert überschreitet die 1,0 Grenze.

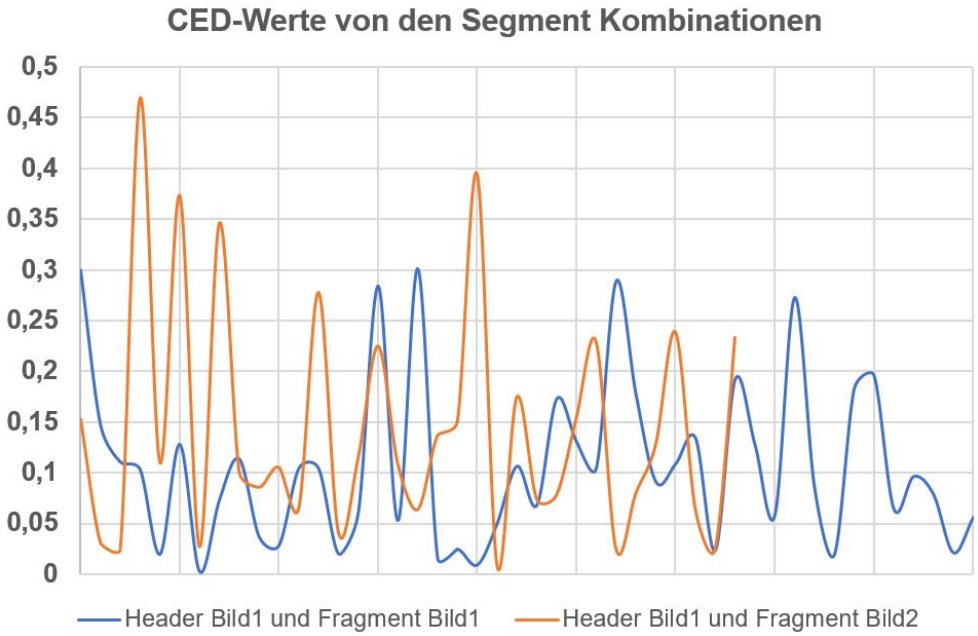


Abbildung 5.15: Vergleich der CED-Werte der Kombinationen aus dem Header-Segment von Bild1 und dem End-Segment von Bild1 und Bild2

Obwohl bei diesem Beispiel kein Einfluss der Position der Header auf das Ergebnis herausgestellt werden konnte, muss damit gerechnet werden, dass dies der Fall sein kann. Der Aufbau des Prototyps, die Header nacheinander abzuarbeiten und nicht parallel, kann dazu führen, dass ein falsches Segment angehängt wird, welches zu einem Header weiter hinten in der Liste, gemessen am durchschnittlichen CED-Wert, besser gepasst hätte. Da nach Benutzung des Segments dieses aus der Liste der verfügbaren gelöscht wird, kann es dem anderen Header, zu welchem es eigentlich passt, nicht mehr angehängt werden kann.

5.8 Szenario 8

Die Ergebnisse der Klassifizierung des Prototyps für Szenario 8 zeigen, dass zwar viele JPEG-Cluster korrekt klassifiziert wurden, jedoch das Aussortieren irrelevanter Cluster unzureichend funktioniert. Bei den fünf aussortierten Clustern handelt es sich um ein Cluster des Wordokuments, die Textdatei, das ausführbare Programm und die Header- und Footer-Cluster der PNG-Datei. Dass das letzte und erste Cluster der PNG-Datei aussortiert wurde, ist mit der Filterung der Marker für Anfang und Ende von PNG-Dateien begründet. Betrachtet man die anderen drei korrekt verworfenen Cluster, sind keine Marker zu finden und die Entropiewerte dieser Cluster liegen unter dem Filter von 7-8. Die Entropiewerte der drei Cluster lauten wie folgt:

$$Cluster\ 0 = 6,179006709563253$$
$$Cluster\ 15 = 5,918176313560457$$
$$Cluster\ 20 = 4,837789323481726$$

Als nächstes erfolgt eine Betrachtung der falsch klassifizierten Cluster, wobei nicht auf die PNG-Cluster und JPEG-Cluster eingegangen wird, da die Begründungen bereits in Szenario 1 und Szenario 5 aufgeführt wurden.
An dieser Stelle ist es interessant zu betrachten, welche Gründe dazu geführt haben, dass zwei Cluster des Worddokuments zu den JPEG-Clustern gefasst wurden. Eine Suche nach den JPEG-Markern in den beiden Clustern ergibt für eines derjenigen einen Treffer. In Abbildung 5.16 ist der Treffer bei der Suche nach den JPEG-Markern rot markiert. Das dargestellte Cluster ist das dritte Cluster der Datei und somit Cluster Nummer 2 im Speicherabbild.

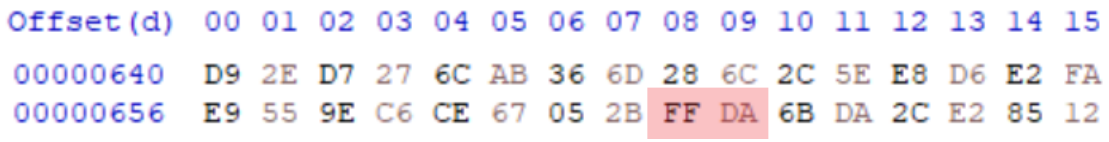


Abbildung 5.16: Hexadezimaler Ausschnitt eines Cluster von dem Worddokument geöffnet in HxD

Der Byte-Wert würde in JPEG-Dateien das *Start-Of-Scan* Segment einleiten, wodurch er relevant für diese Dateitypen ist.
Da das zweite Cluster der Datei ebenfalls nicht verworfen wurden und keine Marker enthält, kann es lediglich durch den Entropiewert als JPEG-Cluster klassifiziert worden sein. Der Entropiewert dieses Clusters liegt mit 7,94 genau in dem Bereich nach dem gefiltert wird, was erklärt, wieso es dem falschen Datentyp zugeordnet wurde.

Nach der Auseinandersetzung mit der Collating-Phase folgt eine Beurteilung der rekonstruierten Bilder. An dem Ergebnis für Bild5 und Bild8 aus Szenario1, dass das Endergebnis nur aus einem Header-Segment besteht, gab es keine Änderung. Die CED-Werte aus allen Kombinationen der Segmente mit den Header-Segmenten zeigen entweder hohe Ausschläge der letzten Bildzeilen über 1 oder eine Berechnung der CED-Werte kann aufgrund von Fehlern beim Dekodieren nicht berechnet werden.
Bei der Betrachtung von dem Ergebnis von Bild2 fällt auf, dass dieses mit dem zweiten Fragment von Bild11 verbunden wurde. Betrachtet man die CED-Werte aus den ersten beiden Fragmenten von Bild2, fällt ein hoher CED-Wert kurz vor Ende, also der Wert, der sich aus den letzten Bildzeilen zusammensetzen, auf. Dieser ist mit 3,58 über dem Grenzwert von 1, was dazu führt, dass das Fragment verworfen wird. Der hohe CED-Wert ist, wie bereits in den vorherigen Szenarien festgestellt, auf die fehlenden Bildinformationen an dem Fragmentationspunkt zurückzuführen. Dadurch, dass das Bild in drei Fragmente aufgeteilt ist, fehlende am Ende des zweiten Fragments Informationen, um Bildzeilen vollständig zu dekomprimieren, weshalb die Werte an solchen Grenzen ausschlagen. Dieser Aspekt erklärt die Tatsache, dass das korrekte Fragment nicht angehängt wurde. Ein Vergleich der CED-Werte an der Grenze zwischen dem ersten und zweiten Fragment, von dem eigentlich korrekten und dem vom Prototyp angehängten zeigt, dass das richtige Fragmente besser gepasst hätte.

$$CED-Wert\ Header-Segment\ und\ zugehöriges\ Fragment = 0,01865235793581998$$
$$CED-Wert\ Header-Segment\ und\ angefügtes\ Fragment = 0,3041955829046037$$

Der Wert liegt deutlich unter dem CED-Wert des Segments, welches durch den Prototyp angehängt wurde. Durch den Vergleichsalgorithmus des Prototyps wird dieser Aspekt jedoch nicht berücksichtigt, was zu falschen Ergebnissen führt.
Diese Fehlkombination führt dazu, dass das Segment, welches eigentlich zu Bild11 gehört, nicht mehr für dieses verfügbar ist. Auch dieses Problem ist auf den Grundaufbau des Programms zurückzuführen, da Segmente lediglich einmal verbunden werden können und für alle folgenden Header nicht mehr zur Verfügung stehen. Außerdem wird Header für Header nach Treffern gesucht, wodurch die Position des Headers Einfluss auf das Ergebnis hat. Ein Positionswechsel des Header-Segments von Bild11 und Bild2 könnte zur Folge haben, dass Bild11 korrekt rekonstruiert werden kann.
Die einzigen Bilder, die identisch zum Originalbild zusammengesetzt wurden, sind Bild1 und Bild12. Beide Bilder lagen unfragmentiert vor, was die korrekte Rekonstruktion erklärt.

6 Fazit

Zusammengefasst lässt sich anhand der erhaltenen Ergebnisse sagen, dass der entwickelte Prototyp in seiner Grundstruktur für die Rekonstruktion fragmentierter JPEG-Dateien genutzt werden kann, jedoch die Lieferung vollständiger und korrekter Ergebnisse gering ist.
Anhand der Szenarien konnte die Fähigkeit zur Klassifizierung von relevanten JPEG-Clustern und zum Verwerfen irrelevanter Cluster überprüft werden. Die Ergebnisse sind hierbei stark von der Größe der vorliegenden Bilddatei und dem Dateityp der nicht JPEG-Datei abhängig. Bei kleinen Bilddateien ist die Wahrscheinlichkeit größer, dass alle Cluster Eigenschaften des Filters der Collating-Phase beinhalten als bei großen. Des Weiteren können Dateien, wie Textdateien, aufgrund ihrer starken Unterschiede im Vergleich zu Bilddaten, erfolgreich als irrelevant erkannt werden. Bei Bildern anderer Dateitypen ist eine komplette Herausfilterung aller Cluster, aufgrund der ähnlichen Entropiewerte und dem Vorkommen der JPEG-Marker, nicht möglich.
Die fehlerhafte Klassifizierung hat einen großen Einfluss auf die Reassembly-Phase, wie die Ergebnisse der Szenarien gezeigt haben. Zusätzlich dazu ist das Endergebnis, das rekonstruierte Bild, von der Position der Fragmentation abhängig. Da die Position der Fragmentation in der Realität variieren kann, ist diese Eigenschaft für den Normalgebrauch problematisch. Allerdings haben die Ergebnisse auch gezeigt, dass die verfolgte Methode der Reassembly-Phase eine Vielzahl von korrekten Ergebnissen liefert, sofern alle Cluster eines Bildes vorhanden und korrekt klassifiziert sind.

Insgesamt zeigen die Ergebnisse, die der Prototyp durch die Verarbeitung der entwickelten Szenarien liefert, dass unter bestimmten Rahmenbedingungen die erwünschten Ergebnisse erzielt werden können. Allerdings ist die Erfolgsquote eher gering und durch den Fakt, dass das zu analysierende Speicherabbild an viele Bedingungen geknüpft ist und lediglich mit kleinen Datenmengen gearbeitet werden kann, ist das Programm bisher nicht für den Allgemeingebrauch geeignet. Durch die weiterführende technische Verbesserung und damit verbundene Vergrößerung von Datenmengen und Bildern, würde die Erfolgsquote des Prototyps enorm sinken oder zu keinem Ergebnis kommen.
Bereits die verhältnismäßig geringe Anzahl an Szenarien verdeutlicht, dass besonders der Algorithmus für die Collating-Phase verbessert werden muss. Viele nicht relevante Cluster werden fälschlicherweise als JPEG-Cluster klassifiziert, während wichtige JPEG-Cluster verworfen werden. Da viele Cluster nicht aussortiert wurden, die für die Weiterverarbeitung nicht benötigt werden, reduziert sich die Datenlast nur gering, wodurch der Zeitaufwand für die Reassembly-Phase steigt. Die Dauer einer Analyse würde unnötig erhöht werden, was für die forensische Polizeiarbeit nicht förderlich ist.

Neben der Collating-Phase des Prototyps weist auch die Reassembly-Phase Schwächen auf. Die Ergebnisse dieser sind extrem von den Ergebnissen der Collating-Phase

abhängig, wie die Szenarien gezeigt haben. Des Weiteren reicht ein Vergleich des durchschnittlichen CED-Wertes und das Rausfiltern des kompletten Segments, sobald ein CED-Wert überstiegen wird, für einen erfolgversprechenden Vergleich nicht aus. Die Ergebnisse zeigen, dass diese Herangehensweise funktionieren kann, aber nicht in jedem Fall muss. Da ein solches Programm im besten Fall jedoch mit allen Szenarien umgehen sollte, reicht dieser Ansatz zur Findung des besten Ergebnisses noch nicht aus.

Bei der Beurteilung muss allerdings berücksichtigt werden, dass es sich um einen ersten Prototyp handelt und daher nicht mit perfekten Ergebnissen gerechnet werden kann. Die Grundstruktur des Prototyps eignet sich für eine Weiterentwicklung und Verbesserung zur Steigerung der korrekten Ergebnisse. Die bereits getesteten Szenarien konnten Fehlerquellen und Probleme aufdecken, die bei einer zukünftigen Überarbeitung des Programms berücksichtigt werden können.
Zusammengefasst lässt sich sagen, dass der Prototyp in seinem bisherigen Zustand unzureichende Ergebnisse für einen Einsatz bei realen Szenarien liefert, allerdings der Grundaufbau und einige Herangehensweisen für eine Weiterverarbeitung genutzt werden können.

7 Ausblick

Unbeachtet der Ergebnisse lässt sich bereits eine Verbesserungsmöglichkeit für den entwickelten Prototyp bei Betrachtung des Programmcodes erkennen. Dieser wirkt unstrukturiert und ist für Außenstehende nur schwer verständlich und schlecht nachzuvollziehbar. Für eine bessere Übersicht müsste zu Beginn eine Struktur erarbeitet und das Programm erneut mit Blick auf diese geschrieben werden. Eine Umformung des Prototyps würde einen höheren Arbeitsaufwand bedeuten, als das Schreiben eines neuen Programms, orientiert an den einzelnen Methoden und Funktionen des Prototyps. Auch die Auslagerung bestimmter Bereiche in andere Klassen kann zur Übersichtlichkeit des Programmcodes beitragen.
Eine weitere ergebnisunabhängige Verbesserungsmöglichkeit besteht in der Ausführung des Programms. Dieses muss über ein Terminal gestartet werden, was für den Normalverbraucher, welcher noch nie Kontakt zu diesem gehabt hat, eine komplizierte Herausforderung sein kann. Zusätzlich dazu muss das Programm Python installiert werden, damit eine Ausführung möglich ist. Für einen ersten Prototyp ist diese Einschränkung vertretbar, allerdings würde eine graphische Benutzeroberfläche die Bedienung um ein Vielfaches vereinfachen. Hinzu kommt, dass ein Python-unabhängiges ausführbares Programm erzeugt wird, welches durch die graphische Oberfläche bedient werden kann. Die Ergänzung hierfür kann ohne ein aufwändiges Umbauen des bereits existierenden Programms erfolgen. Diese Erweiterung dient lediglich der Vereinfachung und ist daher nicht zwingend notwendig, wäre aber für den alltäglichen Gebrauch von Vorteil.

Des Weiteren muss der Benutzer bereits vor Analyse des Speicherabbildes Kenntnis über die Größe der Cluster haben. Zukunftsorientiert könnte dies durch eine Erweiterung des Programms übernommen werden, indem nach Dateisystem-Informationen gesucht wird und daraus die Clustergröße automatisch extrahiert wird. Falls aus verschiedensten Gründen kein Dateisystem mehr vorhanden ist, kann auch weitergreifend ein Algorithmus implementiert werden, der anhand der vorliegenden Daten die Größe identifiziert. Anhaltspunkte könnten hierbei die Dateitypen-Kennungen wie spezifische Marker sein. Diese befinden sich in der Regel am Anfang eines Clusters. Wenn man nach diesen sucht, die Offsets extrahiert und überprüft, bei welcher Clustergröße eine ganze Zahl für den Cluster herauskommt, müsste in der Theorie eine Identifikation der Clustergröße möglich sein. Für eine zukünftige Weiterverarbeitung des bestehenden Programms kann dies als erster Ansatz weiterverfolgt werden.

Weitere Aufgaben für die Zukunft ergeben sich aus den festgesetzten Grenzen des entwickelten Algorithmus, die in Kapitel 3.2 beschrieben sind. Das Programm und damit auch die Testszenarien beinhalten lediglich Dateien des JFIF-Dateiformats. Die Reduktion auf diesen Typ ist mit dem Aufbau und den Markern begründet. Eine Integration

anderer Datentypen hätte mehr Zeit und Arbeit bei der Integration benötigt, die nicht zur Verfügung stand. Weitergehend kann der Prototyp umgestaltet und so ergänzt werden, dass auch weitere JPEG-Dateiformate unterstützt werden. Auch die diversen Kompressionsmodi könnten bei der Weiterentwicklung Berücksichtigung finden.
Für ein möglichst realitätsnah arbeitendes Programm, welches für den Gebrauch in der Forensik verwendet wird, ist diese Kompressionsmodi-Integration mehr als notwendig. Betrachtet man zum Beispiel Delikte, bei denen Bilddaten relevant sind, wie beispielsweise der Bereich der Kinderpornographie, müssen allen JPEG-Dateiformaten analysierbar sein. Es ist mehr als wahrscheinlich, dass Bilddateien mit einer Kamera aufgenommen und nicht ausschließlich aus dem Internet heruntergeladen wurden. Bei Bildern von Kameras handelt es sich um EXIF-Dateien, die bisher nicht berücksichtigt wurden. Durch eine Erweiterung des Programms könnten diese Bildformate ebenfalls implementiert werden.

Zusätzlich zu den JPEG-Formaten kann das Programm allgemein um weitere Dateitypen, nicht nur von Bildformaten, erweitert werden. Es könnte der Fokus von dem Carven von Bilddateien auf einen Smart Carving Algorithmus allgemein für die Analyse von Speicherabbildern erweitert werden. Neben diesen Aspekten können die Aufgaben des Prototyps innerhalb der Preprocessing-Phase, also der Vorbereitung der Daten, ausgearbeitet werden. Beim bisherigen Stand des Programms besteht die hauptsächliche Aufgabe lediglich im clusterweisen Einlesen der Daten. Diese Aufgabe kann durch das vorherige Aussortieren von allokierten Bereichen und Entschlüsseln der Festplatte, sofern notwendig, erweitert werden.

Eine Einschränkung im Programm, die schon beim Programmieren des Prototyps aufgefallen ist, ist der Fakt, dass davon ausgegangen wird, dass die Bilder vollständig vorhanden sind. Alle enthaltenen Bilder können fragmentiert vorliegen, allerdings müssen alle diese Fragmente vorhanden sein. Es kann jedoch vorkommen, dass fragmentierte Bilder, zum Beispiel weil sie zuvor als gelöscht markiert wurden, zum Teil überschrieben wurden. Für die Klassifikation der Cluster ist dieser Fall irrelevant, allerdings wird die Reassembly-Phase durch fehlende Fragmente stark beeinträchtigt. Fehlt ein Fragment innerhalb einer Bilddatei, kann sich der CED-Wert der anderen Fragmente stark unterscheiden. Dadurch können fälschlicherweise Fragmente, die eigentlich zum selben Bild gehören, anderen Bildern aufgrund niedrigerer CED-Werte zugeordnet oder vollständig weggelassen werden. In Szenario 1 zeigen sich bereits die Auswirkungen von fehlenden Fragmenten, aufgrund falscher Klassifizierung, bereits. Durch das Fehlen von Bildteilen kann es zur inkorrekten Anordnung der vorhandenen Fragmente kommen und das erhaltene Bild kann nicht dargestellt werden. Für eine genauere Aussage über die Auswirkungen von fehlenden Fragmenten könnten noch weitere Tests erfolgen. Mit Sicherheit kann aber gesagt werden, dass ein Fehlen des Header-Fragments die Ergebnisse beeinflussen würde. Der Prototyp arbeitet nach dem Prinzip, dass vom Header ausgehend nach zugehörigen Fragmenten gesucht wird. Fehlt dieser Header oder ist er unvollständig, kann das zugehörige Bild nicht rekonstruiert werden und die Fragmente werden

anderen Headern zugeordnet. Es gibt bereits Ansätze, mit denen man übergangsweise einen Header erzeugen kann. Dieses wurde in der vorliegenden Arbeit allerdings nicht untersucht. Für eine Erweiterung des Programms könnte man sich an diesen Ansätzen orientieren und das Problem von fehlenden Headern lösen. Alternativ können auch die Header der anderen Bilder ausgetestet werden, da die Wahrscheinlichkeit beispielsweise bei Bildern, aufgenommen mit derselben Kamera, hoch ist, dass sie ähnlich sind. Hierzu könnte mit dem Tausch der Header von Serienaufnahmen begonnen werden.

Ein weiteres Problem, für dessen Lösung der Prototyp bisher noch nicht konzipiert ist, entsteht, wenn an dem Segment, das den Header der JPEG-Datei enthält, zu wenig Bilddaten hängen. Bei einer geringen Menge von den eigentlichen Bildinformationen kann eine Berechnung des CEDs nicht erfolgen. Einen Einfluss auf eine geringe Menge an Bilddaten kann auch eine große Bildbreite haben. Bilder mit großer Bildbreite benötigen eine Vielzahl an Pixeln, welche über eine Clustergröße hinausgehen. Sind die Segmente jedoch nur ein Cluster groß, ist ein aussagekräftiger Vergleich so gut wie unmöglich. Zukunftsorientiert betrachtet muss dieses Problem behoben werden. Durch die Verbesserung der Kameras wird auch die Pixel- breite weiter ansteigen, wodurch diese Szenarien in den nächsten Jahren immer relevanter werden.

Bei der Implementierung des Prototyps wurde auf eine bereits bestehende Decoder-Bibliothek zurückgegriffen. Für eine Überarbeitung des Programms kann auch eine eigene Decoder-Bibliothek geschrieben werden. Diese hat den Vorteil, dass man einen größeren Spielraum bei Einstellungen des Dekodierens hat und mit dem Anpassen bestimmter Werte arbeiten kann. Neben der Bibliothek wurde auch bei dem Entropiewert, nach welchem die Cluster gefiltert werden, ein bereits ermittelter Wert genommen. Für mehr Genauigkeit kann dieser als Erweiterung selber durch Testdaten bestimmt werden. Die Quantität ist hierbei für die Brauchbarkeit des Ergebnisses entscheidend. Gegebenenfalls könnte damit verbunden bereits eine Unterscheidung der Entropiewerte der verschiedenen JPEG-Dateiformate untersucht werden. Vielleicht unterscheidet sich der Entropiewert abhängig vom gewählten Dateiformat. Ein weiterer Wert, der ermittelt werden kann, um bessere Ergebnisse zu erzielen, ist der CED-Wert. Es könnte ein Referenzwert ebenfalls aus Testdaten berechnet werdeb. Anschließend werden Bildsegmente auf Grundlage dieses Wertes zusammengesetzt, indem nicht der niedrigste durchschnittliche CED-Wert genutzt wird, sondern clusterweise geprüft wird, ob der CED-Wert im Bereich des Referenzwertes liegt.

Weitere Verbesserungsmöglichkeiten für die Zukunft lassen sich aus den Ergebnissen und der Auseinandersetzung mit den Gründen hierfür in der Diskussion herausstellen. Besonders bei der Bewertung der Filtermethoden der Collating-Phase ist zu erkennen, dass diese unzureichend arbeitet. JPEG-Cluster werden nicht als solche erkannt und besonders bei anderen Bildformaten werden fälschlicherweise Cluster als JPEG identifiziert. Dies zeigt, dass die Suche nach den genutzten Markern und der Entropie bisher noch nicht ausreicht. Durch eine Ergänzung von JPEG-Markern und Markern, welche

nicht in einer JPEG-Datei enthalten sind, können die Ergebnisse verbessert werden. Durch eine Gewichtung der Marker, zum Beispiel dadurch, dass EOI und SOI Marker stärker gewertet werden, könnte die Genauigkeit der Filterungsergebnisse verbessert werden. Außerdem kann die Hinzunahme anderer Methoden des Carvings die Korrektheit der Ergebnisse erhöhen. Hier würde sich zum Bespiel die Hinzunahme des Vergleichs der Rate-of-Change-Werte anbieten, was in dieser Arbeit zwar erwähnt wurde, aber keine Verwendung gefunden hat.

Das Programm arbeitet nach dem Prinzip, dass die Cluster, welche in ein Segment aufgeteilt werden, auch zusammengehören. Allerdings muss dies bei einer realen Situation nicht immer der Fall sein. Die Segmentaufteilung erfolgt, wenn eine Lücke zwischen den Clustern vorhanden ist oder ein Header oder Footer zu finden sind. Es kann jedoch sein, dass zwei nicht zusammengehörende Fragmente in ein gemeinsames Segment gefasst werden. Der Prototyp überprüft diesen Fall nicht, was für eine Erweiterung des Programms sinnvoll wäre. Beispielsweise könnten die Cluster nacheinander mit der Berechnung eines CED-Wertes verglichen und anschließend zusammengefügt werden. Alternativ besteht auch die Möglichkeit, die CED-Werte des Segments zu bestimmen und, sobald ein Wert eine bestimmte Grenze übersteigt, auf das Cluster zurückzurechnen, welches an der Position liegt. Anschließend kann an dieser Stelle ein neuer Schnitt innerhalb des Segments erfolgen.

Die Ergebnisse haben ein weiteres Problem mit der verwendeten Vergleichsmethode aufgedeckt. Der durchschnittliche CED-Wert als Vergleich bietet sich nur in manchen Fällen an, da die Zeilen am Ende eines Clusters durch fehlende Informationen hoch sein können. Aus diesem Grund kann zukunftsorientiert ausgetestet werden, ob lediglich eine Betrachtung des CED-Wertes im Grenzbereich der vergleichenden Segmente bessere Ergebnisse liefern würde. Außerdem könnte an dieser Stelle nicht nur ein Vergleich der Ergebnisse für ein Header-Segment mit allen anderen möglichen Segmenten vorgenommen werden, sondern alle Header-Segmente mit allen Kombinationen betrachtet werden. Auf diese Weise könnte man unabhängig von der Header- oder Segmentreihenfolge die besten Treffer bestimmen. Das Programm wurde bisher so konzipiert, dass der Anwender keinen Einfluss auf die Ergebnisse hat. An dieser Stelle könnte man ansetzen und den Anwender miteinbeziehen, indem er durch optische Betrachtung den besten Treffer bestimmt. Des Weiteren könnten die Cluster, die bereits durch die Collating-Phase aussortiert wurden, durch den Nutzer noch einmal gefiltert und ‚zurückgeholt' werden.

Ein allgemeines Problem des Programms liegt neben der Reduktion auf eine geringe Datenmenge in der Geschwindigkeit. Die Geschwindigkeit ist abhängig von der Komplexität des Szenarios, der Vorarbeit der Collating-Phase und der Größe der Daten. Um die Geschwindigkeit zu erhöhen, müsste der Programmcode überarbeitet und mögliche Quellen zur Verbesserung ausgemacht werden.

Neben dem Programm können auch die Testszenarien und deren Bewertung verbessert werden. Die Szenarien decken nur einen Bruchteil der realen Möglichkeiten ab, weshalb durch weitere Szenarien Ergänzungen notwendig sind. Bei der Evaluierungsstrategie ist ein Problem bei der Bewertung des Endergebnisses zu erkennen. Es wird nicht berücksichtigt, inwieweit das Bild dargestellt werden kann, sondern nur bewertet, ob das Bild rekonstruiert werden konnte.

Im Allgemeinen zeigen die erwähnten Aspekte, dass noch großer Verbesserungs- und Erweiterungsbedarf besteht. Dies war allerdings zu erwarten, da das Ziel lediglich darin bestand, eine erste prototypische Implementierung zu entwickeln.

Anhang A: Ausschnitt eines JPEG-Headers

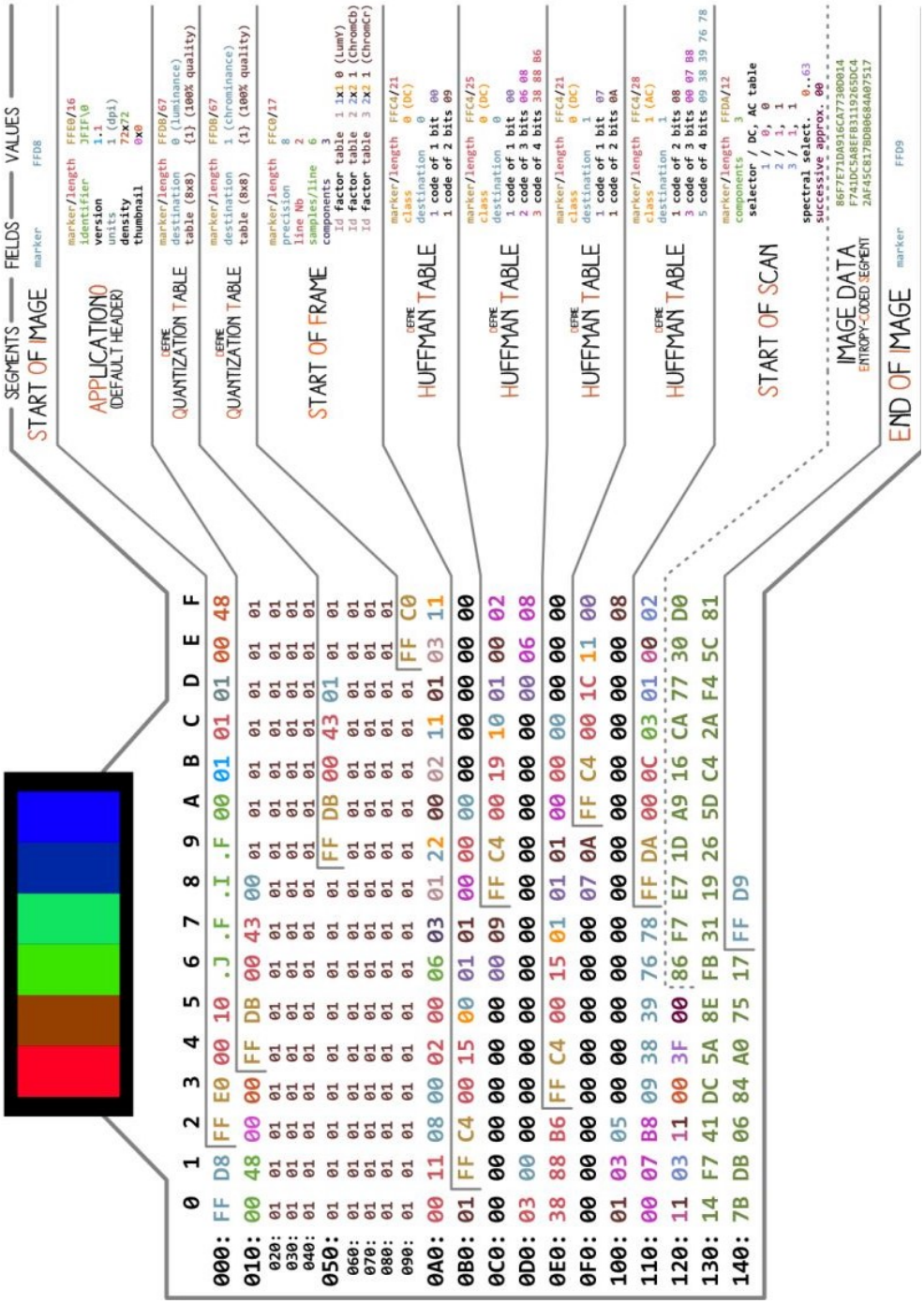


Abbildung A.1: Ausschnitt des JPEG-Headers mit einer detaillierten Beschreibung jedes Segments [49]

Literaturverzeichnis

[1] Bundeskriminalamt, *BKA verzeichnet neuen Höchstwert bei Cyber-Straftaten: Bundeslagebild Cybercrime 2021 veröffentlicht*, 2022. [Online]. Verfügbar unter: https://www.bka.de/DE/Presse/Listenseite_Pressemitteilungen/2022/Presse2022/220509_PM_CybercrimeBLB.html

[2] P. Dalg, "Über 320.000 Fälle im Jahr: Die boomende Branche der Cyberkriminalität," *Der Tagesspiegel*, 11 Mai., 2021. https://www.tagesspiegel.de/wirtschaft/die-boomende-branche-der-cyberkriminalitat-4747507.html (Zugriff am: 5. Apr. 2022).

[3] Bundesministerium des Innern und für Heimat. "Cyberkriminalität." https://www.bmi.bund.de/DE/themen/sicherheit/kriminalitaetsbekaempfung-und-gefahrenabwehr/cyberkriminalitaet/cyberkriminalitaet-node.html (Zugriff am: 5. Apr. 2022).

[4] Bundesamt für Sicherheit in Informationstechnik. "Leitfaden IT-Forensik."https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Cyber-Sicherheit/Themen/Leitfaden_IT-Forensik.pdf?__blob=publicationFilev=1 (Zugriff am: 8. Okt. 2022).

[5] A. Dewald und M. Luft. "Incident-Analyse und Forensik in Docker-Umgebungen." https://www.syssec.at/en/veranstaltungen/dachsecurity2017/papers/DACH_Security_2017_Paper_12A3.pdf (Zugriff am: 8. Okt. 2022).

[6] M. Grotegut, *Windows 7: In Unternehmensnetzen Mit Service Pack 1, IPv4, IPv6.* Berlin, Heidelberg: Springer Berlin / Heidelberg, 2011.

[7] N. Hery-Moßmann. "Was ist eine HDD? Einfach erklärt." https://praxistipps.chip.de/was-ist-eine-hdd-einfach-erklaert_41684 (Zugriff am: 2. Mai. 2022).

[8] T. Eggeling, "Mechanische Festplatten - Eine Ära geht zu Ende" https://www.pcwelt.de/article/1146591/mechanische_festplatten_-_eine_aera_geht_zu_ende-rueckblick.html (Zugriff am: 2. Mai 2022).

[9] T. Tormählen. "Technische Informatik, Teil 8, Kapitel 1: Speicher." https://www.mathematik.uni-marburg.de/~thormae/lectures/ti1/ti_8_1_ger_web.html1 (Zugriff am: 6. Apr. 2022).

[10] S. Follmer. "Festplatten: HDD und SSD - wo liegt der Unter-

schied?" https://praxistipps.chip.de/festplatten-hdd-und-ssd-wo-liegt-der-unterschied_43249 (Zugriff am: 2. Mai. 2022).

[11] G. Viola. "SSD | Solid State Drive | Solid State Disk | Halbleiter-Festplatte." https://www.storage-insider.de/ssd-solid-state-drive-solid-state-disk-halbleiter-festplatte-a-183991/ (Zugriff am: 2. Mai. 2022).

[12] D. Wolski. "Die Technik hinter Solid State Drives (SSDs)." https://www.pcwelt.de/article/1146309/die_technik_hinter_solid_state_drives__ssds_-flash-speicher.html (Zugriff am: 2. Mai. 2022).

[13] D. Janßen. "HDD vs SSD: Warum lohnt sich der Wechsel von HDD auf SSD?" https://blog.mindfactory.de/lohnt-sich-der-wechsel-von-hdd-auf-ssd/ (Zugriff am: 22. Nov. 2022).

[14] M. Lubkowitz, "Hybridfestplatte oder SSD?," *COM-Professional*, 10 Feb., 2014. https://www.com-magazin.de/praxis/festplatte/hybridfestplatte-ssd-236333.html (accessed: Mai. 2, 2022).

[15] S. Follmer. "Hybrid-Festplatte oder SSD? Ein Vergleich." https://praxistipps.chip.de/hybrid-festplatte-oder-ssd-ein-vergleich_47620 (Zugriff am: 2. Mai. 2022).

[16] B. Posey. "Was ist Virtuelle Festplatte (Virtual Hard Drive, VHD)?" https://www.computerweekly.com/de/definition/Virtuelle-Festplatte-Virtual-Hard-Disk-VHD (Zugriff am: 2. Mai. 2022).

[17] E. Uzun und H. T. Sencar, "Carving Orphaned JPEG File Fragments," *IEEE Trans.Inform.Forensic Secur.*, Jg. 10, Nr. 8, S. 1549–1563, 2015, doi: 10.1109/TIFS.2015.2416685 .

[18] T. Schmittner, "Semantic File Carving: Wiederherstellung von Text- und HTML-Dateien," Masterarbeit, Johannes Kepler Universität, Linz, 2011. [Online]. Verfügbar unter: http://www.fim.uni-linz.ac.at/diplomarbeiten/Masterarbeit_Schmittner.pdf

[19] A. Pal und N. Memon, "The evolution of file carving," *IEEE Signal Processing Magazine*, Jg. 26, Nr. 2, S. 59–71, 2009, doi: 10.1109/MSP.2008.931081 .

[20] W. Wiegand, "Verteilte Dateisysteme," Individuelles Projekt, Carl von Ossietzky, Oldenburg, 2005. [Online]. Verfügbar unter: https://uol.de/f/2/dept/informatik/ag/svs/download/thesis/IP-20050729-wiegand-Verteilte_Dateisysteme.pdf

[21] F. Hantelmann, "Dateien und Dateisysteme," in *LINUX für Durchstarter*,

Springer, Berlin, Heidelberg, 1999, S. 99–130. [Online]. Verfügbar unter: https://link.springer.com/chapter/10.1007/978-3-642-60047-0 _6

[22] C. Baun, “File Systems / Dateisysteme,” in *Operating Systems / Betriebssysteme*, Springer Vieweg, Wiesbaden, 2020, S. 111–136. [Online]. Verfügbar unter: https://link.springer.com/chapter/10.1007/978-3-658-29785-5 _6

[23] C. Zimmermann und A. W. Kraas, “Grundlagen Betriebssysteme,” in *Mach*, Springer, Berlin, Heidelberg, 1993, S. 9–26. [Online]. Verfügbar unter: https://link.springer.com/chapter/10.1007/978-3-642-77701-1 _2

[24] M. Mulazzani, S. Schrittwieser, M. Leithner, M. Huber, E. Weippl. “Cloud Speicherdienste als Angriffsvektoren.” https://publik.tuwien.ac.at/files/PubDat_202730.pdf (Zugriff am: 3. Mai. 2022).

[25] Datenrettungsspezialist.de. “SSD defragmentieren: besser nicht – die Alternative.” https://www.datenrettungsspezialist.de/ssd-defragmentieren-schaedlich-oder-nicht-alternative/ (Zugriff am: 3. Mai. 2022).

[26] S. A. Sari und K. M. Mohamad, “A Review of Graph Theoretic and Weightage Techniques in File Carving,” *J. Phys.: Conf. Ser.*, 2020, doi: 10.1088/1742-6596/1529/5/052011 .

[27] G DATA Ratgeber. “IT-Ratgeber: Was ist Defragmentieren?” https://www.gdata.de/ratgeber/was-ist-eigentlich-defragmentieren (Zugriff am: 7. Apr. 2022).

[28] V. van der Meer, H. Jonker, G. Dols, H. van Beek, J. van den Bos und M. van Eekelen, “File Fragmentation in the Wild: a Privacy-Friendly Approach,” in *2019 IEEE International Workshop on Information Forensics and Security (WIFS),* Delft, Netherlands, 2019, S. 1–6, doi: 10.1109/WIFS47025.2019.9034981

[29] R. Poisel, S. Tjoa, and P. Tavolato, “Advanced file carving approaches for multimedia files,” *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.*, vol. 2, pp. 42–58, 2011.

[30] C. Rousseau, Y. Saint-Aubin, H. Antaya, Ascah-Coallier und M. Stern, Hg. *Mathematik und Technologie* (Springer-Lehrbuch). Berlin, Heidelberg: Springer Spektrum, 2012.

[31] W. Burger und M. J. Burge, *Digitale Bildverarbeitung: Eine Einführung mit Java*. Berlin, Heidelberg: Springer, 2009.

[32] A. Hofstätter, “Die JPEG-Kompression : eine Simulation für Lehrerinnen und

Lehrer," Diplomarbeit, Universität Linz, Linz, 2015. [Online]. Verfügbar unter: https://epub.jku.at/obvulihs/content/titleinfo/881702?lang=en

[33] J. C. Busch et al., "Projektgruppe STEGO," Endbericht, C.v.O Universität, Oldenburg, 2009. [Online]. Verfügbar unter: http://www.boles.de/teaching/pg_fb10/endberichte/2009/stego.pdf

[34] L. Schwarz. "Unterschied von jpg und jpeg leicht erklärt." https://www.helpster.de/unterschied-von-jpg-und-jpeg-leicht-erklaert_126003 (Zugriff am: 4. Apr. 2022).

[35] T. Aschermann. "Bildformate: JPG und JPEG - Was sind die Unterschiede?" https://praxistipps.chip.de/bildformate-jpg-und-jpeg-was-sind-die-unterschiede_12229 (Zugriff am: 4. Apr. 2022).

[36] FILExt. "Was ist eine JPE Datei und wie man sie öffnet." https://filext.com/de/dateiendung/JPE (Zugriff am: 4. Apr. 2022).

[37] File-Extension.org. "Dateiendung JPE - File Extension JPE - Einfache Hinweise, wie man die JPE-Datei öffnet." https://www.file-extension.org/de/extensions/jpe (Zugriff am: 4. Apr. 2022).

[38] S. Kurz, "7. Einführung in die Bildbearbeitung," in *Digital Humanities*. Wießbaden: Springer Fachmedien, 2015

[39] T. Hoffmann-Walbeck *et al.*, *Standards in der Medienproduktion*. Berlin, Heidelberg: Springer Vieweg, 2013.

[40] G. Randers-Pehrson. "JNG (JPEG Network Graphics) Format Version 1.0." http://www.libpng.org/pub/mng/spec/jng.html (Zugriff am: 4. Apr. 2022).

[41] S. Thesmann, "Medienobjekte," in *Einführung in das Design multimedialer Webanwendungen: Mit 29 Tabellen* (Studium), S. Thesmann, Hg., 1. Aufl. Wiesbaden: Vieweg + Teubner, 2010, S. 391–496.

[42] P. Bühler, P. Schlaich und D. Sinner, "Bildtechnik," in *Digitales Bild: Bildgestaltung - Bildbearbeitung - Bildtechnik* (Bibliothek der Mediengestaltung - Aufbauset Digitalmedien), P. Bühler, P. Schlaich und D. Sinner, Hg., Berlin, Heidelberg: Springer Vieweg, 2017, S. 15–36.

[43] M. Stirner. "JPEG - Das Bildformat Teil 1: Theorie und Grundlagen." https://www.burosch.de/technische-informationen/339-jpeg-das-bildformat-teil-1-theorie-und-grundlagen.html (Zugriff am: 29. Apr. 2022).

[44] UC DAVIS. “JPEG Image Compression Systems.” https://www.ece.ucdavis.edu/cerl/reliablejpeg/compression (Zugriff am: 29. Apr. 2022).

[45] IONOS Digital Guide. “Progressive JPEGs: Eine Einführung.” https://www.ionos.de/digitalguide/websites/webdesign/progressive-jpeg/ (Zugriff am: 17. Mrz. 2022).

[46] D. H. Boggs. “Use progressive JPEGs to reduce loading times of your web pages.” https://www.davidhboggs.com/blog/how-to/use-progressive-jpegs-to-reduce-loading-times-of-your-web-pages-2782-thread.html (Zugriff am: 29. Apr. 2022).

[47] T. Strutz, *Bilddatenkompression: Grundlagen, Codierung, Wavelets, JPEG, MPEG, H.264*, 4. Aufl. (SpringerLink Bücher). Wiesbaden: Vieweg+Teubner, 2009.

[48] R. Menn, “JPEG,” *TecChannel Workshop*, 30 Dez., 1999. https://www.tecchannel.de/a/jpeg,401190 (Zugriff am: 11. Mrz. 2022).

[49] M. Y. U. Khalid. “Understanding and Decoding a JPEG Image using Python” https://yasoob.me/posts/understanding-and-writing-jpeg-decoder-in-python/ (Zugriff am: 11. Mrz. 2022).

[50] A. Gebhard, D. Paulus, M. Dege, “Grobe Lokalisation und Verfolgung von Patientengesichtern und Gesichtsmerkmalen in Farbbildfolgen in Echtzeit,” Friedrich-Alexander-Universität, Erlangen - Nürnberg. [Online]. Verfügbar unter: http://www5.informatik.uni-erlangen.de/Forschung/Publikationen/1998/Gebhard98-GLU.pdf

[51] N. Mohammadi, “Implementierung eines neuen JPEG-basierten Steganographie-und Kryptographie-Verfahrens mit Mikrocontroller-realisierter Webseite,” Bachelorthesis, Hochschule für Angewandte Wissenschaften, Hamburg, 2017. [Online]. Verfügbar unter: https://reposit.haw-hamburg.de/bitstream/20.500.12738/7995/1/BachelorarbeitNoushin Mohammadi.pdf

[52] E. Günther, “Entwicklung eines JPEG-Dateianalysators,” Diplomarbeit, Hochschule für Technik und Wirtschaft, Dresden, 2021. [Online]. Verfügbar unter: https://htw-dresden.qucosa.de/api/qucosa%3A76129/attachment/ATT-0/

[53] T. Merz, “Gut verpackt: Drucken von JPEG-Bildern mit PostScript Level2,” S. 236–243. https://ctan.uib.no/support/jpeg2ps/jpeg2ps.pdf (Zugriff am: 21. Nov. 2022).

[54] G. Born, *Dateiformate - die Referenz: Tabellenkalkulation, Text, Grafik, Multimedia, Sound und Internet*, 1. Aufl. (Galileo computing). Bonn: Galileo Press, 2001.

[55] A. Burri, “JPEG-3D Codec Software,” Semesterarbeit, Institut für Integrierte Systeme, Eidgenössische Technische Hochschule, Zürich, 2001. [Online]. Verfügbar unter: http://www.purrly.ch/download/thesis/j3d/bericht.pdf

[56] C. Meinel und H. Sack, “Multimediale Daten und ihre Kodierung,” in *Digitale Kommunikation: Vernetzen, Multimedia, Sicherheit* (X.media.press), C. Meinel und H. Sack, Hg., Dordrecht, Heidelberg: Springer, 2009, S. 161–305.

[57] M. Rehbein, “Digitalisierung,” in *Digital Humanities: Eine Einführung*, F. Jannidis, H. Kohle und M. Rehbein, Hg., Stuttgart: J.B. Metzler Verlag, 2017, S. 179–198.

[58] B. Jackson, “JPG vs. JPEG: Die gängigsten Bilddateiformate verstehen,” Kinsta, 23 Sep., 2019 (Zugriff am: 17. Mrz. 2022).

[59] R. Zwönitzer, T. Kalinski, H. Hofmann, A. Roessner, and J. Bernarding, “Verlustbehaftete Bilddatenkompression,” in *RöFo-Fortschritte auf dem Gebiet der Röntgenstrahlen und der bildgebenden Verfahren*, vol. 180, p. WS_101_1, 2008.

[60] O. Halvani und T. Knierim. “Digitale Forensik: Carving und semantische Analyse in der digitalen Forensik.” https://docplayer.org/12963624-Digitale-forensik-carving-und-semantische-analyse-in-der-digitalen-forensik-oren-halvani-tamara-knierim.html (Zugriff am: 4. Apr. 2022).

[61] R. R. Ali, K. M. Mohamad, S. Jamel und S. K. A. Khalid. “A REVIEW OF DIGITAL FORENSICS METHODS FOR JPEG FILE CARVING.” http://www.jatit.org/volumes/Vol96No17/17Vol96No17.pdf (Zugriff am: 4. Apr. 2022).

[62] R. K. Pahade, B. Singh und U. Singh, “A Survey on Multimedia File Carving,” *IJCSES*, Jg. 6, Nr. 6, S. 27–46, 2015, doi: 10.5121/ijcses.2015.6603 .

[63] H. Lee und H.-W. Lee, “Block based Smart Carving System for Forgery Analysis and Fragmented File Identification,” *Journal of Internet Computing and Services*, Jg. 21, Nr. 3, S. 93–102, 2020. doi: 10.7472/jksii.2020.21.3.93 . [Online]. Verfügbar unter: https://koreascience.kr/article/JAKO202019962560384.page

[64] A. Afrizal, N. D. W. Cahyani und E. M. Jadied, “Analysis and Implementation of Signature Based Method and Structure File Based Method for File Carving,” *Indonesian Journal on Computing (Indo-JC), Vol. 6, 2021, Art. Nr. No. 1, doi: 10.34818/INDOJC.2021.6.1.457 .*

[65] K. M. Mohamad und M. M. Deris, “Fragmentation Point Detection of JPEG Images at DHT Using Validator,” in *Future generation information technology: First international conference, FGIT 2009, Jeju Island, Korea, December 10-12, 2009* ; proceedings (Lecture Notes in Computer Science 5899), Y. Lee, T. Kim, W. Fang und D. Ślezak, Hg., Berlin, Heidelberg: Springer, 2009, S. 173–180.

[66] N. Škrbina und T. Stojanovski, “Using Parallel Processing for File Carving,” [Online]. Verfügbar unter: https://www.researchgate.net/publication/232804768_Using_Parallel_Processing_for_File_Carving

[67] B. Högel. “Entropie (Informationstheorie).” https://www.biancahoegel.de/computer/lexikon/entropie.html (Zugriff am: 8. Okt. 2022).

[68] B. Schildendorfer, “Carving fragmented JPEG images: Content-based file carving of non-contiguously fragmented JPEG images,” Master Thesis, Department for Information Security at the University of Applied Science, St. Poelten, 2012. [Online]. Verfügbar unter: https://phaidra.fhstp.ac.at/open/o:2100

[69] Y. Tang et al., “Recovery of heavily fragmented JPEG files,” *Digital Investigation,* Jg. 18, S108-S117, 2016, doi: 10.1016/j .diin.2016.04.016.

[70] A. Ravi, T. R. Kumar and A. R. Mathew, “A method for carving fragmented document and image files,” *2016 International Conference on Advances in Human Machine Interaction (HMI)*, 2016, S. 1-6, doi: 10.1109/HMI.2016.7449170.

[71] Pexels GmbH. “Stock Fotos, Lizenzfreie Bilder Kostenlose Bilder.” https://www.pexels.com/de-de/ (Zugriff am: 8. Okt. 2022).

[72] M. Kalisch und L. Meier, “Klassifikation,” in *Logistische Regression*, Springer Spektrum, Wiesbaden, 2021. [Online]. Verfügbar unter: https://link.springer.com/chapter/10.1007/978-3-658-34225-8 _5

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht. Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt oder anderweitig veröffentlicht.

Kira Marie Kähler

Mittweida, 29. November 2022