João Miguel Nunes Veloso

Bachelor in Computer Science and Engineering

Automated support tool for forensics investigation on hard disk images

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Informatics Engineering

Adviser: João Leitão, Assistant Professor,
NOVA University of Lisbon

Examination Committee

Chairperson: Doutor Paulo Orlando Reis Afonso Lopes
Raporteurs: Doutora Maria Dulce Pedroso Domingos
Doutor João Carlos Antunes Leitão

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

February, 2021

This document was created using the (pdf)LaTeX processor, based in the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

To my mother.

Acknowledgements

The work developed in the past few month, and reflected in this thesis would not have been possible if not for the hard work of several people whom i had the privilege to meet along my path. I would firstly thank João Leitão for all the time, advice, knowledge, dedication and guidance not only during the development of this work but also throughout my studies.

My mother, Paula Nunes for all the love, guidance, help and support in every step of my life. António Ramalhete, for the adventures and experiences.

My thanks also extends to Rui Tomás, for an amazing summer internship and the mentoring and friendship that followed to this day.

I would like to thank my colleges and friends, Luís Grilo and David Romão whom I work daily during the course and provided an excellent environment with laughter and learning.

Thanks to Grupo Jerónimo Martins, Blocks Technology and Deloitte for the growth opportunities personal and professional that I have been given throughout my academic career. NEEC for the opportunity to develop my leadership skills and create a positive impact in our faculty.

Finally, i would like to extend a special acknowledgement to Mariana. Thanks for the companionship and for making me a better person each day.

May i continue to grow and learn as much in the coming years as in these last ones.

João

Abstract

The advent of computers and the massification of mass storage has led to the generation of large volumes of data by individuals. This, in turn, created a challenge for the world's law enforcement agencies, who are many times faced with the need to look, in a timely manner, at someone's hard drive to find offensive multimedia contents, such as child abuse digital proof of. However, the large amount of data found in captured laptops of suspects turns this task unfeasible, unpractical, and very time-consuming since the investigator has limited time and resources.

When we take into consideration that a suspect might be from a foreign country, the time window available for a criminal investigation is even shorter. Furthermore, some criminals might have some technical knowledge and might try to hide evidence, making the task of the investigator even more challenging.

In this thesis we developed a new system that can assist in criminal investigators in looking for evidence on images of hard drives extracted from the devices of suspects. In this work we assumed that the content on the hard disk images is not encrypted and we focused on child abuse multimedia content. Although the platform was designed to be extensible as to handle other types of content in the future, some examples are predicting the location where a picture was taken to analyzing other types of documents. The tool automatically indexes the contents of the hard drive, and looks for potential evidence that is important to the criminal investigators. This tool is able to help not only police departments but several other organizations such as the Internet Hotline Providers in Europe (INHOPE) and Non-governmental organization (NGO) fighting for children.

Keywords: Digital Forensic, Forensic Science, Picture Triage, Machine learning, Searching

Resumo

A revolução tecnológica das últimas décadas trouxe com ela a massificação do armazenamento em massa, isto levou a uma enorme produção de dados por parte de indivíduos. Consequentemente, emergiu um desafio para os investigadores, que muitas vezes se deparam com a necessidade de analisar, em tempo útil o disco rígido de um suspeito para encontrar conteúdos multimédia ofensivos, como por exemplo conteúdos digitais de abuso infantil. No entanto, a grande quantidade de dados encontrados em computadores ou dispositivos confiscados de suspeitos torna esta tarefa inviável, impraticável e muito demorada, uma vez que o investigador tem tempo e recursos limitados.

Quando temos em consideração que um suspeito pode ser de um país estrangeiro, a janela de tempo disponível para uma investigação criminal é ainda mais curta, em Portugal. Os investigadores criminais possuem apenas 48 horas para apresentar uma acusação. Infelizmente, este não é o único problema que os investigadores enfrentam, alguns criminosos possuem conhecimento técnico e podem tentar esconder provas, tornando a tarefa do investigador ainda mais desencorajadora.

Nesta tese, desenvolvemos um novo sistema capaz de ajudar os investigadores forenses na procura de provas em imagens de discos rígidos extraídas dos dispositivos dos suspeitos. Neste trabalho assumimos que o conteúdo das imagens dos discos rígidos não se encontra cifrados, o nosso foco foi em conteúdo multimédia relacionado com abuso infantil, embora a plataforma possa ser estendida para lidar com outros tipos de conteúdo no futuro. Alguns exemplos são a possibilidade de prever a localização onde uma fotografia foi tirada e o processamento de outros formatos de documentos que não sejam fotografias. A ferramenta indexa automaticamente o conteúdo do disco rígido e procura potenciais evidências que sejam importantes para os investigadores criminais. Esta ferramenta é capaz de ajudar não apenas os departamentos policiais, mas várias outras organizações como a INHOPE e Organização não governamental (ONG)s que lutam contra este tipo de material.

Palavras-chave: Forense Digital, Ciência Forense, Triagem de Imagens, Machine learning, Pesquisa

Contents

List of Figures

List of Tables

Acronyms

AES Advanced Encryption Standard.

API Application Programming Interface.

AUC Area Under the Curve.

BLOB Binary large object.

CI/CD Continuous integration and Continuous delivery.

CLI Command-line interface.

CNN Convolutional neural network.

CSV Comma-separated values.

DAC Discretionary access control.

DES Data Encryption Standard.

DEX Deep EXpectation.

fn False negative.

FP False positives.

fpr False Positive Rate.

FUSE Filesystem in Userspace.

GUI Graphical user interface.

HDD Hard Disk Drive.

HFS Hierarchical File System.

HFS Plus Hierarchical File System Plus.

HTTP Hypertext Transfer Protocol.

HTTPS Hyper Text Transfer Protocol Secure.

INHOPE Internet Hotline Providers in Europe.

JPA Java Persistence APIs.

JSON JavaScript Object Notation.

JWT JSON Web Token.

MAC Mandatory access control.

MBR Master Boot Record.

MFT Master File Table.

MVC Model-View-Controller.

NGO Non-governmental organization.

NIST The National Institute of Standards and Technology.

NSFW Not safe for work.

NTFS New Technologies File System.

OAS OpenAPI Specification.

ONG Organização não governamental.

OS Operating system.

POJO Plain Old Java Objects.

RAID Redundant Array of Inexpensive Disks.

RAM Random Access Memory.

RBAC Role-based access control.

ROC Receiver Operator Characteristic.

ROM Read-only memory.

RSA Rivest-Shamir-Adleman.

SEO Search Engine Optimized.

SSD Solid State Drive.

SSL Secure Sockets Layer.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

tpr True Positive Rate.

UI User interface.

URL Uniform Resource Locator.

XIRAF XML-based indexing and querying for digital forensics.

XML Extensible Markup Language.

Chapter 1

Introduction

> The only impossible journey is the one you never begin.
>
> *Anthony Robbins*

1.1 Context

The current technological revolution of our time is changing our lives, making them more comfortable, more connected, and increasingly digital. However, criminals have also started to use more of these technologies in their activities. Domo, a business cloud service analyst, estimates that more than 2.5 quintillion bytes of data are created every single day, and forecast that this estimate will grow even more, they also predict that by 2020 1.7MB of data will be created every second for every person on earth[1].

This has made the work of law enforcement agents a constant battle to search and filter increasing amounts of data, particularly when looking for evidence on drives that were apprehended from suspects. This is in part due to the decrease in storage cost and the increase of crimes committed using digital devices.

In this work we describe a novel approach towards processing, managing, filtering, and querying from disk images obtained from potential suspects and presenting them in a way that is simple for law enforcers to use, helping them to locate evidence on time.

1.2 Motivation

Every day more and more crimes are committed using computers, they are also getting increasingly sophisticated. This poses a real challenge for law enforcement professionals to keep up.

Upon seizing a suspect's hard drive detectives and analysts must guarantee forensic integrity, so they create forensic copies of the digital devices. These images are byte by byte copies of the original.

In the next phase, called the extraction process, they inevitably fall short, because there are not many adequate software solutions to assist in this process furthermore they require significant technical knowledge and their use has costs that are not negligible.

The ideal solution would be simple to use and free, here is where our work intends to fill in the gap. Our main goal is to provide free software that can examine the contents of hard disk images and process them in a scalable way, despite the high volumes of digital material. This process must happen in the useful time since suspects cannot be held for an indefinite amount of time. Additionally, it should maximize the trace coverage, meaning the investigator can trace back the evidence and prove it was not planted. This was done by comparing the hash of an extracted file to the hash of when the image as generated.

Finally, our solution should minimize the necessity of specialized technicians, being this restriction motivated by a limited budget of law enforcement agencies.

1.3 Main Results

In this work, we developed and made available a new tool for the forensics community to automate the analysis and data extraction from disk images.

We created a novel solution that can extract files from disk images, both present and deleted, and extract insights from them. In particular, in this work, we ensured that, when an image is analyzed, we can determine with an accurate degree and a confidence interval if it contains child pornographic material.

The developed solution was planned to be tested in a real-world scenario in conjunction with a local law enforcement agency. However due to recent events regarding the spread of the COVID-19 pandemic a field test was not possible and we had to replace this phase with datasets, more details will be described in Section 5.6.

1.4 Document Structure

The remainder of the document is structured as follows:

- **Chapter 2** describes the Related Work that was studied to elaborate this work. Focusing on underlying systems that manage relevant data and how they operate.

- **Chapter 3** provides a detailed view of the requirement and the main components necessary for building the proposed solution, This chapter also presents a high-level description of the architecture.

- **Chapter 4** discusses our implementation and how it tackles the objectives identified in the previous chapter.
- **Chapter 5** presents the expected evaluation of our solution. It details how we generated the testing datasets, how we conducted the experiments, and the obtained results.
- **Chapter 6** analyses the results of our solution and concludes this document by discussing the future work.

Chapter 2

Related Work

> Great things in business are
> never done by one person.
> They're done by a team of people.
>
> *Steve Jobs*

This chapter presents the related work that served as the basis for our own proposed architecture.

There are two leading similar solutions to the one proposed in this document. The first is XIRAF[2] which applies automatic forensics analysis tools to evidence files from hard disk images. Once an image is inserted into the system, XIRAF executes several forensic tools from its tool repository, wraps the output and stores them in the form of Extensible Markup Language (XML) for later reporting.

The second and most recent solution, is HANSKEN [3], developed by the Netherlands Forensic Institute, it builds on XIRAF and turns the previous stand-alone architecture into a Digital Forensics as a Service (DFaaS) [4]. Initially published in [5], this system was considered as a "game-changer". It addresses challenges such as: i) Security, ii) Privacy, iii) Transparency, iv) Multi-tenancy, v) Future proof, vi) Data retention, vii) Reliability, and viii) High availability.

A common downside of these two tools is that they are not open source [6], and not easily available in the particular case of XIRAF, and in the case of HANSKEN this is a paid service. They are also not optimized for any particular scenario as the one we consider in this work (child pornography).

In the following section, we will provide a historical overview of digital forensics in order to understand the past and current challenges, followed by a detailed overview of the major file system and what to expect when analyzing them.

When we identify a file as being a valid picture we need to label it, this can only be done using machine learning models. Thus we present a comparison of several options we can choose from and pick the best candidate to use in our architecture.

Finally, we need to make the results available to the end-user, thus we talk about indexing algorithms and how they can be applied to images.

2.1 Digital forensics

The advent of digital forensics has existed since personal computers were invented, almost forty years ago. Initially, the forensics techniques were developed mainly for data recovery. Examples of these early techniques can be found in data recovery techniques they executed for 70 hours to restore a copy of a highly fragmented database mistakenly removed by a careless administrator[7].

By the mid-1980s, utility programs that could perform tasks such as data recovery started being advertised, including "Unformat, Undelete and Diagnose & Remedy"[7]. In those days, there were no automated tools for those purposes. Data forensics has emerged from a collaboration between professionals and law enforcement on an ad hoc and case-by-case basis.

Around this time, the FBI started in 1984 the “Magnetic Media Program” to find child pornography offenders. Unfortunately, it was only used in three cases[8].

The “Golden Age” of digital forensics was from 1999 to 2007. People saw it as having a magical mirror that could retrieve an artifact of the past and started to prove valuable to law enforcement agencies in criminal cases where emails and instant messages were involved. At the time, it was beginning to be possible to perform network and memory recovery even months after the fact[9].

The popularity of these techniques started spreading, also appearing in TV Series, nicknamed the “CSI EFFECT”[10].

Nowadays, Digital Forensics is facing a crisis, that is related to factors that make these activities increasingly complex and time-consuming. The factors include:

- The growing size of storage devices
- Increasing embedded flash storage
- Encryption
- Popularity of cloud services
- Legal aspects related to digital evidence.

The standardization of secure encryption into operating systems has created a significant challenge for forensic examiners, meaning that it is becoming increasingly impossible to recover digital pieces of evidence without a key or passphrase[11]. This is present in hard drives and network traffic alike.

In summary, the traditional digital process is illustrated in Figure 2.1.

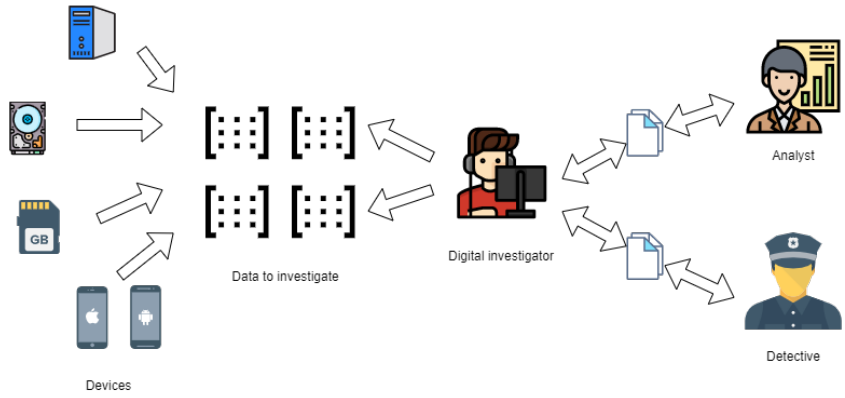

Figure 2.1: Traditional Digital Forensics process

2.2 Architecture

Since the closest work to our goal in this thesis are the XIRAF and HANSKEN systems, we will use their proposed architectures for our soluction.

The XIRAF architecture describes three components illustrated in Figure 2.2, which are: the feature extraction, tool repository, and the storage system[2].

From the XIRAF perspective, the starting point for the operation of the system happens when one or more pieces of digital evidence are fed into the system[2]. This content is produced in the form of a Binary large object (BLOB) to be analyzed. A BLOB is a group of binary data stored as a single entity in a system capable of storing and retrieving them, these include images, audio, and other formats.

The feature extraction manager takes the input and tries to extract as many features as possible, and it achieves this by running the tools present in the tool repository and parsing output.

The tool repository contains a collection of tools for feature extraction, they are called and afterwards, the outputs are merged in the form of a XML format that is stored in the storage subsystem.

The storage subsystem stores the output as an XML tree that annotates those BLOBs. BLOBs are managed by the BLOB manager who is responsible for giving access to the original BLOB input data.

Both tools and user queries require some level of access to the BLOB. To make this possible, the XML annotations are stored in MonetDB [12], a high-performance database system that provides several front-ends, including an XQuery front-end. The XIRAF architecture and modules are illustrated in Figure 2.2, there are several ideas we can take from it to serve as a base for our work. The first is that we similarly will need a Tool

repository to invoke and conduct the feature extraction process, an interface to collect the results, and a database to store and make them available.

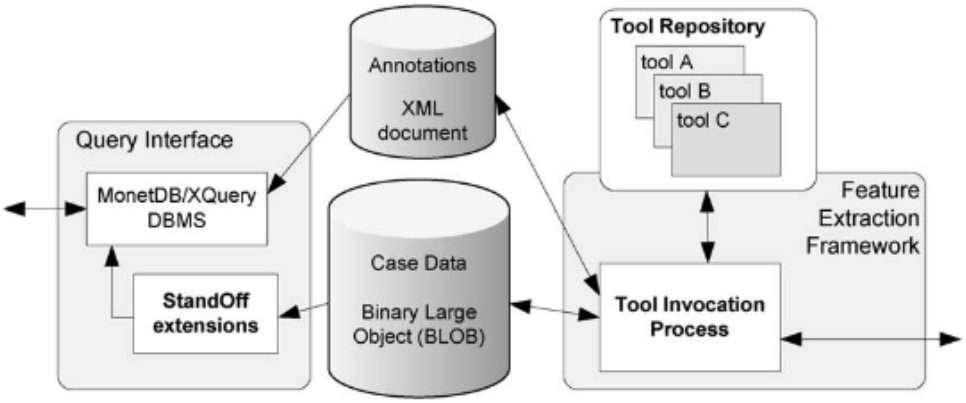

Figure 2.2: XIRAF framework architecture taken from [2]

Looking at the HANSKEN architecture [5], we can distinguish the RPC framework, the Gatekeeper Service, the Lobby Service, the Orchestration service, the Project Service, the Keystore, the Data service, the Trace Service, and a User interface.

A simplified view of this architecture is provided in Figure 2.3.

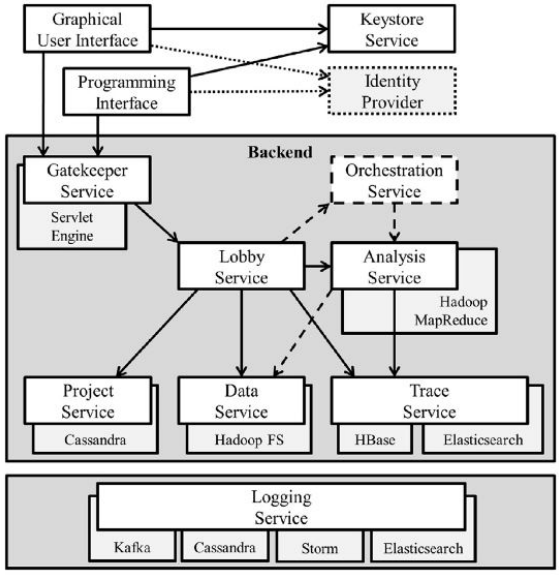

Figure 2.3: HANSKEN Modular view [5]

The RPC Frameworks is one of the most critical components of the HANSKEN architecture. It describes the communication between the different modules and the shared functionality of the modules. It includes mechanisms for initial set up and dynamic failover, where failover hosts are registered previously in Zookeeper [13].

One of the core features of the above architecture is that it reports its findings to the outside world, it deals with the authentication over a RESTful web service. Since our

work will be executed with no external connection this module will not be described any further.

The Lobby Service funnels the user calls to their intended module, and it keeps track of the routes that the functions should follow and assures that the requests are executed in the appropriate order.

The Orchestration Service, in theory, would make business decisions. However, the paper does not provide any further details and states that there is no actual implementation of this module, and is only showed as a conceptual piece of the architecture [5].

The Project Service is used for storing information related to images, HANSKEN agglomerates this data into cases that form a project. The unique identifier of the photos is translated into a name that makes sense to investigators. This is done on top of a key-value store. The described implementation uses Kryo[14] and stores the images and project details in the Cassandra datastore[15].

The images are retrieved from the Data Service, their implementation takes a hybrid approach model, being possible to run the Data service as a standalone, like any other module in HANSKEN, or embedded in another module for performance reasons.

The Keystore Service is responsible for storing the encrypted obfuscated keys and encrypted shared secrets[5]. The Extraction Service analyses the data and extracts from its traces, it does this by applying the tools from forensics libraries and sending the results to Trace Service.

The Trace Service is responsible for storing and retrieving traces. A trace is a metadata, a full keyword index, and a link to data of the trace. The authors used ElasticSearch[16] and stored the traces using HBase[17].

Each trace has one or more types. These properties are generated based on another set of properties, e.g., a unique identifier, type, and name, combined with additional properties for the types of a trace, e.g., modification date, e-mail subject, or phone number.

The ultimate goal of HANSKEN is to be an open service available to any law enforcement agency that needs it. This service would let their clients invoke Hansken through their Application Programming Interface (API) and showcase the results as required by the clients [5].

Taking into account the two previously discussed systems, there are a few components that our architecture will also need to have.

These include a Disk image extractor, Feature Extraction, a way to store files and metadata associated with each of these files, and a GUI interface.

The Disk image extractor is needed to obtain the files present in a raw image obtained from the devices of suspects. Since there are multiple file systems, we will need a component that can operate on top of any particular file system.

The Feature Extraction is present in both architectures. On XIRAF, it is the tool repository, and on Hansken the Extraction Service, the core ideas are similar. An automated component that can run some type of metadata extractor from the files that are present on the disk image.

Another component that would be of interest is one that can store individual files which providing privacy and security. A common solution would be to use key-value stores such as HANSKEN and making it searchable.

In HANSKEN [5], there are no details of how the progressive web app is implemented, only some screenshots are shown. In our work, we will use a JavaScript framework to speed up development time, improve code stability and ease of use. Figure 2.4 summarizes our initial proposal.

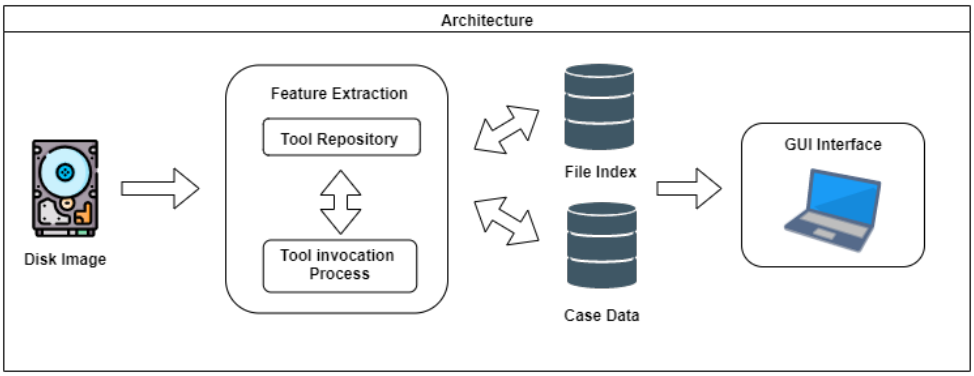

Figure 2.4: Simplified initial Architecture

2.2.1 Memory

The RAM holds the programs that the computer is currently executing. Furthermore, it is common for a program and data manipulated by other programs to be unencrypted in contrast to when they are in the hard drive, this gives forensic investigators a unique insight into what the suspect was working on, the system past state, and even recover encryption keys for the content of the disk or the Hypertext Transfer Protocol (HTTP) traffic[18]. There are even several frameworks capable of automatically reassembling and redrawing all apps Graphical user interface (GUI)s from a multitude of data elements present in an Android device memory. One example is GUITAR[19] which can generate 80 - 95 % accurate screenshots of what the GUI had shown previously.

Moreover, the data in memory is volatile, meaning it disappears when the power is switched off. Memory is simply a linear supply of pages, addressed as an offset from the first page [18].

When the operating system needs, it can swap the memory page and store it on disk. This is done when free memory is in low quantity, and another process needs additional memory space. This process is called paging. On windows, they are stored on pagefile.sys, making it a forensics artifact of value because it will survive a reboot[18].

Currently, modern computers use a technique called virtual memory, this allows operating systems to give more memory than the physical limits of a computer. They achieve

this by writing to the hard drive the exceeding amount. Thus allowing the process to allocate more memory than is physically available, since we are writing to disk the content there have been ways of recovering what was in the systems in a previous execution [18].

In the scope of this work, we do not specifically address memory forensics, with our focus being instead on hard drives and image processing.

2.2.2 Disk

Disk images are bit-streams that were extracted from physical media. They play an essential part in any forensic investigation. They can be extracted from computers and mobile devices. Nowadays they are widely used by digital forensics investigators for preserving activities, maintaining data integrity, and chain of custody[20] while enabling access to potentially valuable data.

A disk image is ultimately a sector-by-sector copy of the data. They can also be named "snapshot" and include all allocated files, file names, and other metadata information associated with the disk volume.

Upon creation, it is stored as a single file or set of files depending on the software that was used, the simple act of turning on the device or booting the operating system can result in data being changed and possibly destroy some files. This includes modifications to operational metadata and other aspects of the original data objects such as byte order, character encoding, file system information, MAC (modified, accessed, created/changed), permissions, and file sizes[20]. Due to this, programs that manipulate the image use low-level input-output operations and without any intervention of the host operating systems. Nowadays there are available both free and commercial solutions such as FTK Imager or by the Macintosh Disk Utility.

2.2.3 Discussion

Excluding mobile drives and portable disks, the two central locations on a computer that contains data are the memory and the hard drive.

Regarding memory, it is not common for forensics investigators to obtain memory dumps from computers, it is mostly applied to smartphones. Although there exist some forensics methods for iPhones, there are more for Android. This is due to the fact the source code is widely available, and some examples have been discussed on [19], which shows that it is possible to piece together Apps GUIs and VCR. The work presented in [21] proposes a framework capable of automatically recovering pieces of photographic evidence produced by applications.

Our work will only focus on the contents found in hard disks. However, as we stated earlier, they may be encrypted [11]. There are several ways that the encryption may be broken depending on the scheme being used. However, in this work, we do not undertake this challenge and assume that the disk images provided to our system have no encrypted content.

2.3 File system

A file system is a piece of software that manages how the information is stored and retrieved. It is made of structures that make up the common content of a partition.

A hard drive contains a partition, which houses a file system that contains the data. There are some exceptions to this, however, they are not relevant for this work (i.e. swap space)[22].

The advantage of separating the data into smaller pieces and identifying each piece uniquely with a name is that it provides isolation and identity, each data block is a named file. Since it manages these structures like a system, it is named "file system".

There exits thousands of file systems, in this work we will focus on the ones that a forensic investigator are more likely to encounter, these are: NTFS (Section 2.3.1), HFS Plus (Section 2.3.2), Ext3 (Section 2.3.3) , Ext4 (Section 2.3.4), and FUSE (Section 2.3.6).

A file system can be used in several types of media, such as hard drives, removable thumb drives, optical media, and so on. The future trend for hard drive is to use Solid State Drive (SSD). This saves energy and has faster response times. The drawback is that they are more expensive.

2.3.1 New Technologies File System

The New Technologies File System (NTFS) was developed by Microsoft and is the default file system in Windows, meaning this is the most common file system for forensics.

When developing NTFS, Microsoft designed it for reliability, security, and support for large storage devices, currently in the terabyte's zone. This is a scalable design because the internal structure will inevitably change during normal usage[22]. Microsoft never officially published an on-disk layout for NTFS. However high-level descriptions of components and low-level details can be found from their party like the Linux Foundation and other research groups that describe their perception of the on-disk structures [23].

The core concept behind NTFS is that important data is allocated to files, including the administrative data, typically hidden on other file systems. Therefore, NTFS does not have a specific layout. The entire system is considered a data area. Any sector can be allocated to a file, an exception for this is the first sectors of the volume which contains the boot sector and boot code. [22].

In the next subsections, we will detail key aspects of existing file systems, which are presented mostly for comprehension and to identify aspects that we might need to be careful when supporting automatic inspection of disk images from these file systems.

2.3.1.1 Master File Table

The Master File Table (MFT) holds the information about every file and directories, each file or directory must have at least one entry in this table. Each entry is 1 KB in size, the

first 42 bytes have a reserved purpose and the remaining are used for attributes. This structure is illustrated in Figure 2.5.

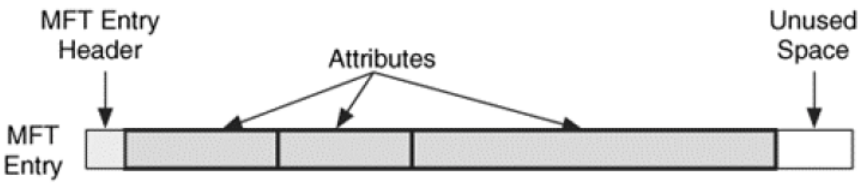

Figure 2.5: MFT entry [22]

This table is also a file and has a record to itself in the MFT which is named $MFT. The MFT location is in the boot sector, which is found in the first sector of the file system. This is illustrated in Figure 2.6.

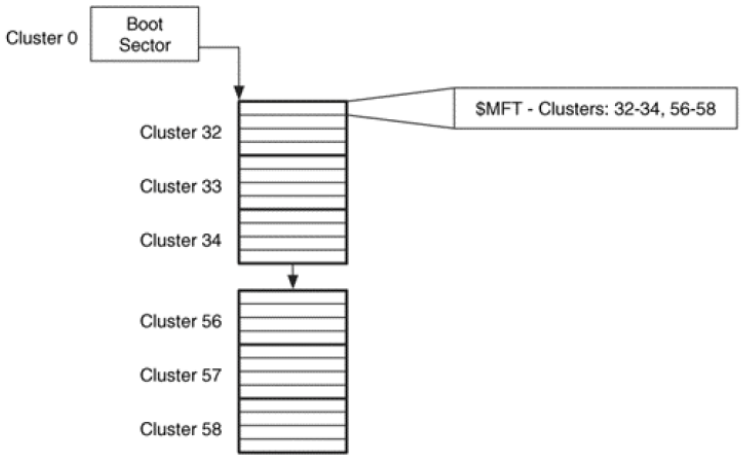

Figure 2.6: MFT Boot Sector [22]

2.3.1.2 Entry Addresses

The entries in the MFT are sequentially addressed using a 48bit value, and the first entry corresponds to value 0. Microsoft calculates the maximum MFT address by dividing the size of the $MFT by the size of each entry.

There is also present a 16-bit sequence number that is incremented when the entry is allocated, combining the entry and sequence number we obtain a 64-bit file reference address as shown in Figure 2.7.

Figure 2.7: Example of the MFT entry [22]

This makes it easier to determine when a file system is in a corrupt state. An example of a scenario that can lead to this is a crash while data structures for a file are being allocated. We can use this to recover deleted content, and if we find an unallocated data structure with a file reference, we can determine if the MFT entry has been reallocated since its creation [22].

2.3.1.3 File System Metadata

Since everything in the volume has to be a file, there must exist files somewhere in the file system to store the system administrative data. These are called metadata files by Microsoft.

The first 16 MFT entries are reserved for file system metadata files, the unused entries contain only basic and generic information. These can be found in the root directory, although they are typically hidden from common users.

2.3.2 Hierarchical File System Plus

The HFS Plus is the file system used by Apple computers running Mac OS.

Initially introduced in Mac OS 8.1 [24], it shares an architecture very similar to its predecessor Hierarchical File System (HFS) although naturally bringing several changes.

The more significant according to apple development portal [25] are summarized in the following table:

Feature	HFS	HFS Plus	Impact
User visible name	Mac OS Standard	Mac OS Extended	None
Number of allocation blocks	16 bits worth	32 bits worth	Decrease in space usage and files on large volumes
Long file names	31 characters	255 characters	User benefit and improved cross-platform compatibility
File name encoding	MacRoman	Unicode	Allows for international-friendly file names, including mixed script names
File/folder attributes	Support for fixed size attributes	Allows for future meta-data extensions	Future system can use metadata to improve Finder experience
catalog node size	512 bytes	4 KB	Maintains efficiency in the face of the other changes
Maximum file size	2^{31} bytes	2^{63} bytes	User benefit

Table 2.1: HFS Plus new features [25]

These features are available through the Operating system (OS) interface, since Mac OS 9.0. The key advantages they bring are an efficient use of disk space, international-friendly names, future support for named forks (see Section 2.3.2.3), and support for

booting non-Mac OS.

2.3.2.1 Use of Disk Space

HFS divides the space on the volume in equal-sized pieces called allocation blocks. It uses 16-bit fields to identify a particular allocation block [25].

The use of 32-bits values for the allocation of blocks allows for 2^{32} (4,294,967,296) allocations within a volume. This means that the file system allocates more blocks resulting in a smaller allocation block size, which means less average wasted space. It also allows the file system to support more files. These changes are particularly important in volumes containing small files.

2.3.2.2 Core Concepts

HFS Plus can best be understood by discussing the core structures that manage and organize the data on the volume. These are the volume header (Section 2.3.2.4), the catalog file (Section 2.3.2.5), the attributes file (Section 2.3.2.6), the allocation file bitmap (Section 2.3.2.7), and the startup file (Section 2.3.2.8).

In the following sections we describe these structures, since understanding them can bring benefits to this work.

2.3.2.3 Fork (file system)

A Fork can have several meanings in computer science, when we refer to them in this work, it is in the context of file systems. In a file system, a fork is additional data associated with an object within the file system itself. This enables the file system to have multiple sets of data for the content. This is not a common feature. However HFS and HFS Plus have native support for this[24, 25].

Originally it was designed to store non-compiled data that would help to render the systems GUI. This was revealed to be very useful and so other usages started to appear, such as document processing, storing parts of a compiled resource separately, and so on. Since HFS Plus the file system can have an arbitrary number of forks (See Table 2.1).

2.3.2.4 Volume header

Every HFS Plus volume contains a 1024 byte header at the start of the volume. It includes information on the whole volume and the location of other structures. There is also a copy of this table stored in the last 1024 bytes at the end of the volume. It is intended exclusively for use when disk repair is needed. The first 1024 bytes and the last 512 bytes are reserved[25], making the allocating block 1536 bytes in total, the HFSPlusVolume Header describes it

There are several key points to take away from the previous table, the first regarding hidden files, is that we can detect their presence, chronology place the volume, and

determine if it is often, or rarely accessed. Additionally, metadata can be extracted from the finderInfo array. This information can provide valuable insights, such as which files the suspect frequently opened, which operations are more frequent possibility to flag them for manual inspection in a later stage of the investigation.

2.3.2.5 Catalog file

HFS Plus uses a catalog file to store information about the hierarchy of the files and folders on the volume. This catalog file uses a B-Tree based implementation.

A B-tree consists of a header node, index nodes, leaf nodes, and map nodes, the location of the header node is obtained from the catalog file's header node. From this node, the operating system can search the tree for keys.

Every file or folder is assigned a unique catalog node ID know as CNID. Typically the CNIDs are sequentially allocated, starting at kHFSFirstUserCatalogNodeID, more recent implementations now allow for CNID values to wrap around enabling reuse, this is where the kHFSCatalogNodeIDsReusedBit is used to set nextCatalogID.

Since the catalog file extends a B-tree[26] file implementation it inherits the basic structures and definitions, it only changes two things: the format of the key used for the index and leaf nodes and the format of the leaf node data records.

2.3.2.6 Attributes file

The HFS Plus implementation reserves named forks for use in the future, an attribute file, is also a B-tree[26]. An attribute file is a particular file present in the HFSPlusForkData records the volume header, with no record in the catalog file, with variable length and three data record types.

A volume can have no attributes files if its extended attribute files contain zero allocation blocks[25]. The leaf nodes of an attribute contain the attributes, they can be Fork data which are used for attributes with large data, or extension attributes which augments fork data attribute allowing the fork to have more than eight extents.

These records are known as recordType field, which describes the types of attributes that are present in the data record, the values can be a Folder Record, File records, folder thread record, or file thread record. A thread record is used for linking a B-tree file to a CNID.

2.3.2.7 Allocation file

The allocation file is used to track whether each allocation block in a volume is currently allocated to a structure or not. The implementation is a bitmap. This bitmap contains one bit for each allocation block present in the volume. In the advent of a bit being set, the corresponding allocated blocks are being used somewhere in the file system structure. The opposite means it's available for allocation.

HFS provides several advantages mainly a simplified design, making it extendable and offering the possibility of being shrunk. The result is an implementation that quickly creates a disk image suitable for volumes of varying sizes.

A 32-bit number determines the size of allocation blocks, the size of the allocation file can be up to 512 MB in size, HFS's only supported 8 KB.

2.3.2.8 Startup file

The startup file is specifically designed to hold the information needed to boot a device that does not have HFS Plus built-in ROM support. The boot loader can find this file without knowing the volume status (B-trees, catalog file, and so on). Instead, the volume header contains the location of the first eight extents of the startup file [25].

2.3.3 Ext3 Concepts and Analysis

The Ext3 file systems were widely used in many Linux distributions before being replaced with its successor Ext4.

Ext3 added file system journaling, but the basic structure remains the same as in its predecessor Ext2. The Ext family of file systems is based on the UNIX File System know as UFS. Ext removed many of the components of UFS because they were no longer needed, which made the Ext file system easier to understand and be explained[22].

In Ext, there are two primary data structures responsible for storing data in the file system, they are known as the superblock and the group descriptor. The superblock can be located at the beginning of the file system and contains the size and configuration information.

As previously stated, the file system can be divided into block groups, each of which contains a group descriptor data structure that describes the layout of the group. These group descriptors are located in a table called the descriptor table, located after the superblock. In the advent of the primary copies of the superblock being damaged more copies can be found throughout the file system in pre-determined locations.

2.3.3.1 Superblock

The Ext superblock is located at 1024 bytes from the starting point of the file system and occupies 1024 bytes. In normal conditions, most of these bytes are not used[22]. This structure is only used for configuration values and contains no boot code. Typically the first blocks of each group contains a copy of the superblock.

The information contained within it is quite simple, such as the block size, the total number of blocks, the number of blocks per block group, the number of reserved blocks before the first block group. In some instances, it also contains the total number of inodes per block group, and some nonessential data such as the volume name, last write time, last mount time, and the path in the file system where it was last mounted.

Some values can inform the operating system of the file system being clean or if some consistency check needs to be executed. The superblock can also store the total number of free inodes and blocks currently allocated.

To determine the file system layout, the first block size number is used, and the number of blocks is employed to calculate the file system size. If the result is less than the volume size, this means there could be hidden data, typically called volume slack. The first block group is located in a reserved area. Figure 2.8 illustrates this structure and also shows that the groups don't necessarily need to have the same size.

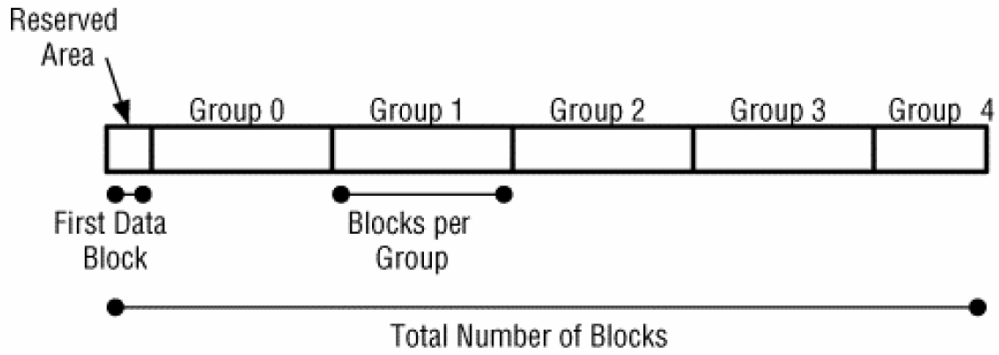

Figure 2.8: Layout of Ext [22]

In Linux, the volume label used in the superblock can identify the file system; an example of this is in Unix where /dev/hda5 refers to its device name. Another way is by using the volume label and the system configuration files. For example /etc/fstab in Linux list the file system that should be mounted and possibly refer to the /dev/hda4 device as "LABEL=rootfs" if the volume label is rootfs [22].

2.3.3.2 Block Group Descriptor Tables

The group descriptor table is what comes after the superblock. It contains a group descriptor data structure for every block group. Backup of the table can exist in each block group unless the sparse superblock feature is enabled. Additionally, the block groups contain administrative data, such as superblocks, group descriptor tables, inode tables, inode bitmaps, and block bitmaps[22]. A simplified layout of the block group is depicted in Figure 2.9.

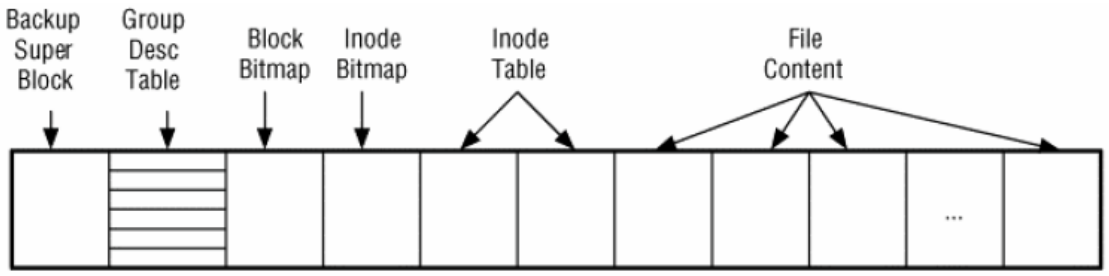

Figure 2.9: Layout of a sample block group [22]

The block bitmap manages the status of the allocation block for each group, and the starting block address is present in the group descriptor. Its size in bytes can be obtained by dividing the number of blocks in the group by eight[22]. Linux, when creating a file system, defines the number of blocks per group as being equal to the number of bits in a block. Thus, the bitmap will take precisely one bock.

The inode bitmap is responsible for keeping the state of the inodes in the group and its starting block address. Its size in bytes is the division of the number of inodes per group by eight. In practice, there are fewer inodes than blocks per group. The only exception is if the user when creating the file system changes these values. The starting block address for the inodes are in the group descriptor, its size is calculated by multiplying the number of inodes per group by the size of each inode, which is 128 bytes[22].

The group descriptor contains the number of free blocks and inodes in the block group. The superblock contains the total number of free blocks and inodes in all groups.

2.3.3.3 Boot Code

The boot code is not always present in every Ext file system. It only is valid if it contains an OS Kernel. All other non-boot file systems don't need this. In the cases where boot code is present, it will occupy 1024 bytes before the superblock, this means that before the first two sectors. The contents of the boot code are executed after it gains control from the boot code in the Master Boot Record (MBR) present in sector 0.

Additionally, the Ext file system knows which blocks have been allocated to the kernel and which are present in memory.

Looking at current Linux systems, many of them don't have boot code in a predetermined file system location in the file system. Instead, there is a boot loader in the MBR, and it knows in which blocks the kernel is located. Thus, the code in the MBR is enough to load the kernel.

2.3.4 Ext4

Ext4 is an enhancement on Ext3, currently, it is the default file system for Linux distributions, this makes it an important subject for forensics investigator.

Since Ubuntu 9.10, Fedora 11, and OpenSuse 11.2, Ext4 is the primary volume file system and boot partition as well. Its predecessor, named Ext3 was one of the most highly used file systems in the Linux community as well. Still, Ext3 suffered several limitations that Ext4 addressed, the main one being the ever-increasing size of storage devices[27].

Since 2002, the need for Ext4 has been evident [28] with multiple proposed extensions for Ext3 to provide new features for maintenance and evolution. It is backward compatible with Ext3.

A significant difference regarding Ext3 is on the organization of group blocks, the previous 128 MB block group limit resulted in a lower file system size, not ideal for modern drives [29]. This is a result of the fact that only a small number of group descriptions could be placed in the span of a single block group. The solution proposed for this problem is the meta-block group feature [28], where a single descriptor block can describe a group consisting of a series of blocks.

2.3.5 FAT Concepts and Analysis

The File Allocation Table (FAT) is probably one of the most simple file systems present in conventional operating systems. FAT was used on Microsoft DOS and Windows 9x operating systems, but since then was replaced with NTFS (Section 2.3.1). It is supported by all Windows and most Unix operating systems and from a digital forensics point of view, it will be encountered by investigators for years to come. The reason for this is frequently found in the compact flashcard. These can range from digital cameras to USB thumb drives, which many times use this file system, which makes it an important file system to discuss.

One of the main reasons we stated earlier that the FAT file system is simple is due to its small number of data structures. However, this has made it necessary, over the years, to modify those data structures, to provide new features.

There are two main data structures in FAT that address multiple purposes and belong to various categories of the model. Every file and directory is allocated as a data structure, called a directory entry. This contains the file name, size, the starting address of the file content, and other metadata. Files and directory content are kept in data units, which are named clusters, they can be allocated to more than one cluster. The relation between these data structures is shown in Figure 2.10

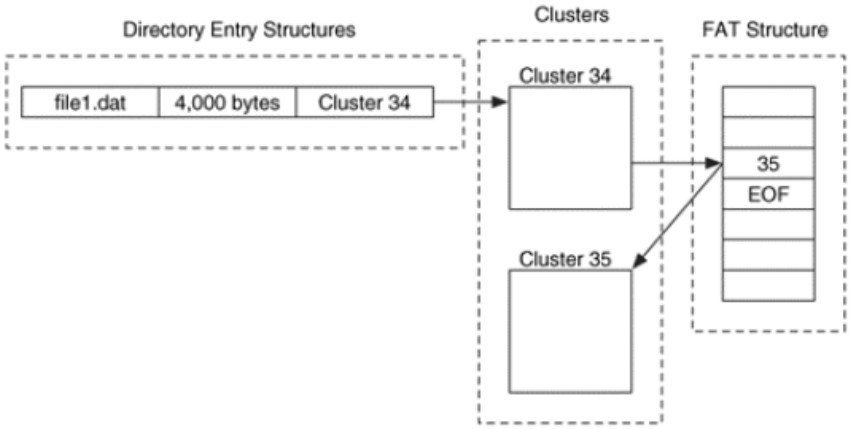

Figure 2.10: Relationship of FAT structure [22]

There are three different versions of FAT: FAT12, FAT16, and FAT32. The significant difference among these versions is the size of the entries in the FAT structure. The layout of a FAT file system is comprised of three physical sections, which can be seen in Figure 2.11.

The Reserved area is the first section and includes data in the file system category. Typically in FAT12 and FAT16, there is only one sector. It can be, however, altered in the boot sector. The second section is the FAT area, and it houses the primary and backup FAT structures. This structure is present immediately after the reserved area. The third section is named the data area and stores the cluster that will be allocated for the file or the directory.

Figure 2.11: Physical layout of a FAT file system [22]

A FAT 32 file system boot sector contains additional information, that includes the sector address of a backup copy of the boot sector and a major and minor version number. FAT 32 also has a FSINFO data structure that contains information about the location of the next available cluster and the total amount of free clusters [30]. This data is not guaranteed to be accurate and is only a guide for the operating system. The next subsection will describe these aspects in more detail.

2.3.5.1 Essential Boot Sector Data

One of the first concepts that are relevant to understand the FAT file system is the location of the three previously discussed layout areas.

The data areas in FAT are organized into clusters and described in the boot sector. There are slight differences in the data area from FAT12 and FAT16 to FAT32. In FAT12 and FAT16, the beginning of the data area is solely used for the root directory, whereas, in FAT32, the root directory can be anywhere in the data area. However, it is rare for it not to be in the beginning similar to previous versions of FAT. Dynamic size and location of the root directory allows FAT32 to avoid bad sectors and enable the directory to grow as large as needed. In FAT12/16, the directory has fixed sizes that come from the boot sector. Figure 2.12 shows a detailed comparison of FAT12/16 and the FAT32 file system.

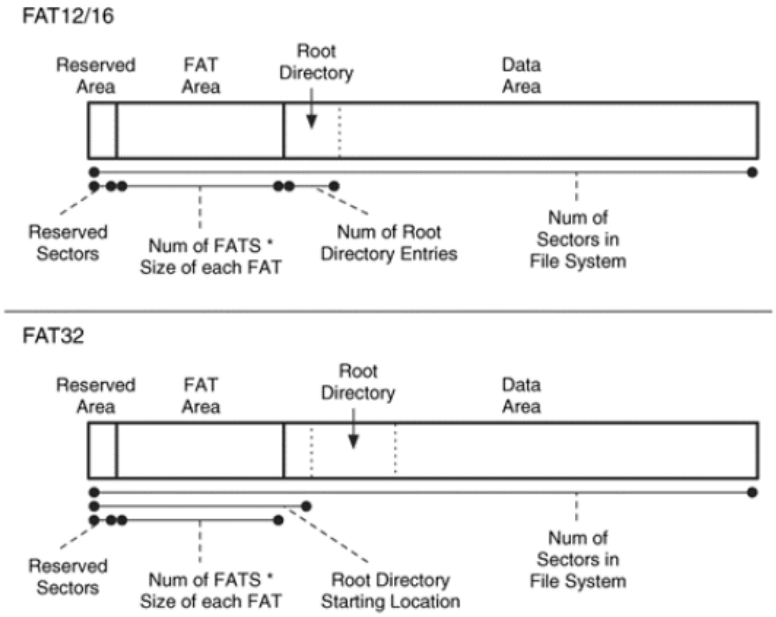

Figure 2.12: FAT file system layout [22]

2.3.5.2 Non-essential Boot Sector Data

In addition to the layout information, the boot sector contains non-essential values. These non-essential values have these names because they are not needed for the file system to work normally. The purpose of these values is for convenience and, in some cases, may not even be correct. An example of such a value is a string called the OEM name. This corresponds to the tool that was used to create the file system. However, it's an optional value. Windows 95, as an example, sets it to "MSWIN4.0", windows 98 to "MSWIN4.1", Windows XP to "MSWIN5.0" and so on. Linux is also capable of using FAT, it set this value to "mkdosfs"[22].

FAT file systems have a 4-byte volume serial number that according to Microsoft specifications, is generated at creation time using the current time, although the operating system can decide any value it sees fit.

Additionally, there is an eight-character string, which can be "FAT12", "FAT16", "FAT32",

or "FAT". Almost every toolset sets a value for this string currently, but it is not a requirement for it to be updated. The only accurate way of determining the file system type is by calculation.

The last label is an eleven-character volume label string that can be specified by the user on creating the file system. It is saved in the root directory of the file system.

2.3.5.3 Boot Code

The boot code for a FAT file system is intertwined in the system data. This is the complete opposite of Unix file systems where it is completely isolated.

The first three bytes of the boot are jump instructions in assembly that make the CPU jump the configuration data to the rest of the boot code.

Having boot code in a FAT file system does not guarantee however, that the drive is bootable. In these cases, the code display a message to signal another disk is needed to boot the system. The boot code is called from the instructions present in the MBR, and this locates and loads the appropriate operating system file.

2.3.5.4 Content Category

The content category is where the data that comprises a file or directory is stored. FAT names its data units as clusters. A cluster is a group of consecutive sectors. These must be of the power of 2, such as 1, 2, 4, 8, 16, 32 or 64. According to Microsoft official specification[31], the maximum cluster size is 32KB. Every cluster has an address, and the address of the first cluster is 2. Meaning a cluster address can't be 0 or 1. Every cluster is located in the data area region of the file system, which is the last of the three areas[22].

2.3.6 File system in Userspace

FUSE is the most widely used userspace file system implementation[32]. As such many file systems were implemented using FUSE, this is because of its simple API and changeable internal components architecture, however it does suffer from performance issues these are due to a bottleneck regarding FUSE daemon[33].

FUSE consists has a part kernel and a part on user-level. The kernel part is implemented in the Linux kernel module, that when loaded, registers a fuse file system. A fuse file system behaves like a proxy for the user-level daemon.

A FUSE driver is registered with the path /dev/fuse block devices, and this device is the interface between daemons and the kernel. Generally, the daemon reads FUSE requests that arrive at /dev/fuse, processes them and afterward writes the response to the same path.

A simplified and modular view of FUSE architecture is illustrated in Figure 2.13.

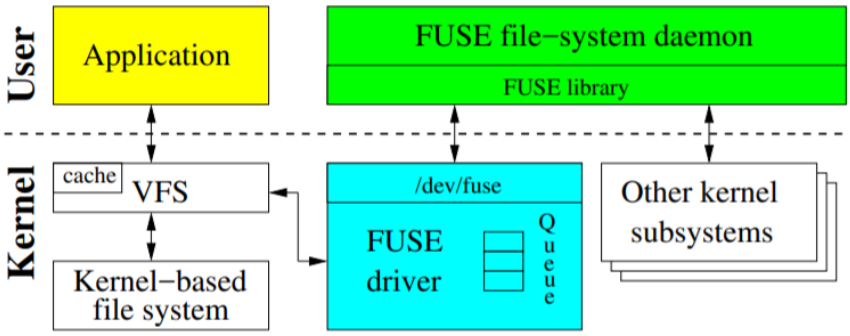

Figure 2.13: FUSE high-level architecture [34]

When an operation occurs over a mounted FUSE file system, the VFS is responsible for routing the transaction to the FUSE's kernel driver. This driver then allocates a structure for the request and places it in the queue. The process that started the operation is placed in a waiting state. Sometimes processing might require FUSE to reenter the kernel, which can generate a lot of overhead, which in turn makes FUSE slow. Recent work proposed a solution to make a FUSE implementation work without crossing the userspace [35]. Without this, previous implementations could also complete requests without communicating with the user-level FUSE daemon, but they had to be present in the kernel page cache.

2.3.7 Discussion

After presenting a brief overview of the various file system one thing is very clear, we need a tool capable of abstracting the file system and presenting the disk images and files within them, in a uniform manner capable of being fed in a pipeline efficiently.

The only tool capable of providing such level of abstraction is Sleuth Kit (TSK) and the Autopsy Forensic. They are both Unix-based tools released in early 2001 [22].

The Sleuth Kit is a bundle of over 20 command-line tools that are organized into groups[36]. Certain groups include volume tools, disk tools, file system tools, and searching tools. Additionally, the file system tools contains disk drive utilities, and drive volume utilities, that are capable of organizing the data into categories. Each of the tools can be broken into two parts, the first uniquely identifies its group, and the second identifies its function. An example is a file name category (starts with f) that lists (the ls) another example would be *istat* which manipulates the metadata category (the i) that displays statistics (the stat).

2.4 Machine Learning Techniques

To solve a problem on a computer, we need an algorithm. An algorithm is a simple sequence of instructions that operates over some input and generates some output[37]. Counterpointing, there are cases where we can not create or do not have an algorithm to

perform some complex task. An example is distinguishing a spam email from a legitimate one because this requires answering the question: is something spam? and what should be taken into account?

Machine Learning tries to solve a problem where we lack the in-depth knowledge necessary to build a clear algorithm. Taking the example above we can quickly compile hundreds of spamming emails and try to make the computer "learn" from that data.

The advantage nowadays is that computer hardware can store and process large amounts of data, which has made this technology increasingly viable.

Machine learning can be defined more formally as programming computers to optimize performance criteria using example data or past experience[37]. This work will use models that had parameters previously defined, and they were built by inferring them from training data or past experiences. The backbone of building mathematical models is the theory of statistics because it makes inferences from samples. Machine learning can be split into four basic kinds of problems, this are:

- Unsupervised learning
- Supervised learning
- Reinforcement learning
- Semi-supervised learning

Unsupervised learning is useful for finding unknown patterns in data sets without classifying previously, it can also go by the name of self-organization because of this. It models probability densities of given inputs [38].

Supervised learning is a function that maps input to an output based on example, input-output pairs[39]. Inferring a function from labeled training data from a set of examples[40].

Reinforcement learning, sometimes abbreviated to RL, is related to how an agent should take certain actions when placed in an environment to maximize some notion of rewards[41].

Finally, Semi-supervised learning combines some of the approaches explained earlier. It joins some labeled data with a large amount of unlabeled when the model is in training.

2.4.1 Not Safe For Work

NSFW are images that contain nudity and pornographic material. The detection of such material and the disciple of automatically identify corresponding imagery is also referred to as NSFW[42].

In recent years the reach in the field of NSFW detection has increased, this is due to governmental guidelines in some countries trying to limit the availability and distribution of such material. Also, companies denying access in their infrastructures to such materials requires automatic mechanisms for classification.

Some earlier approaches were based on skin detection and human body part detection using classifiers with hand-crafted feature[43, 44].

We found some implementations of this method using JavaScript and Python(nude.py).

Other techniques include using visual color words for detecting child pornography [45]. This approach aimed to help forensic investigators identify illicit images from a large image set.

However, looking at more recent methods, we see a trend in using Convolutional neural network (CNN), which has the advantage of not requiring any feature before training. CNN's can have learning algorithms that are supervised, semi-supervised, or unsupervised (see Section 2.4).

The primary difficulty in NSFW is distinguishing between beach photos of people in bikinis and trunks and real pornographic content. CNN overcame this by training using large datasets which allow the algorithm to take away specific features of pornographic content as if it was a human training.

2.4.2 Age Group Recognition

Age Group Recognition is the ability of an algorithm to classify faces into predefined age groups with an acceptable accuracy[46]. It is not a simple process even for humans[47], the Mean Absolute Error (MAE) of human age estimations from visual appearance are 4.7 year[48] .

Automatic age group classification can be made more accurate by using large databases such as identification systems. Some other major fields that take advantage of age group recognition are monitoring and bio-metrics [49], because of this they ensure younger children have no access to prohibited internet pages, and when used in vending machines can control the access to alcohol and tobacco for underage people[50].

The use of this technology is possible because of significant facial changes, such as craniofacial growth and skin deformation[46]. The most common age groups are 0-3, 4-7, 8-13, 14-22, 23-35, 36-47, 48-59 and 60+. Looking at the age group distribution, we observed that they are not evenly distributed this is because younger ages experience greater changes [48].

The challenges faced in age group classification are low images, facial expressions, and facial poses [51], additional factors like genetics, gender, and race also causes people to age differently. External factors such as facial hair, glasses and cosmetic surgery may hide the real age of a person [52], thus throwing off algorithms.

2.4.3 Discussion

Analyzing our architecture of Figure 2.4, it is apparent the need of a tool in the repository to identify child pornography.

Using a real child pornography dataset would provide a reliable and accurate result. However, it would be illegal and immoral for several reasons. Thus we need a new way to solve our problem.

Glancing at the problem, we need a way first to identify pornography material and afterward separate a child from an adult. The solution for this is first to use an NTFW algorithm and secondly age classification algorithms[42].

There are several NSFW detection algorithms, the most popular are based on CNN architectures (see Section 2.4), they are Yahoo, Clarifai, nude.py and I2V [53–56]. Initial approaches were based on skin detection and human body parts, and they used hand-crafted features, an implementation of this algorithm is nude.py [43, 44].

Yahoo released an open-source NSFW detection API freely available on GitHub in 2016.

Clarifai is a paid commercial solution from Clarifai Inc, which specializes in computer vision, which relies on CNNs in age group identifying. Another free NSFW detection is Illustration2Vec abbreviated to I2V, trained initially with Anime images [57]. It distinguishes from other NSFW solutions in identifying explicit pornographic actions and specific body parts. For this, it is not a useful feature and will not be taken into account. Additionally, it is normal for algorithms that focus on pure NSFW detection to achieve much more precise results[42].

Considering the most adequate threshold for each algorithm, these parameters need to be tested on a comprehensive dataset. Fortnightly [42] has already done this for us. They had limited resources and could not manually inspect millions of images. Nonetheless, they validated their results by extrapolating the measured on a simulated dataset of sufficient size. This is archived by generating a series of 100 sets, each comprising 1 million data points, each representing an image, in the 100 datasets 1000 images were NSFW, and the remaining 999000 were SFW or in statistics 0.1%. The results are demonstrated in Table 2.2, where r@k is the recall after k images are retrieved, Rank1 refers to the first relevant image. For our particular work, it's crucial to determine if there is child pornography content of a given disk image or not. Ideally, independent of the number of images present in that image.

Rearrangement strategy	Rank1	r@100	r@1000	r@5000
random	1463.00	0.0000	0.0000	0.0050
binary (Yahoo)	144.39	0.0009	0.0071	0.0397
ranked (Yahoo)	8.74	0.0109	0.0977	0.3851
binary (Clarifai)	169.41	0.0006	0.0294	0.0283
ranked (Clarifai)	32.83	0.0031	0.0608	0.1386
binary (I2V)	324.65	0.0003	0.0030	0.0142
ranked (I2V)	41.79	0.0024	0.0234	0.0918

Table 2.2: Average performance over NSFW dataset

These results here them use to define the threshold for each classifier.

Afterward, they took a dataset of 2,000 sample images and calculated the ROC curves for all five classifiers, Yahoo, Clarifai, I2V, nude.py, and a coin toss. A ROC curve or receiver operating characteristic is a technique used for selecting classifiers based on their performance [58]. It shows how well a model can distinguish between two things in the content safe for work and not safe for work. The AUC value varies from 0.0 to 1.0. We want a solution with the highest AUC possible. The results of their work are summarized in Table 2.3 where tpr means true positive rate and fpr false positive rate. Thus the best algorithm will have a high AUC, tpr and a low fpr.

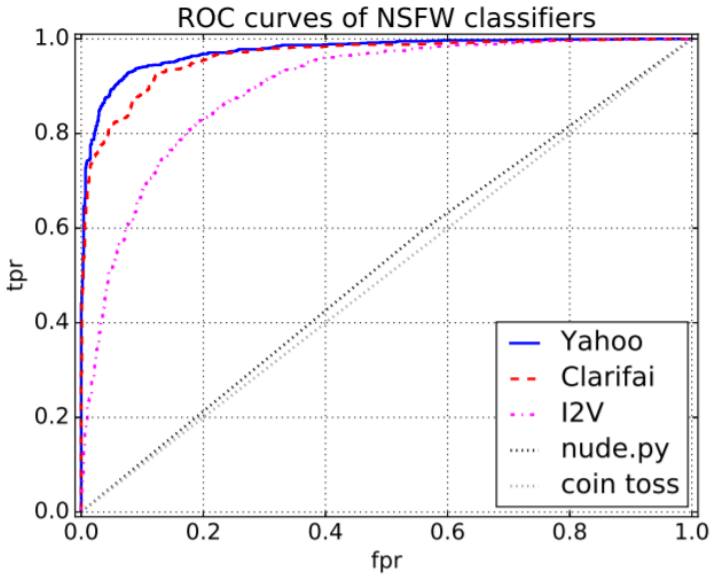

Figure 2.14: ROC curves of Yahoo, Clarifai and I2V compared to a random coin toss classifier on 2,000 samples test set. The x-axis represents the false positive rate (fpr) and the yaxis the true positive rate taken from [42]

The Receiver Operator Characteristic (ROC), shown on Figure: 2.14 corresponds to a graphical representation that summarizes the information contained in a set of confusion matrices for a binary classifier, in this case, if an image is or not NSFW. The ordinate axis represents the True Positive Rate (tpr), calculated by dividing the true positive,ie: tp, which in this case correspond to NSFW images, which the model has correctly classified as an NSFW image, by the sum of the true positives with the false negatives,(ie: fn, which in this case correspond to NSFW images that the model has classified as non NSFW.

$$tpr = \frac{tp}{tp + fn}$$

Thus, the fpr is the proportion of NSFW images that it correctly classified.

In the axis of the abscissas in this graphical representation, we can view the fpr that is calculated by dividing the False positives (FP), which correspond to non NSFW images and which were classified as NSFW and the sum of the false positives with the true

negative, which in this case corresponds to non NSFW images and, which the model correctly classified as non NSFW.

$$fpr = \frac{fp}{fp + tn}$$

The fpr informs about the proportion of NSFW images that were classified incorrectly. Note that in cases where tpr is equal to 1, this means that all the NSFW images were classified correctly. In cases where fpris equal to 1, means that all images that are not NSFW images are incorrectly classified as NSFW images.

Classifier	AUC	Threshold	tpr	fpr
Yahoo	0.975	0.384	0.928	0.076
Clarifai	0.963	0.682	0.922	0.121
I2V	0.896	0.090	0.826	0.190
nude.py	0.518	-	0.594	0.558
Coin toss	0.500	-	0.500	0.500

Table 2.3: Results of NSFW proposed classifiers[42]

Another important metric to consider when choosing the best algorithm is the Area Under the Curve (AUC), it is used to compare ROC charts generated from different classifiers. That is, the more the area under the curve approaches one, the better the classifier in question. Analyzing Table 2.3, Yahoo is the classifier with the best performance, since it presents an area under the curve higher than the other models and close to one.

Regarding age estimation this is a challenging subject, currently some of the state-of-the-art solutions use a form of CNN, some examples are VGG-FACE[59] CNN and Googles GoogLeNet[60] CNN. There result distributed by age are shown in Table 2.4

Label	Model by VGG-Face CNN	Model by GoogLeNet CNN
0 - 2	93.17	86.75
4 - 6	62.11	27.89
8 - 13	42.06	21.47
15 - 20	24.23	14.10
25 - 32	86.17	76.61
38 - 43	8.88	12.03
48 - 53	38.17	7.05
60+	60.70	34.63
Overall Accuracy	59.90	45.07

Table 2.4: Results VGG-FACE CNN and GoogLeNet CNN by Label taken from [59]

Looking at these results we can note that the VGG-Face CNN is pretty accurate until the 15 years old mark, 25 to 32, and for people with more than 60 years. Generally

speaking, the overall accuracy is 59.90% but taking into account our primary goal is to identify children under the age of 18 if we used the VGG-Face CNN the overall accuracy would be less than that. A better solution is therefore needed.

Luckily for us, some competitions challenge researchers to come up with solutions for this, one of these examples being the ChaLearn LAP 2015 challenge on apparent age estimation that had more than 115 registered teams.

The winner explained their model with great details and made it available for any interested party to use [61]. They used CNNs with the VGG-16 architecture [62] and pre-trained on ImageNet for image classification. However, ImageNet has a limited number of age annotations. To overcome this, they crawled 0.5 million images of celebrities from IMDB and Wikipedia pages, at the time of writing this thesis it is a public dataset for age prediction [61].

Deep EXpectation (DEX) of apparent age, starts by detecting the face in the image and then proceeds to extract the CNN predictions from an ensemble of 20 neural networks on the cropped face[61]. DEX, can be divided into five stages: input image, face detection, cropping the face, feature extraction, and finally prediction. Upon receiving the image, DEX detects a face using the Mathias detector [63] to crop the image the face needs to be aligned, the original image is rotated between -60° and 60° in five steps. The image chosen for the next step is the one with the strongest detection score. Afterward, the chosen picture is then fit in a 256×256 pixels square and analyzed by the CNN. An estimated age value is then outputted.

The CNN of DEX was optimized on the crawled dataset. Being well documented, freely available, and proved in a contest. Furthermore, comparing with the previous state-of-the-art CNN, DEX can predict to an accuracy of 96.6%, making it a perfect piece for our architecture.

2.5 Search engine indexing of pictures

The reason to index a large quantity of data is for later optimizing the speed and performance when finding relevant documents for a search query. Without the presence of one beforehand, the program would have to look up every document. This might not be seen as a problem if we have a low amount of documents to search, but issues start to arise as the number of documents increases. The difference in response time for some cases could be from some milliseconds to several hours.

Several techniques already exist, such as the Suffix tree, Inverted index, and N-gram index. These are designed taking into account the following factors:

- **Merge factors** - When data is placed in the index, how are the words added to the index, and is it possible to run such an operation asynchronously;
- **Storage techniques** - How is the index stored and is it possible for the data to be compressed or even filtered;

- **Index size** - How much computer storage the index requires to operate;
- **Lookup speed** - How much time does it take to find a specific word;
- **Maintenance** - How costly it is to maintain the index over time;
- **Fault tolerance** - How resilient the index is to corruption, bad data, and data loss.

2.5.1 Suffix tree

Since they were introduced in [64], suffix trees have been one of the methods of choice for text indexing, because they are easy to implement and provide many important features.

It can be described as a compressed tree built by all file suffixes using their keys and positions in the text as their values.

They can be implemented in several ways that affect performance.

2.5.2 Inverted index

An inverted index is similar to a database that stores the mapping from content to files.

Their primary focus is supporting fast searching and the main drawback of using an inverted index is the additional processing that is required upon insertion of a new document to the database[65].

The internal structure contains a *postings list* for every file present in the index. This postings list contains for each file a unique document identifier know as dockIds, which for increased retrieval speed are typically stored incrementally. These lists can quickly be accessed by consulting the *lexicon*, which is a dictionary that serves as a lookup table for each term that exits in the list and points directly to it[66]. It's prevalent among search engines, and several architectures use it, such as ADABAS, DATACOM, and Model 204[67–69].

2.5.3 N-gram index

N-Grams or n-word phrases use a tokenized index. The text is broken down word by word when they encounter one of a list of specified characters, then an n-gram of each word is emitted, which corresponds to the specified length.

The implementations are similar to a sliding window that moves over the word, this, in turn, produces a continuous sequence of characters that are in a specified length[70].

2.5.4 Image indexing

The previously explained indexes focus heavily on text indexing. However, some of these can also be applied to images. This is the case of inverted indexes, the idea behind it is to add a field to store the image. In Elasticsearch, this is known as the field key and is used for Binary datatype in the form of Base64 encoded string.

A Base64 schema is a group of encoders that convert binary to text using ASCII characters, each digit represents 6 bits of data.

An additional field like the properties mapping can serve to store the location on the image that the picture was extracted from and other metadata that we find relevant during the development phase.

2.5.5 Discussion

We need software that is optimized for distributed document indexing. A viable solution is ElasticSearch. ElasticSearch is a document-oriented database that was built with speed and performance in mind, it is specially designed to store, manage and straightforwardly retrieve documents.

On storing documents, ElasticSearch converts them to JSON format making them queryable and retrievable. Additionally, schema-less, means that we do not need to provide a prior scheme before using it, indexes are automatically generated and fine-tuned for type guessing and high precision. We can access them through its REST API.

Under the hood Elasticsearch is a collection of clusters, these are servers that save and give federated indexing and search capabilities. Behind the implementation there is an inverted index implementation (Section 2.5.2), this is also why it takes some time for a file to become searchable.

Its REST API is very well documented, and examples are abundant on the internet, this is an additional reason for why we chose it.

2.6 Summary

In this chapter we have discussed the current leading state-of-the-art architectures that exist today that can automate the processing of digital images, being the most know XIRAF and HANSKEN. XIRAF executes the tool in its repository and wraps its output in the form of a XML file for later reporting. HANSKEN was developed by the Netherlands police and builds upon XIRAF, transforming its standalone architecture into a cloud solution that can be used internally by a restricted amount of government agencies.

Furthermore, we showcased the evolution of digital forensics and how the decrease in storage cost has created new challenges that investigators face every day. Afterward, we presented a detailed overview of the main file system and a simplified view of their inner working.

Additionally, we discussed how we could identify a suspicious picture using machine learning techniques, more specifically using two CNNs. Afterward, we studied the comparison of available state-of-the-art models and their performance, from this we choose the Yahoo NSFW CNN to identify if a picture contains nudity and DEX to estimate the age of a person. This is due to their excellent performance, free of use and availability.

Finally, after processing an image, the metadata associated needs to be saved and searchable thus we presented some techniques like suffix tree, inverted index, and N-grams.

In the following chapter, we will present the requirements that our solution guarantees, how it can correctly identify suspicious files, and a high-level representation of each module.

Chapter 3

Architecture and requirement

> If I had eight hours to chop down a tree, I'd spend the first six of them sharpening my axe.
>
> *Abraham Lincoln*

In this chapter, we present the main modules of the project that we developed in the past months.

The work can be divided into 7 phases, the ability to correctly identify a suspicious image, make the results queryable, preserve the integrity, secure the transport of the data, prevent tampering and provide an easy-to-use interface.

We start, in the next subsection, by providing a high-level overview of the main challenges and requirements of our solution, the principles that are followed to solve each of them, and how they lay the path for the implementation and satisfaction of each of them in the following chapter.

3.1 Overview

In this chapter, we describe the architecture of our automated extraction tools for forensics investigation that focuses heavily on child pornography content. However, our architecture is extensible, and it can contain more tools in the tools repository, the work presented in [2] takes the same approach. An example would be extraction from documents such as PDF, Word, PowerPoint, and so on, for instance, to identify fiscal fraud.

Our main objective is to build a proof of concept to demonstrate that global forensics investigators can have access to systems that integrate several tools to speed up their work.

3.2 Design principles

The system needs to be designed as having a client and server components, for this, we need to design an architecture with 8 concerns, these are confidentially, access control, authentication, integrity, transparency, future proof, data retention and reliability.

Firstly, we considered a system that will operate on a home computer and should provide the results in a timely fashion but also be able to be used in a cloud solution. This is because in some cases the user will be on site and unable to run the solution anywhere else. However using a CNN is not a lightweight task and as the amount of data increases a slowdown is inevitable on a personal computer, hence such computational power should be delegated to a centralized and more powerful infrastructure.

Additionally, the more standards we follow the better the final code will be and easier to maintain and develop while making sure we do not generate technical debt.

In the following subsection, we dwell further into each concern in greater detail.

3.3 Design considerations

In our solution, we need to start by finding a way to make a solution that can run on a Windows machine and on a cloud provider. Therefore we choose to develop the final build for a container based solution.

Consequently, when the user needs to run the solution we can run it in any modern operating system where Docker can be installed. Docker will simplify the management of all the dependency's, code and models thus making the configuration process easy and straightforward. Simultaneously, Docker containers can package the solution and easily be deployed to the cloud.

Therefore, the final delivery for this work is a container image that can be executed on a personal computer and server anywhere in the world.

3.4 Confidentiality

When dealing with sensitive data we need to guaranty confidentiality, this means the data stored cannot be read by unauthorized principals. The classic solution for assuring confidentiality on stored data is by using symmetric encryption.

Symmetric encryption can also be called single-key encryption, it was been used since Julius Caesar, and the second world war and is still relevant today.

The algorithm starts by receiving as an input a plaintext and a key and applying an encryption algorithm to transforms and substitute the plaintext based on the key. The result of this operation is called ciphertext, this can be viewed as an unreadable version of the input. Two different keys will produce two different results. Furthermore, it is possible to go the other way by applying a decryption algorithm, this is achieved

by running the algorithm in reverse order, using the same key used to generate the ciphertext.

The main requirement for assuring secure symmetric encryption is to use a strong encryption algorithm, this means in the advent of attackers collecting several cipher-text they are unable to obtain the secret key. Moreover, if the secret key would be shared with someone this exchange would need to be done securely, if the key is discovered the algorithm becomes useless.

Regarding attacks, they generally fall into two categories; cryptanalysis, which studies the nature of the algorithms and tries to discover some pattern to exploit in the plain-text or ciphertext; and the second is a brute-force attack, which consists in trying every possible key until a match is found, if the key is long we can securely assure that the time to find a match is longer than the remaining time of the universe, yielding practical confidentiality.

Regarding the algorithm, the common use is block ciphers, this algorithm takes the plaintext input at fixed-size blocks intervals and produces a block of ciphertext of equal length. The most popular algorithm are Data Encryption Standard (DES) and Advanced Encryption Standard (AES). In recent years DES was been proven to not be secure due to its key length being small and vulnerable to a brute force attack considering the computing power of modern computers while AES which can have keys with double the length and larger block sizes is considered secure.

3.5 Access Control

The next step after we authenticate a principal is to prevent him from accessing resources he does not have permission to use or access.

Furthermore, our main objective can be translated into preventing unauthorized principals from gaining access to resources and allowing authorized users to handle the resources legitimately, for this we need an access control policy.

Hence we need to implement what is called an authorization database, this database dictates what types of accesses are permitted, under what circumstances, and by which users. Access control policies are generally grouped into the following categories: Discretionary access control (DAC) ,Mandatory access control (MAC), and Role-based access control (RBAC).

DAC, uses what is known as discretionary policy, controlling the access of the entity based on a set of predefined authorizations that explicitly states what is allowed or not.

MAC, compares security tables with security clearances, thus restricting or not the ability of an entity to access a resource. RBAC, assigns a role to every entity that is allowed to access the system and applies the rules associated with that role, changing a role configuration will affect every user that possesses that role.

In our case, it makes sense to divide access policies into normal users, only able to access the case data they were assigned and an administrative user capable of assigning

cases and opening them. So the best choice is using an RBAC based solution. Administrators can further directly access the machine to update the system and configure it[71].

Additionally, RBAC is widely used both in commercial and non-commercial applications, The National Institute of Standards and Technology (NIST) has also issued a standard that requires access control and administrators to support role based solutions.

The database management relations can be seen as a many to many relationships between the users and the system resources, this can be implemented using a variety of databases both SQL and No-SQL.

3.6 Authentication

The user authentication is the first line of defense when protecting access to data, there is four main way of a user proving its identity, by something he knows exclusively, something he possesses, something he is, like a fingerprint, or something he does, such as a voice pattern or handwriting.

Any of these methods could protect our application and serve as well however none of them are perfect, a password can be stolen, tokens can be forged. In respect to biometric, these bring with them a heavier administration overhead and false positives and false negatives.

The most used option is password protection, as stated earlier this does have some disadvantages and vulnerabilities, on the human side, an attacker can use social engineering attacks to infer or deceive the user into giving its password to compromise the security of the system.

Another attack is brute force over the password, however, this can be mitigated by using long and strong passwords.

Additionally, we need to protect the password in the advent of a security breach of the database occurring or if the login session is captured, this is mitigated by never saving the password in plaintext. There are several ways of storing the password safely, the best is by using a hash function with a fixed-length salt value. A hash function takes the input, in this case, a string, and produces a digital and non-reversible summary. Some examples are MD5, SHA-1, SHA-2, among others. Recently several vulnerabilities have been found in each one with different degrees, one of the most secure seems to be the SHA-1. The purpose of the salt is to serve as input when producing the hash value, this makes using a dictionary with precalculated hashes of common passwords impossible since each dictionary would have to be generated for every individual password.

3.7 Integrity

The objective associated with checking the integrity of a file is to be able to detect and preserve that file and detect any modification that might happen intended (User action) or unintended (System errors or disk failures). To protect our files and disk images the

architecture employs the Reed–Solomon algorithm, this algorithm belongs to a group of codes that allow to perform error detection. It is used widely in several applications being the most popular for distributing multimedia content like CDs, DVD's and Blu-ray, we additionally store the output of this algorithm in a Redundant Array of Inexpensive Disks (RAID)6 storage system, with all files being encrypted.

During the development of the solution, we simulated each virtual hard drive as a folder enumerated from zero to five, this can easily be modified to run in the cloud and redirect the input and output operation to virtual hard drives.

This solution assumes that we have at least 6 Hard Disk Drive (HDD) available to the program. However, this begs the question: What can we do if we only have a single HDD and if a file gets corrupted?

Reed–Solomon is a complex algorithm and explaining it to its full extent is beyond the scope of this work, we can simply define it as an algorithm that applied symbols to fixed chunks of data and can detect errors based on such symbols. Additionally, the algorithm can correct errors in the data up to a certain number of errors, this change is greatly increased with the use of RAID6 [72].

3.8 Future proof

The most important aspect of any investigation is to be able to prove that evidence came from a suspect and was not planted or tampered with.

Notably, this does not depend only on our solution. Firstly when the media is apprehended it has done legitimately, with the existence of a warrant. The officer should create a copy of the disk contents bit by bit, this can be done using The Sleuth Kit or EnCase. Also, we generate a unique identity for each file and the drive itself using a hash function.

Afterward, our architecture can use the same hash function and compare to check the file has not been tempered. In the advent of identifying a suspicious file, the detective only needs to compare the hash value to the original hard drive, to create proof that the disk image has not tampered.

3.9 Secure connection

In any criminal investigation, privacy is a very important aspect, this aspect in computer science is applied not only to what happens inside a system but also to how other systems exchange messages with it. Thus we need to make sure that when a legitimate user accesses the system, an attacker observing the traffic as the legitimate user connection cannot observe the content of the exchange or manipulate it. In other words, we need to secure our web services from spoofing and reply attacks.

A widely used solution for this is using Secure Sockets Layer (SSL) or when applied to the internet Transport Layer Security (TLS). SSL provides a way of securing Transmission Control Protocol (TCP) connections, it provides its security by defining a two layer

protocol on top of the TCP connection. This architecture can be used in a various range of protocols, however, we are only interested in the HTTP, with is how we will establish a connection to obtain our results.

Furthermore, when HTTP is over SSL it is denominated Hyper Text Transfer Protocol Secure (HTTPS) and guarantees a secure connection between the client and the server. Since all modern browsers already can establish an HTTPS connection we don't need any additional configuration on the client-side. When a client uses HTTPS we guarantee the request Uniform Resource Locator (URL), body and parameters of a request are encrypted and if captured can't be read or resent to the server.

For a server to establish HTTPS connection it needs an SSL certificate, there are several types of certificates, they differ based on the number of domains and sub-domains they can validate. In our particular case, a single certificate that fully secures a domain name is enough. An SSL certificate ensures to a client that a host is the owner of the web service they are trying to connect. This is achieved by relying on asymmetric cryptography.

3.10 Reliability

There is no point in developing a system that crashes often and can execute during a limited amount of time and then needed to be restart. When designing any piece of code we must ensure that there are no know bugs and that it can run normally without crashing.

Furthermore, this isn't determined exclusively by testing or by using a good design pattern, the choices we make on what the under-laying technology is also matters.

Therefore, if we look at our system requirements we need a solution capable of running on Linux, Mac, and Windows with little to no complexity during the install and execution phases. Additionally, we need to generate deterministic results, although some machine learning models don't always generate the same output for the same input the difference needs to be as little as possible. Another advantage of this is the fact that almost all investigations are not performed by a single individual but by a team, having the same setup and results across different machines will empower the investigation and confer an additional trust level to the solution. After identifying these requirements we came to realize the best way of deploying and making the final build readily available would be through a docker image.

Docker, in recent years, has gained a lot of traction and attention. In addition to providing a client solution for all major operating systems, it comes bundled with all the requirements a solution needs to run, thus not being needed to install other software, except of course the installation of docker itself. Not only does Docker assist in the debugging process but also as a huge and growing community behind it, meaning there are a lot of resources available to assist with its usage.

Although a stable solution, that is easy to maintain and deploy, Docker may exhibit poor performance on Windows and Mac, or at least not as good as on native Unix systems,

like Ubuntu. It also has, a significant learning curve for advanced features.

Never the less if we take into account the benefits and the downsides, Docker is the best way of making our solution widely available and opens the possibility of in the future an architecture that can support Continuous integration and Continuous delivery (CI/CD) processes.

3.11 Separation of concerns

In recent years, there has been a lot of talk about how we build reliable solutions, as such software engineers have built repeatable designs to guarantee a stable solution.

Although there are a lot of patterns they share common principles, the most common is the separation of concerns.

The term separation of concerns was first introduced in [73] around 1974, it brought along the notion that it was a bad practice to co-locate different concerns within the same code. This is very important especially when dealing with a layered application like ours.

In our case we will be using the Model-View-Controller (MVC), a software design pattern that focuses on separating concerns and reusing code, its goal is to separate the data presentation from the data itself. This is done throughout a controller, giving it its name MVC.

The View represents the expressing of a system state, the model can alter the view and the model, some examples of implementations are PHP, HTML and Applets, summarily it's the layer that manages the presentation and interface for the user.

The Model links the view and the controller together, it is responsible for storing the information, managing business logic, and rules of execution.

Although there are more valid design patterns, we choose this one because of past experience, and faster development time. When using JavaScript Object Notation (JSON) to represent system data objects, we can guarantee that future front-end clients won't need to change the back-end, making rewriting the web pages or developing an app require less time.

3.12 Labeling pictures

A typical CNN that we observed in the state-of-war needs a specific running environment to be executed, as well as other dependency or libraries and the pre-trained model. These require a lot of time to configure and are prone to error since sometimes there are additional steps in configuration the environment on the host machine.

In the previous Section 3.10, we stated that the final build would be a Docker image because we can package the image with all the necessary code and library to be executed regardless of the operating system or its state. The solution for the CNNs that will label our pictures is to also package them inside a container with everything they need. Previously the best solution would be to use a Docker inside a Docker solution, however,

this is not recommended because it can lead to many low-level technical problems, that stem from how Docker is implemented [74].

Consequently, there is another solution in which the two containers are not independent of each other and don't have problems. The solution is for one of the Docker images not to run its own daemon but to connect to the host daemon. Meaning, there will be a Docker Command-line interface (CLI) running in one of the containers and another running on the host system, thus only one Docker container is running in the machine. This is archived using the bind mount option.

3.13 Storing results

When we need to store data for the long term and retrieval later we need a database.

The two main database types are SQL and NoSQL, the development of the NoSQL database system started in the late 2000s when SQL database started to face speed and scaling issues. Previously SQL system was developed almost 30 years earlier and focused on minimizing data duplication, this was because that at the time the cost of storage greatly exceeded the cost of development time, something that nowadays is the complete opposite. SQL databases provide rigid, sometimes complex, and tabular schemes that do not scale vertically easily.

Furthermore, some NoSQL databases accept JSON document with a key-value pair and generate their columns dynamically using internal graphs in opposite to an SQL database where the tables have fixed rows and columns. As such, the developments have to beforehand know what they will be storing, and in the advent of the future needing to change they have to modify the schema. Concerning our particular case, this is not ideal because we want to allow our solution to be extended in the future as easily as possible.

The drawback of using a NoSQL database is that they generally don't respect the ACID(atomicity, consistency, isolation, durability) property, they guaranty that transactions either run completely or not at all, resists power failures and future transactions reflect the previous valid transaction. Additionally, NoSQL tends to duplicate documents and as a result, be larger than the equivalent SQL option. Recently some NoSQL have fixed these issues by supporting a compression algorithm to reduce the storage footprint.

Overall, the best solution for us is the NoSQL option, because of speed, scalability, and how easy it is to change the schema dynamically. The most used no NoSQL database systems are MongoDB, Elastic Search, Couchbase Server, CouchDB, NeDB, Terrastore, and so on.

3.14 Interface

One of our major goals when developing this architecture is to make it simple and intuitive for non-technical personal, furthermore, it should be compatible with not only

desktop access but also mobile. To design a truly scalable system we can't limit our interface to only desktop accesses, nowadays in Europe, more than 40% of accesses are from mobile[75].

However coding two different solutions is unreasonable and unnecessary given today's templates and frameworks. The solution is choosing firstly a framework and later a template that also adapts to mobile. This will make development and maintenance cheaper, won't require an app to be launch in Google or Apple store subsequently not having to pay their licensing fees.

Additionally, our solution does not require any hardware integration like Bluetooth or motion sensors making a native or progressive app even more unnecessary.

3.15 Open standards

In chapter 3.11 we stated the importance of separating concerns and using a well defined design pattern right from the beginning of the project.

The Internet has evolved a lot throughout the years and recently several open standards for web apps have appeared, some even make available stub generator for a variety of languages. These stubs take endpoints as input and generate the whole project code, meaning the programmer only needs to add the login to each function.

The OpenAPI Specification (OAS) is an open initiative with the objective of providing a common standard for creating RESTful APIs that provide a way of generating documentation and network traffic inspection without the source code.

Additionally, its definition can be utilized to obtain documentation, display API tools, testing tools, and client stubs[76].

The most recent specification is version 3.0.0, there are enormous code generators written in very different programming languages like Go, Java, Typescript, and Ruby, all open-sourced themselves. Regarding usage and popularity, the most widely used is Swagger.

3.16 Summary

In this chapter we defined and presented the design principles that are necessary to assure when designing our solution, they are confidential, access control, authentication, integrity, transparency, future proof, data retention, and reliability.

Confidentiality is achieved when we encrypt the data we store and this can not be broken easily, as such we should use the AES algorithm.

Additionally we presented several option to implement access control, they were DAC, MAC, and RBAC. Since in our solution, we want to associate a case with a fixed number of user, and this being a relation of 1 to 1 to most suitable is RBAC.

In order to assure that a user is who we claims to be we need an authentication mechanism, for this we will use a password system.

Furthermore, we must assure that our data is not tampered with or altered, this is achieved by the use of error codes and by possessing multiple copies to restore in the advent of failures. The error codes used were Read-Solomon and RAID6.

Future proof means we can prove that a suspicious file came from the suspect's hard drive, we can prove this by comparing the signature of the file with its original.

To protect the traffic between a legitimate user and the system, the connections should be protected using SSL.

The core of our solution is labeling an image correctly and efficiently, this is done by using two state-of-the-art CNN that can determine the age of a person and the level of NSFW. For us to store these results we choose a NoSQL database system.

There are already standards that reflect the best practices when developing a system, a common one is OAS, we will follow it to achieve a robust, easy to maintain, and scalable solution.

In the following chapter, we present how these requirements are materialized in a specific architecture, which modules are needed, and their interactions.

Chapter 4

System Implementation

> Ideas are a dime a dozen. People who implement them are priceless.
>
> *Mary Ash*

4.1 Overview

In this chapter, we present the main modules of the project that were developed in the past months.

The final architecture can be split into 7 unique modules Elasticsearch, Feature Extraction, Reed-Solomon with Keystore, PostgreSQL, API, and Web page. Each of them will be explained in greater detail throw out this chapter, and how they meet the requirement from the previous one.

4.2 Execution Flow

The goal of our solution is to receive a disk image as input and allow for the user to search throw the results using a friendly GUI.

However, this is not straightforward, the process starts after a disk image is obtained. Firstly the inspector needs to upload it to our systems, for this we will sign in to the system, create a case associating that case information and bind the captured image to it. After the upload is finished we will have the option to share access to this resource with other investigators that also are working and is relevant for their work. In regard for this part to work, and following the client-server model we need a server running to receive

the file, since we are building a solution with reliability in mind this will be contained in a Docker image. When saving the image, we need to guaranty that we can detect errors and can not be tampered, thus we will use a RAID6 architecture with Reed-Solomon and encrypt each file. The access control will be enforced by a database following an RBAC schema.

In order to start the search, the user needs to specify which image we desires the system to start with, this will call the Docker images containing the trained CNNs and a thread pool to concurrently iterate over the desired image, each image result is saved in ElasticSearch.

Graphically we need a login page to authenticate the user, afterward a landing page to view the state and health of the system. A screen to create a case entry in the database, a start scan to tell the server to process the specific image, and in the advent of being an administrator to give or revoke access to other users. Finally, a results screen to show and give the option to vary the search options. All of this needs to be compatible with mobile and desktop access.

Finally, when the scan is over and the image search is completed the user can query in the result tab and customize the search option, these queries are processed by the server and served by ElasticSearch.

4.3 Architecture modules

In the previous chapter, the system was specified as a distributed system using client and server modules, Figure 4.1 shows seven different modules, Elasticsearch, Feature Extraction, Reed-Solomon with Keystore, PostgreSQL, API, and Web page fit the general client-server architecture.

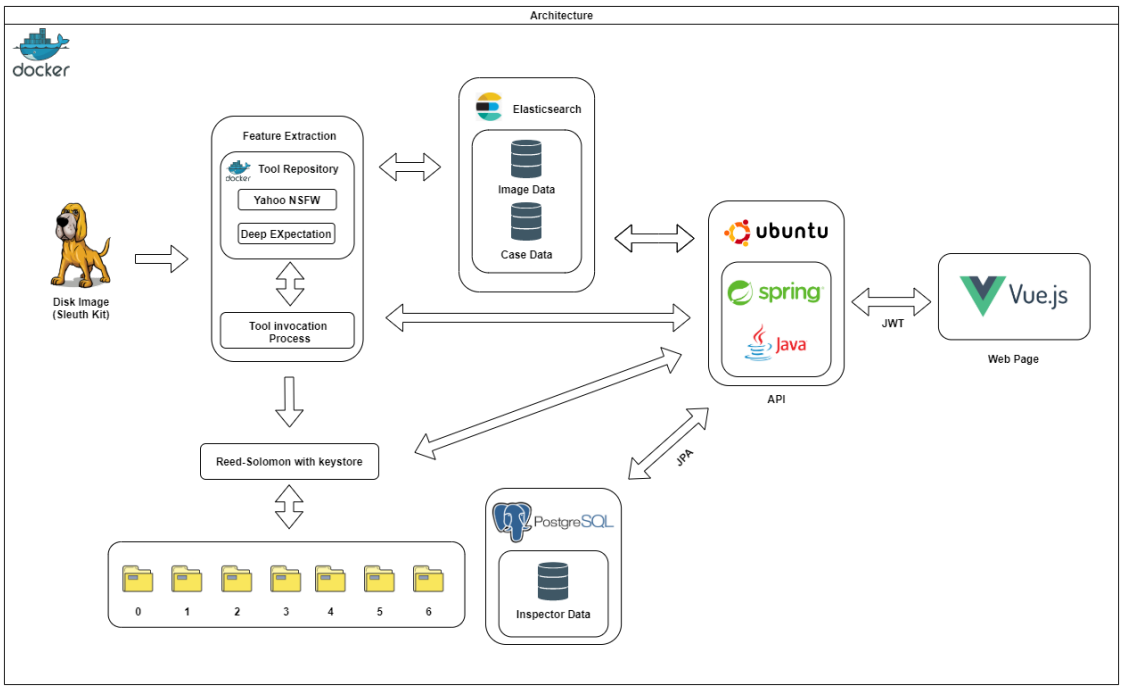

Figure 4.1: Final Architecture

The Elasticsearch module is responsible for making information associated with an image or case available and searchable.

Feature Extraction which contains the Neural Networks, python, and necessary frameworks is responsible for evaluating valid pictures and generating the information to be presented to the investigators.

Client authentication is performed by the spring framework, which is equipped with an authentication table, stored in the PostgreSQL database. This is similar to the structure present in UNIX systems(inspired in the /etc/password file) where each entry is associated with a user. After successful authentication by the client, a JSON Web Token (JWT) is sent, it is up to the spring to inform the control policy, for this, it has a configuration file where it implements a table where each user is associated with a password and a value Boolean who dictates whether it can be authenticated.

In a Classical UNIX system, this entry would have three parameters R (Read) , W(Write) and E(Execute) in our architecture the use A (Admin) and U (User). A user can only open new cases and search the cases it has previously assign by an admin, an admin can view all cases, assign and remove a user from a designated case.

Additionally, the implemented solution offers guarantees of integrity, authenticity, confidentiality, and high availability. The use of Reed-Solomon brings many advantages, when used with the encryption of the data it is even better because it protects the access to the original file to someone other than the client, assuming the key is correctly protected and that the cryptographic methods used are not possible to break in a reasonable time.

4.4 Image format

The architecture support for disk images can be divided into three major formats, Raw Single, Raw Split, and EnCase:

- **Raw Single** this is a complete bit by bit copy of the content of the disk image, it can be *.img, *.dd, or *.raw.
- **Raw Split** meaning the copy of disk image is splinted across multiple files. The advantage of this is performance, on a multi-core machine the search can be paralyzed more easily, the most files are typically named *.001, *.002, and so on.
- **EnCase** the EnCase format is a specific priority format that adds additional information check to detect image tampering like MD5 check for the whole image and Redundancy Check [77]

The module that is responsible for reading and making the data available for the other components in the architecture is the Sleuth kit, and it fully supports all the format above, in the case of an EnCase image, it even checks if the file has not been tampered with during processing. To generate testing images to validate the final build of the proposed solution we used FTK Imager. FTK Imager or Forensic Toolkit Imager is widely used to generate disk images from live hard drives, USB, DVDs, CDs, or even folders. During the generating phase it searches for deleted information like files or emails, at the end, it also outputs the MD5 and confirms the image integrity.

4.5 Key management

Confidentiality is achieved by encrypting the files with a symmetric key algorithm AES.

We used PBKDF2 to generate a key from the user's password, however, the seed used as a password hash and not the password itself, this was necessary for the server to be able to generate the same key without having to save the password in plaintext, this key was then used to encrypt the data with AES. To protect against brute force attacks the number of cycles in the key generation was set to 150000, taking about 700 ms to calculate, rendering this type of attack unfeasible. NIST also published guidelines recommending the use of PBKDF2 [78].

In the final build, we provide default certificates to encrypted and secure the solution. This should be changed with new certificates. The user can indicate a Keystore and its respective password, thus allowing for a different key for each file. Furthermore, the Keystore will always need to have to entry, one with "Rivest-Shamir-Adleman (RSA)" for signing the file and another with "metadata" for encrypted the metadata part of the file

All the data stored by the server both upload and generated during processing will use these certificates thus making this a vital part of the configuration process.

4.6 RBAC

As stated in Section 3.5 the relations between the users and the roles is a many to many relation, in our environment we can except for the users to change frequently, and the users can have one or more roles. In contrast, the set of roles rarely changes, each one represents a specific set of rights over the systems resources.

Furthermore, a simple implementation of an RBAC is an access matrix, this concept is present in Linux systems and is used to manage user's permissions over the files. The top matrix relates the users to their specific roles, in our system there will be more users than roles, meaning every matrix will be marked signifying the user is assigned to that role. Regarding our specific scenario, a user can be assigned multiple and different roles depending on the investigation at hand. The lower matrix controls the roles, there will be fewer roles than cases.

Additionally, RBAC easily comply with the principle of least privilege by only assigning the minimum roles to operate over the needed cases and perform the necessary actions that it is required to undertake his work.

We implemented this access matrix using PostgreSQL, the choice was based on its integration with Java Persistence APIs (JPA), allowing for the tables and relationships to be self-generated making future changes over the user Plain Old Java Objects (POJO) classes direct effect the table and relations within the database.

4.7 Logging

To integrate logging into our system and provide a way for the system administrator to trace future problems we used the logging framework that comes preconfigured with Spring Boot.

The major advantage of using this framework is it requires zero configuration, the only addition that we made was to save the logs in PostgreSQL, this makes them available at any time and are still present after a system restart.

In our particular case, these logs provide additional system information when it comes to the reporting phase and the chain of custody.

Lastly, to help in our development this can be used as an initial debug tool and in the future help, other developers and support teams work with the project.

4.8 Sleuth Kit

We stated in the previous chapter that it would be unfeasible to read every disk image independently of its file systems, thus the solution was to use The Sleuth Kit.

Also know in the forensics community has the TSK it is simply a collection of UNIX-based command, however, it does not come bundled in with most Linux distribution the only exception to this is BackTrack which provides the TSK commands out of the box.

The Sleuth Kit is capable of analyzing different file systems such as NTFS, UFS 1/2, FAT, ExFAT, Ext2, Ext4, HFS, ISO 9660, and YAFFS2, these formats can be read separately, raw format, expert witness, or in file and volume system Analysis. During testing, we found that the most widely used format was raw (dd).

Additional features include recovering deleted files, reporting on all removed files, and making the files present in the image searchable. In our solution we did not take advantage of the built-in search feature of the TSK instead we annexed the file to our databases, this was needed to provide a faster preview for the users and support additional analytics.

Since TSK is a command-line set of utilities there have been a few GUI developed to help non-technical users to use the framework, these include Autopsy and CAINE, both are paid solutions. Furthermore, we don't need a GUI we can access the TSK directly through the command line or its library.

Regarding the integration we opted to invoke TSK directly through the command line event, this was due to the ease of integration, portability, and parallelism offered by this option.

4.9 RAID

The system is designed to maintain the chain of custody since this is not enough to maintain the integrity of our files, we also reset to other techniques to prevent the event of physical data loss, this can be the result of a power surge, hardware issue, human error or even intentional damage.

In order to protect the evidence and the processed result, we must duplicate the data across multiple drives, to guaranty a high fault tolerance rate we use RAID. RAID can be seen as a collective of N hard drives that are physically independent.

The most common configurations are RAID1 to RAID6, since the information we are storing has the potential of being crucial we choose RAID6. The advantages of using this configuration are the guaranty that 2 of the HDD can fail and the information can still be recovered, the disadvantages of this approach are a slower write and a more complex control, and costly for the HDDs[79].

4.10 Feature Extraction

The main core of our solution and the advantages we set out to achieve were to identify amongst all the files of the hard drive those that contain child pornographic material and index them for future search by the users of the system.

The analysis starts when the user selects an image, this image is then fed into the Sleuth kit interface and iterates over each file. The files are sent to a pipeline, this pipeline request a thread from a thread pool, the number of threads in this pool depends on the

hardware of the server, during testing we found that a laptop with an Intel Core i7-7700HQ, 16 GB DDR4, GTX 1060 has able to handle 6 threads processing six images simultaneously.

When entering the pipeline an SHA-1 is calculated for each file, later it is compared against the ElasticSearch database. In the advent of the file being processed in a previous scan, a record will be found and returned, no additional operation will be executed. Since SHA-1 is secured and to date there is no known attack (unlike MD5) we can safely use it, additionally being a hash function it is not possible to revert this operation.

From this point the pipeline is customizable, we focused heavily on the image processing part, so our python framework starts by detecting if the file is an image. In the advent of being a thread is called from the pool, and the CNN with the train models for calculating the NSFW score and for estimating the age of every person in the image is also executed.

The final step is joining the results of CNN with the SHA-1 and transmitting them to ElasticSearch in the form of a JSON file over HTTPS.

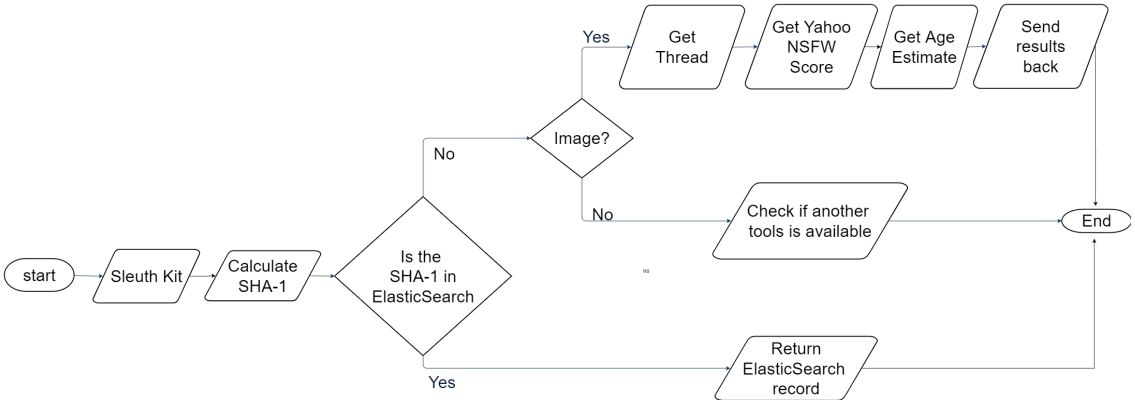

Figure 4.2: Flow of file in pipeline

4.11 Image Classification

In the previous Section 4.10 we discussed the general process that occurs when a picture or compact archive is detected and the flow the system follows to extract the desired meta-information for later searching and result presentation.

The models are contained in Docker containers and are ready to be executed at any time, looking at the two CNN we can see some similarities. They are both written in python, required the same library's, mainly pandas, numpy, tensorFlow, and scikit.

As a result, we decided to encapsulate each of them in a Docker image, this choice was made because the execution of these models is the most time-consuming part of the analysis process, a bottlenecked. In the advent of in the future, someone tries to scale this part can be executed on multiple machines, over a huge amount of data, and still get the results in a reasonable amount of time, ideally, these models would be run in a cluster.

As a result, every model can be executed independently and asynchronously, creating a machine learning environment with all the necessary dependencies, frameworks and library's.

Another alternative to creating docker images would be to use a Python environment such as conda and virtualenv. The problem with this solution is they don't support nonmajor Python dependencies making a large scope of libraries not supported.

A Docker container can fully encapsulate the code, dependency's and the model, making this solution consistent and portable, this is the reason they are scalable on a cluster. In the case of future collaboration and future works, we can share not just the code but the entire development environment itself by simply pushing the generated image to Docker.

On the other hand, container images can also be executed in a cloud service like AWS, Azure, or Google Cloud Platform.

Regarding how the models are actually executed our solution support two forms, the first is over HTTP Rest request in with we send the image for processing and the container return a JSON that contains the data generated from the models, in this case, Docker is actively running in the background and responds to request. On the other hand, the Docker containers can also receive the picture as input in the form of a path from Docker CLI, reads the picture from the operating system path, and return the results. The latter is mainly for debugging and testing purposes.

4.12 PostgreSQL

We stated in Section 3.5, that the best option for managing access control would be using an RBAC solution, for this, we need a database to store the relation and data. There are several options, however, we choose PostgreSQL, because of the ease in integrating with Spring, and we can take advantage of the JPA framework. This means we do not need to create a schema or SQL query, JPA can create and manage everything during execution.

The development of PostgreSQL started in early 1986 in a project lead by professor Michael Stonebraker that took place at the University of California in Berkeley. It is credited as one of the first database systems.

Initially, it utilized another interrogating language, it was only in 1994 that support for SQL was introduced by two students of professor Michael Stonebraker. The system continues to evolve to this day, since 1997 it was open-sourced.

In the last years, it became one of the most used database systems, this is because it is scalable, extensible, and open source.

Additionally, we choose this database system because of the ease with which it integrates with JPA, meaning Java can automatically save change made to the user account to a persistent state while offering the programmer complete abstraction.

This is possible due to a standard API that structures and maps the database for us, the scheme, foreign keys, constraints are generated based on the relation between Java classes and object on run-time.

4.13 Elasticsearch

Elasticsearch is the leading enterprise search engine, it is built on apache Lucene, open-sourced, and written in Java. Shay Banon, the creator, built the solution from scratch with scalability and efficiency in mind. The main advances of using this engine to store and search the information produced by the feature extractors are that it allows aggregation and pattern finding more rapidly than a traditional SQL database. Another main reason why we choose is the fact that document-oriented, using JSON to serialize the input.

Additionally, Elasticsearch is schema-free, meaning in the future if the feature extraction pipeline is changed to include more features, Elasticsearch will automatically map the new property without the cost of reducing performance on future searches.

Thus making it a perfect choice for our architecture, to protect Elasticsearch from being available to requests that are not inbound from the Spring framework, each connection requires authentication to access the data. This is done by REST endpoint and the keys are only made available to our Spring server, ensuring the request to view case data has to come from authorized server.

4.14 Backend

In order to develop a system architecture, the first aspect to consider is the design pattern that is most adequate to guaranty a deliverable that in the future will be easy to manage, modify and understand by others. Accordingly, it is good practice to separate the presentation layer and the data access layer, the most common design pattern for this type of situation is MVC.

The additional advantage of using MVC is the fact that it reduced development time, making it even a clearer choice. For this reason, we must choose a programming language that supports this design pattern, we choose Java because of past experiences. However, we did not use Java by itself since the timeline of delivery was strict we used Spring. Subsequently, this brought with it several advantages like security which provides a customizable JSON Web Token (JWT) that can interact with our costume RBAC and automatically authenticate the principal and apply the security filters defined to each rule of there are any associated with it.

Also, we needed to define a RESTful API, so we defined one using the OpenAPI Specification, this means we started by looking at the requirements of our architecture and defining each endpoint. This was done on the official Swagger page, the output is a stub written in Java using the Spring framework. This stub already come with the previous documentation, deployable environment, clients and helps in future testing.

The final RESTful API was also tested using Postman and the final published docs were generated.

4.15 User Interface

In recent years several different JavaScript frameworks have appeared, the most popular are React, Angular, Vue, Meteor, Ember, and Backbone. There exist many more, choosing one from this list is not easy, the reason for the existence of so many is that each one tries to solve a specific problem, this naturally brings several advantages and disadvantages associated.

The best choice depends on the type of project and the requirement associated was stated earlier each framework is optimized for a particular use case. For instance, React ensures a fast rendering, guarantees stable code, and is Search Engine Optimized (SEO) friendly. Angular on the other hand provides data-binding capabilities, easy unit testing, and dependency injection to separate concerns

Since the main objective of the dashboard is to show and make searchable the results of the images we need a framework that can provide a way of creating a bridge between the user query and ElasticSearch.

After careful research, we found the best choice is Vue.js, this is because the library know was *ReactiveSearch-vue*, an open-source library that comes out of the box with a wide range of highly customizable User interface (UI) components that interact with ElasticSearch automatically.

Subsequently, if we analyze everything to this point we can see that we would need to implement.

The user experience starts with login, by default the system comes with a prepared account. However this can be changed, or a new account created. This leads us to the register page, the end-user need to be able to add colleges, but not everyone that register should be able to access what they please. This brings us to Access Control from where we used an RBAC. Furthermore, we need to assign the detective to each case, each detective only need to access the cases that they were assigned respecting the least privileges principle

Finally, we need to make the indexed content searchable, and searchable by case name, NSFW percentage, and ages.

In the subsection, we will delve into more details about each screen, the features, and the technical decision that took place.

4.15.1 Login

The first for the user to interact with the system started in the login screen, all endpoint is protected using end-to-end encryption and required a token to prove authenticity for that request.

The token used is a modified JWT that contains, the user's roles, name, and token expiration date, when the token expired the users need to re-login. Figure 4.3, shows the landing page of our front office.

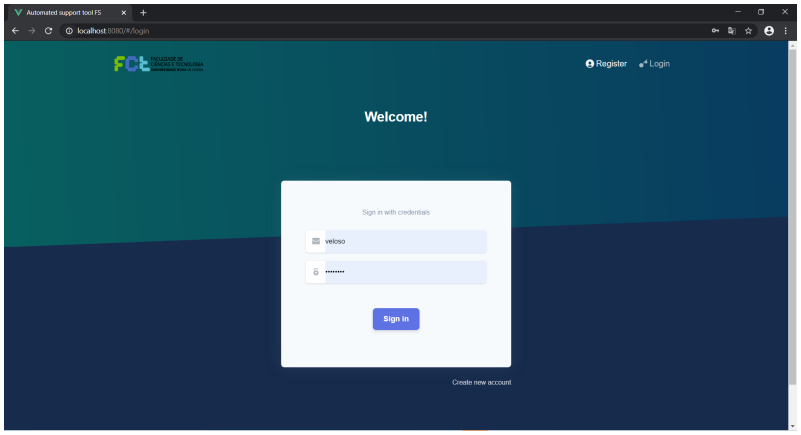

Figure 4.3: Login Screen

4.15.1.1 Dashboard

After a successful login, the user is greeted with the Dashboard shown in Figure 4.4. The idea behind this landing page is to provide a quick overview of cases, previously processed pictures, images available to the detective, latest messages from the system, and optionally if the detective wishes to check an individual image for benchmark purposes it is possible to use the analyze option on the bottom right.

The messages shown are divided into three types, information, warning, and error. Information is logs of events being performed by users, these are visible to the systems admins, some examples include the creation of cases, deleting, starting the search, or adding colleagues. The warning includes failed login attempts and exceptions that are not critical and can be recovered. An error can be a connection problem between Spring and ElasticSearch or PostgreSQL or an error in the Java I/O API.

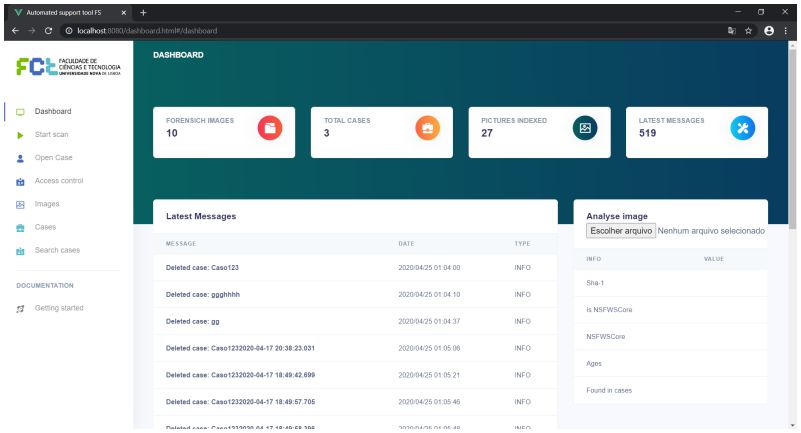

Figure 4.4: Dashboard

4.15.2 Open case

The first step after an investigation takes place, is to register disk images in the system. This will let the system know how to index the evidence and which users can view and run operations associated with the case resources. The system requires a case name, starting date, optionally an organization unit, forensics images to associate, and a user that can access the result and start scans. Figure 4.5 shows the screen in greater detail. The images can be uploaded throw the Start Scan Figure 4.6 or in case of running the server from a virtual machine by placing them in the shared folder provided by VirtualBox.

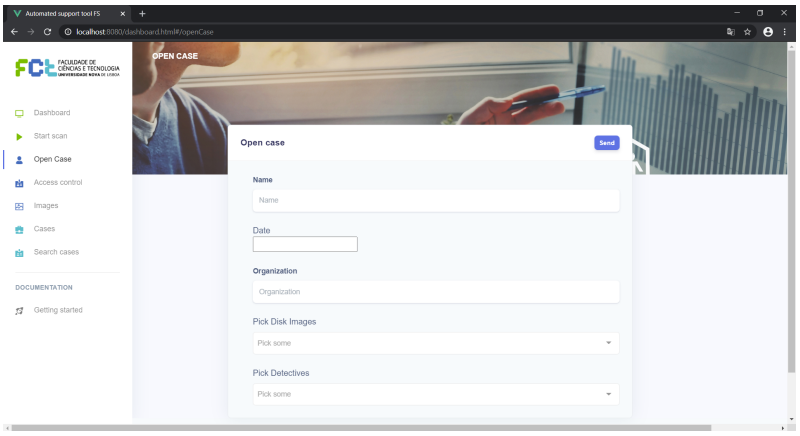

Figure 4.5: Open Case screen

4.15.3 Start Scan

After creating and assigning detectives to a case the next steps is to choose which images are a priority to scan, as stated earlier the scan is made concurrently with a fixed thread pool that depends on the available hardware in the event of a time constraint it is recommended to star with images the inspector believes will hold the most significant proofs to the case. In future work, we discuss how we could improve this by performing an initial fast scan and providing the detective with additional information to help in the decision-making process. Figure 4.6 shows the start scan screen, it shows the name, File system type, date of creation, and if there is an operating system present the name.

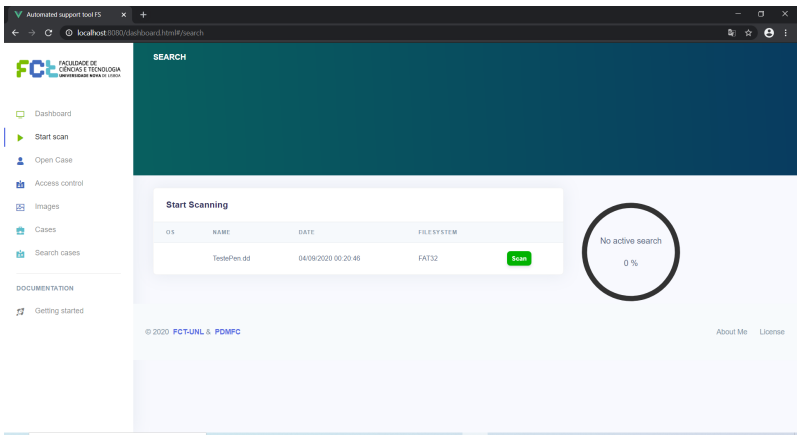

Figure 4.6: Start Scan Screen

4.15.4 Role Control

Another important part is access control, not all the users of the platform need to view all the cases. Furthermore, not all of them need to have an administrative right over the case, this is in line with the principle of least privilege.

When the case is created in the system the user that created the entry is automatically assigned the role of administrator. Thus having the right to assign more users to the case and if necessary also nominating them as administrators.

Additionally, operations available to administrators are removing project access to a user and demoting them to a normal user, the user interface can be seen in Figure 4.7.

Since a normal user can't change anything in this screen because we lack the rights and it is not necessary to know the other team members this tab option is not shown in their menu. Furthermore, if they tried to access they will get a not found error.

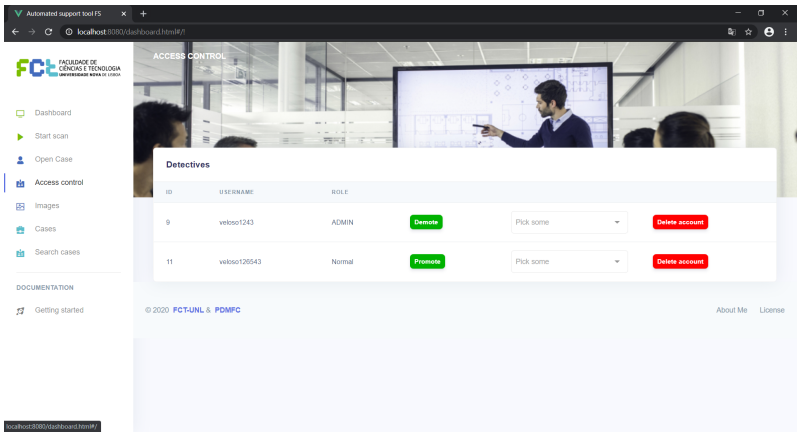

Figure 4.7: Role Control Screen

4.15.5 Searching

The most important piece of the front-end is the search tab, it is the main reason and the core of this work. As a result, great care was taken in designing the graphical user interface, the main target of the audience is digital investigators with very little to no technical knowledge.

As we can see in Figure: 4.8 we can filter by case name, NSFW score, and ages, the currently selected terms are shown in the button and allow for a better user experience.

Additionally, if the user hovers over a picture it will display a preview of the indexed information, the on click will show the image in its original size and format.

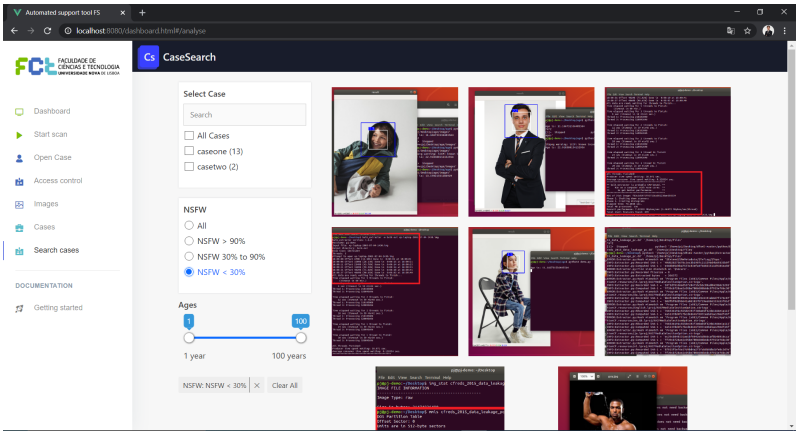

Figure 4.8: Search Screen

4.15.6 Mobile

In the previous Section 3.14, we stated that in Europe almost 40% of access occurs from a mobile device, thus our solution would also need to be adapted to this paradigm. This is exactly one of the results of using a progressive template. Figure 4.9 shows how the home page adapts when running on an Oneplus 6, the menu was the same option as it would from a Desktop computer. Additionally, all the screen adapt equality well.

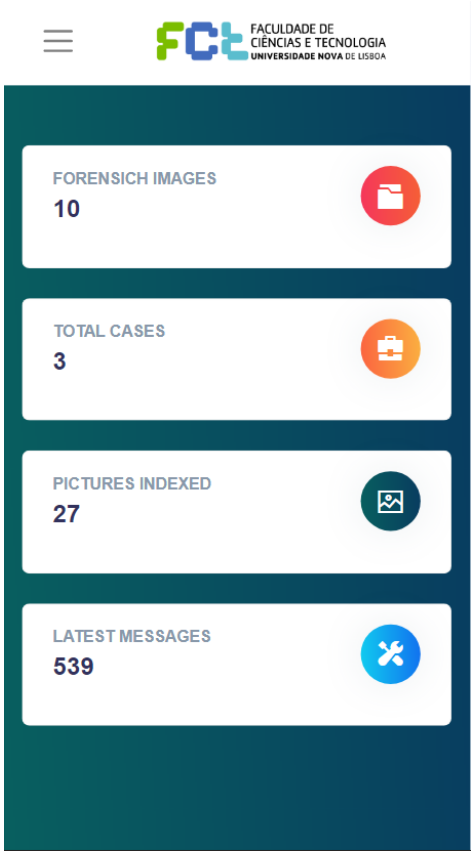

Figure 4.9: View from a mobile device

4.16 Summary

In this chapter, we presented a typical flow execution of the system when an investigation starts to how the results are presented and how they can be searched.

The execution starts with the user inserting an image into the system, this will be saved using error code detection, duplicated, and encrypted. After a while the files in the images is iterated over using The Sleuth Kit, a thread pool is running with contains a Docker image containing the pre-trained CNN models, that will receive an image as input and estimate the age and NSFW score. These results are stored in ElastiSearch. Later they will be made available throw the search tab, the user, supposing we has access can limit the age, filter by case, and NSFW score.

The system interface was developed using a JavaScript framework know as Vue.js, the available screen are the login, dashboard, create a case, start scan, modify a user role, and search.

In the following chapter, we explain how we tested the final build of our solution, which dataset we considered relevant, and the conclusion we can infer from the results.

Chapter 5

Evaluation

Measure what Matters.

John Doerr

5.1 Overview

In this chapter, we will present how we tested our final solution. Our main goal is to demonstrate that our solution would theoretically work in a realistic scenarios.

To this end, we begin by describing our experimental methodology in Section 5.2, followed by a presentation of the tool that we developed to help us in conducting our experiments in Section 5.3 and then, detail how we structured the dataset in Section 5.4.

Since we divided the problem into two subproblems we also divide the results Chapter into two parts. Firstly we analyzed the NSFW CNN. Then we evaluated the accuracy of the age prediction CNN. We also tested several different scenarios of the feature extraction pipeline with benchmarks that represent good, bad pictures, removable devices, and hard drives that contain major operating systems.

5.2 Experimental Methodology

We intend to test our solution in a real-world scenario, however, faced with the current pandemic this was simply not possible. To overcome this we divided the problem of finding child pornographic pictures into twofold subproblems. Firstly detecting if in the picture there is a child present and assessing if the image contains any nudity. To solve the first problem we used an age estimator CNN and for the nudity assertion the Yahoo NSFW CNN.

In alternative we resorted to using datasets, we should test our program in a device that the investigators would take to the field. Accordingly, we choose a mid-range laptop with an Intel Core i7-7700HQ, 16 GB DDR4, and GTX 1060.

During the testing, we tested each part of the solution individually (i.e. NSFW detection and age estimation), together, mixed with pictures of cats, mixed with household objects and operating system files. Although the multiple file system types that are most common such as NTFS, FAT/ExFAT, Ext2, Ext4 and HFS.

5.3 Experimental Tools

In order to evaluate and assess the performance of your solution, we will use several datasets which contain several thousand pictures, analysis of each picture individually is unfeasible. Thus we created an automatic system for query the build for a feature, saving the output, and comparing it to the information labeled in the original dataset.

The testing program can query three different parts of the solution, the two subcomponents responsible for extracting age and evaluation NSFW value respectively, and the main module responsible for analyzing the disk images globally. Thus we created a tester capable of taking the dataset, inputting them to the different components, and saving the output for evaluation.

The input of the age and NSFW predictive modules are images and the output is a number, in the case of the age an integer from 5 to 100 and in the case of the NSFW a percentage value that represents the confidence the CNN believes the images are Not safe, 100% being completely sure the image is pornographic and 0% completely sure the image is safe. The testing program iterates over the inputted images to obtain the result and changes the name of the picture to reflect the results, the first part of the name represents the age, and the second separated by a hyphen the NSFW score of the CNN. In parallel, these results are also placed in an Comma-separated values (CSV) file for later manual processing. An example would be XX-87.JPEG, this means we tested only the NSFW model that attributed a score of 87 in 100 to that picture, meaning that it is very likely pornographic. Another example would be 23-XX.PNG, meaning we only tested the age and it predicted there exits one person with 23 years of age in the advent of multiple persons the program separates the ages by using a slash in the name and the CSV a comma.

Regarding testing the two models simultaneously it is done when testing the module itself which takes as input a forensics image generated from a memory device. The testing program mocks the iterator and feeds it the files to be analyzed, some examples are operating systems files, compress archive, text, pictures, and BLOB's. The output of a picture looks like 23-44.JPEG and the CSV reflects this data.

5.4 Dataset

NIST, provides several forensic images that were created to mimic a normal hard drive extracted from a potential suspect, they include different Windows versions, pens, and camera cards. They can be divided into six file types, images, graphics, documents, archives, audio, or video. Our system however is only capable of analyzing valid pictures or in the advent of a compressed file containing pictures and not being password protect, extracting them and processing them. Nevertheless, in a real-world scenario, we would encounter similar files, so even if we don't do any feature extracting from them we will still include them to make it as closes as to a real disk image as possible[80].

Additionally, we need to focus on the content of our pictures, some need to be NSFW and other need to contains children. The goal is to test if our solution can detect individually if a person is nude and if there are children in the picture. The goal of testing each part separately is to overcome the fact that it wasn't possible to test in a real-world scenario and if we detect these properties individually them we won't have to deal with the use of unethical picture and guarantee with a certain degree of safety that our solution will be able to achieve its goal in practice.

Furthermore, both of these datasets need to be authorized for academic usage. Accordingly, we found a NSFW scrapper containing images from famous pornographic movies [81].

Regarding the age estimation, we used a dataset that contains people of different ages, different racial groups, and children. In order not to bias our model, which was trained with pictures from celebrities from IMDB, our testing dataset needs to contain pictures of nonfamous people, labeled with their respective ages. Thus it is a vital piece and needs to be tested with a specific dataset.

Moreover, to test if we are only detecting people and not animals we will also use a dataset that contains several animal pictures to test for false positives [82]. This dataset contains about twenty-eight thousand animals including animals such as dog, cat, horse, spider, butterfly, chicken, sheep, cow, squirrel, and elephant.

Additionally, not all the images will be of animals, children, or adults some will simply be of landscapes, rooms, or objects. These images are not relevant to our solution and should be ignored. Nevertheless, they should be included in our testing, for this, we used images from Googles Open Images Dataset. This dataset includes more than 9 million varied images with rich annotations divided into 600 categories, we used the Watch, Axe, Drink, Laptop, Mug, Tap, Calculator, Apple, and Dice categories.

5.5 Evaluation

In order to automate the testing and later validation, we created a program capable of generating the tests and later directly querying our build and comparing the results.

The evaluation was performed using several test disk images, first, we test each dataset individually and later mixed them.

In conclusion, the images that were tested included operating system images, unreadable images, disk images containing compacted files, an image containing only animal pictures, an image containing only NSFW pictures, another containing only non NSFW pictures, and a mix of both.

Then we mixed these clones of pictures with a different operating system, mainly Windows and Linux, removable drivers, and memory cards.

5.6 Results

5.6.1 Not Safe for Work testing

Regarding the testing of the NSFW model, we used a dataset that was authorized for academic usage. Accordingly, we found a NSFW scrapper containing images from famous pornographic movies divided into subcategories such as drawing, Hentai, neutral, and pornographically. We are interested in the neutral and pornography classes to benchmark the solution with real NSFW pictures and non NSFW, thus allowing us to compute the accuracy [81].

In order to assert the correct classification of the images we calculated the tpr and fpr, this represents pictures that the model classified correctly as being NSFW. The tpr is achieved by adding the sum of the true positives and the false negatives, False negative (fn), which in this case correspond to NSFW images which the model has classified as not being NSFW.

Thus the tpr tells in what proportion the NSFW images are correctly classified. The opposite of this is the fpr, which informs the percentages of NSFW pictures that was wrong classified.

The results are reported in Table 5.1.

Dataset	tpr	fpr
NSFW	0.938	0.056
SFW	0.911	0.038
NSFW and SFW	0.901	0.061

Table 5.1: Results of the tpr and fpr for the evaluated NSFW classifier

Looking at Table 5.1 we can view how accurate the CNN is, from these results we note that in our experiment every 9 in 10 pictures were labeled correctly and less than 1 in 10 were labeled as NSFW incorrectly.

In conclusion, we can safely state that our solution can distinguish between a NSFW and a NSFW for almost every picture in the dataset. Thus providing a promising validation that it would work in a real-world scenario.

5.6.2 Age prediction testing

In order to test the age prediction module we used the All-Age-Faces dataset or AAF for short, which contains 13322 face images, ages 8 to 80. This dataset was built to be used for age prediction and gender classification. Each picture is labeled with the individual's serial number and specific age[83].

Additionally we tested specifically for children, this being our primary objective in finding and labeling correctly. For this purpose, we used an additional dataset containing more than 1200 photographs of over 100 children, ranging from ages 2 to 8 making seven different facial expressions, happy, angry, sad, fearful, surprise, neutral, and disgust [84].

To better understand if we were labeling our target demographic correctly we divided the dataset into three subcategories, ages 2 to 8, 8 to 18, and 18 to 80. Thus representing a child's typical age gap, teenagers and an adult. The oldest person in the dataset gives the age limit.

The results of this test are shown in Table 5.2.

Dataset	Exact	1-off
2 - 8	64.0 ± 1.2	96.6 ± 0.9
8 - 18	55.6 ± 2.1	89.7 ± 1.8
18 - 80	64.0 ± 4.2	96.6 ± 0.9

Table 5.2: Age group estimation results [mean accuracy ± standard deviation](%)

We can assess from the previous table that the model is not completely precise, however, if we consider a margin of 1 year the accuracy greatly increases, and since being off by one is only relevant if the person is 18 or 19 years this model is working adequately. Furthermore, if we take into account the results for children age 2 to 8, and the accuracy being of almost 97% it is a very promising indicator of how well the solution would behave.

During the analysis of a forensic image, not all images are going to be of people. This is another scenario that needs to be tested, in the advent of our solution encountering these images, it needs to ignore them.

This is because they do not contain people, thus not being relevant for the problem we set out to resolve.

In order to test this, we need to have a test with animals, objects, and landscapes. As such, we use an animal dataset that contains about twenty-eight thousand animals, some examples are dogs, cats, horses, spiders, butterflies, chickens, sheep, cows, squirrels, and elephants [82].

Further, we needed a dataset specifically made for object detection not containing people or animals, for this we used Googles Open Images Dataset and selected the Watch, Axe, Drink, Laptop, Mug, Tap, Calculator, Apple, and Dice categories.

The results of this test are shown in Table 5.3.

Dataset	tpr	fpr
Dog	1.000	0.000
Cat	0.991	0.005
Horse	1.000	0.000
Spider	1.000	0.000
Butterfly	1.000	0.000
Chicken	1.000	0.000
Sheep	1.000	0.000
Cow	1.000	0.000
Squirrel	1.000	0.000
Landscapes	1.000	0.000
System files	1.000	0.000
Corrupted picture	1.000	0.000
Invalid picture	1.000	0.000
Googles Open Images	0.977	0.041

Table 5.3: Age group estimation results when applied to animals, object, landscapes, system images, corrupted, and invalid pictures

The tpr represents if the images were correctly ignored and the fpr represents if the CNN mistakenly detected somehow a face and tried to predict its age.

The CNN behaved accordingly to what was expect, having only mislabeling some pictures of Cat and the Google dataset as people but ignoring almost every picture.

To summarise, the CNN when applied to people it can correctly distinguish between a child, teenager, and an adult, even if we have to give a tolerance of one year. Additionally, when applied to pictures that do not contain people, such as animals, objects, invalid pictures, corrupted pictures, or landscapes the model correctly ignores these images.

5.6.3 File System testing

In order to access the files within an image, we used The Sleuth kit as a middleware framework. The Sleuth kit is already a reference in the forensics community, it is still being developed after several years and continues to improve each day. Also, it comes bundles with Backtrack, a Linux distribution focused on pen testing and including several GUI programs to provide an easy to use interface for investigators to analyze images without having to know the details of the operation of The Sleuth kit.

However, we did confirm that our prototype could open the most common file system types. Additionally, the source code is available through a GitHub repository that includes several functional and unit tests for the Core and KeywordSearch NBMs. Throughout our architecture we don't use Keyword Search so these were ignored, the others were executed and validated. We mainly focused on the recovery of deleted files and iterating the files of a disk image.

5.6.4 Simulating real disk images

The final step in our validation is checking if we can iterate over all the files and access them for processing and guarantee the file didn't suffer modifications. A way of ensuring this is by calculating the MD5 before the file system image is created and then comparing it to the file when the extraction process occurs.

This is already an option that can be enabled through the use of a flag when creating the images in The Sleuth Kit, when the image is generated if the flag is active, in addition to creating the image that reflects the physical drive there will also be a text file with the name and location of each file in the disk image and the corresponding MD5 for comparison. Due to this, the unit test validates the feature and during our test, the hashes found in ElasticSearch also matched the ones found in the file.

Regarding the recovery of deleted files, this depends on how they were deleted and if the file system is damaged or not. After a file is deleted if the space is not written over by another file then the Sleuth kit can detect and recover the deleted file.

The work presented in [85] tested the recoverability of a deleted file across FAT, NTFS, and Ext. The test image was of a hard drive that successfully shut down having deleted several BLOB of 512 bytes that were removed through the recycling bin. This was done over FAT, NTFS, and ext over 6728 files. The test results from their study are showcased in 5.4.

Total	Deleted	Recovered	Fully recovered	Intact metadata	Partial metadata
FAT	2894	1221	885	1118	2
NTFS	965	406	374	371	3
Ext	2869	583	372	1225	21
Totals	6728	2210	1631	2714	26

Table 5.4: Recovered File Analysis By File System taken from [85]

Looking at the result we can see that FAT and NTFS deleted files have more than a 40% chance of being recovered, respectively of 42.19% and 42.07%. Since FAT is the most widely used in removable devices, cameras, and memory cards and NTFS used in most Windows version since Windows XP we can somewhat expect to encounter usable files for investigations when the files were originated in a Windows machine. On the other hand, when the file system is Ext, mostly used in Linux distributions the value falls to 20,32%, meaning that it's twice more likely to recover a deleted file from a Windows user than from a Linux user.

Conversely, if the file system is damaged the likelihood of The Sleuth kit recovering a file greatly decreases, this is because it is difficult to order the fragments of what was the original file or in our case the picture. There exist however a specific algorithm that is designed to recover the JPEG files in these conditions and can guaranty 97% success rate [86].

5.7 Summary

In this Chapter, we have presented our methodology for testing our final solution.

We started by dividing our problems into two subproblems, firstly determining if the picture is NSFW and later determining the age of the person in the picture. As such we build distinct test scenarios that would test the different components individually and build up to a test that would be as close to a real-world scenario as we could get without using illegal or unethical datasets.

The results indicate that the solution would correctly identify a picture containing child pornography. Thus presenting a valid solution for the problem we set out to solve. However we also found that using a standard laptop for processing a disk image would require a lot of processing power and time, this is still more accurate, fast, and reliable than a human searching in the same data. A solution for this would be to use a more powerful and centralized solution, speeding up the processing and limiting the bottleneck to the upload of the image.

In the future, this solution still needs to be tested in a real-world investigation or using past investigation data containing an illicit picture, whenever the opportunity arises.

Chapter 6

Conclusion and Future Work

Everything has an end

Nicholas Lea

Digital forensic has evolved throughout recent years, however with the massification of mass storage and increase of computing power several new challenges have appeared. In short more computing power means it's more difficult to break and access encrypted traffic and store data and massification of mass storage leads to more data being needed to analyzed when a suspect is detained. As a result, we focused on speeding up the analysis of unencrypted disk images to locate illicit material and presenting them in a searchable way to the users.

In order to achieve our goal of automatically identify illicit content on the apprehended disk images obtained during the investigation, we divided the problem into two subproblems, identify the age of a person and if that picture is NSFW. We found that the state-of-the-art already had focused on solving these two problems and presented tested and accurate solutions. Additionally, there existed similar system architectures in forensics that could be of value to us. Hence, we studied these individual components that could also cope with the extraction aspect, i.e., wrapping the output of the chosen modules.

To properly test and validate the solution we had planned to test during an active investigation[1] and detective if we had made a meaningful impact in their day-to-day lives and speed up their search. However, this was not possible due to COVID-19 restrictions, because of this we had to test the solution with artificial datasets separating them in age and NSFW. The results are very promising and give the solution a good confidence level that it would work in a real investigation.

[1] This would be achieved by a cooperation between PJ and PDM

Future work

As future work we considered some directions, one of the objectives of this work was to make it as extensible as possible, thus we created a modular system with an integrated pipeline, this pipeline can be extended to add further forensic modules. The first venue would be to search for emails, search history, and Facebook activity, there is already an example of such a tool developed in python by Simson L. Garfinkel[87], we would only need to wrap the output of the text file generated and sent them to ElasticSearch. We did not add this to the final build because it would not contribute to solving the problem we focused on in this thesis.

A second venue to pursue would be to include WhereML in our pipeline. WhereML is a CNN by J. Randall Hunt that can predict the location of a picture accurately. The trained model and configuration details for Amazon web services are available making the integration with our pipeline simple has adding another docker image and using an additional API. There is also a Twitter bot that can be used for demonstrating the accuracy and precision of the prediction, the bot respond to each twit where it is identified with 3 location it believes the picture was taken from.

Concerning continually adding more features, a slowdown is inevitable, and another problem will arise, allocating the necessary computing power in extracting irrelevant features for a specific scenario. Hence a feature extraction pipeline that is customizable for each use would be an interesting approach to be explored.

Additionally, one of the topics we discussed in Chapter 1 was digital forensics has a service, the solution described can be made available through the use of a public cloud like Amazon web services or Azure as a paid service for companies. The main modifications would be the necessity of having a domain and due to the ephemeral nature of containers configuring persistence storage for the use of a volume for saving the data.

Finally, in a post-pandemic world, this solution should be tested and validated in the field through close cooperation with police forces.

Bibliography

[1] *Becoming A Data-Driven CEO*. URL: https://www.domo.com/solution/data-never-sleeps-6.

[2] W. Alink, R. Bhoedjang, P. Boncz, and A. de Vries. "XIRAF – XML-based indexing and querying for digital forensics." In: *Digital Investigation* 3 (2006). The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06), pp. 50 –58. ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2006.06.016. URL: http://www.sciencedirect.com/science/article/pii/S1742287606000776.

[3] M. van Justitie en Veiligheid. *Hansken*. May 2017. URL: https://www.forensicinstitute.nl/products-and-services/forensic-products/hansken.

[4] R. van Baar, H. van Beek, and E. van Eijk. "Digital Forensics as a Service: A game changer." In: *Digital Investigation* 11 (2014). Proceedings of the First Annual DFRWS Europe, S54 –S62. ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2014.03.007. URL: http://www.sciencedirect.com/science/article/pii/S1742287614000127.

[5] H. van Beek, E. van Eijk, R. van Baar, M. Ugen, J. Bodde, and A. Siemelink. "Digital forensics as a service: Game on." In: *Digital Investigation* 15 (2015). Special Issue: Big Data and Intelligent Data Analysis, pp. 20 –38. ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2015.07.004. URL: http://www.sciencedirect.com/science/article/pii/S1742287615000857.

[6] B. Carrier. *Open Source Digital Forensics Tools: The Legal Argument*. 2002.

[7] C. Wood. *Computer security : a comprehensive controls checklist*. New York: Wiley, 1987. ISBN: 047184795X.

[8] C. of Virginia Joint Commission on Technology and Science. *Regional Computer Forensic Laboratory (RCFL) National Program Office (NPO)*. 2004. URL: http://dls.virginia.gov/commission/pdf/FBI-RCFL.pdf.

[9] S. L. Garfinkel. "Digital forensics research: The next 10 years." In: *Digital Investigation* 7 (2010). The Proceedings of the Tenth Annual DFRWS Conference, S64 –S73. ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2010.05.009. URL: http://www.sciencedirect.com/science/article/pii/S1742287610000368.

[10] H. D. E. Shelton. *The 'CSI Effect': Does It Really Exist?* 2008. URL: https://nij.ojp.gov/topics/articles/csi-effect-does-it-really-exist.

[11] E. Casey and G. J. Stellatos. "The Impact of Full Disk Encryption on Digital Forensics." In: *SIGOPS Oper. Syst. Rev.* 42.3 (Apr. 2008), 93–98. ISSN: 0163-5980. DOI: 10.1145/1368506.1368519. URL: https://doi.org/10.1145/1368506.1368519.

[12] P. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and J. Teubner. "MonetDB/XQuery: A Fast XQuery Processor Powered by a Relational Engine." In: *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*. SIGMOD '06. Chicago, IL, USA: Association for Computing Machinery, 2006, 479–490. ISBN: 1595934340. DOI: 10.1145/1142473.1142527. URL: https://doi.org/10.1145/1142473.1142527.

[13] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. "ZooKeeper: Wait-Free Coordination for Internet-Scale Systems." In: *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*. USENIXATC'10. Boston, MA: USENIX Association, 2010, p. 11.

[14] EsotericSoftware. *Kryo*. Sept. 2009. URL: https://github.com/EsotericSoftware/kryo.

[15] A. Lakshman and P. Malik. "Cassandra: A Decentralized Structured Storage System." In: *SIGOPS Oper. Syst. Rev.* 44.2 (Apr. 2010), 35–40. ISSN: 0163-5980. DOI: 10.1145/1773912.1773922. URL: https://doi.org/10.1145/1773912.1773922.

[16] H. Akdogan. *ElasticSearch Indexing*. Packt Publishing, 2015. ISBN: 1783987022.

[17] H. Patel. "HBase: A NoSQL Database." In: May 2017. DOI: 10.13140/RG.2.2.22974.28480.

[18] J. Kävrestad. *Fundamentals of Digital Forensics: Theory, Methods, and Real-Life Applications*. Springer, 2018. ISBN: 331996318X. URL: https://www.amazon.com/Fundamentals-Digital-Forensics-Real-Life-Applications/dp/331996318X?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=331996318X.

[19] B. Saltaformaggio, R. Bhatia, Z. Gu, X. Zhang, and D. Xu. "GUITAR: Piecing Together Android App GUIs from Memory Images." In: *ACM Conference on Computer and Communications Security*. Ed. by I. Ray, N. Li, and C. Kruegel. ACM, 2015, pp. 120–132. ISBN: 978-1-4503-3832-5. URL: http://dblp.uni-trier.de/db/conf/ccs/ccs2015.html#SaltaformaggioB15.

[20] K. Woods, C. A. Lee, and S. Garfinkel. "Extending Digital Repository Architectures to Support Disk Image Preservation and Access." In: *Proceedings of the 11th Annual International ACM/IEEE Joint Conference on Digital Libraries*. JCDL '11. Ottawa, Ontario, Canada: Association for Computing Machinery, 2011, 57–66. ISBN:

9781450307444. DOI: 10.1145/1998076.1998088. URL: https://doi.org/10.1145/1998076.1998088.

[21] B. Saltaformaggio, R. Bhatia, Z. Gu, X. Zhang, and D. Xu. "VCR: App-Agnostic Recovery of Photographic Evidence from Android Device Memory Images." In: *CCS '15*. 2015.

[22] B. Carrier. *File System Forensic Analysis*. Addison-Wesley Professional, Mar. 2005. ISBN: 0321268172. URL: https://www.xarg.org/ref/a/0321268172/.

[23] *Linux NTFS driver*. 2004. URL: http://linux-ntfs.sourceforge.net/.

[24] *Documentation Archive*. URL: https://developer.apple.com/library/archive/navigation/index.html#topic=TechnicalNotesÂğion=ResourceTypes.

[25] *HFS Plus Volume Format*. Mar. 2004. URL: https://developer.apple.com/library/archive/technotes/tn/tn1150.html#CoreConcepts.

[26] D. Comer. "Ubiquitous B-Tree." In: *ACM Comput. Surv.* 11.2 (June 1979), 121–137. ISSN: 0360-0300. DOI: 10.1145/356770.356776. URL: https://doi.org/10.1145/356770.356776.

[27] K. D. Fairbanks, C. P. Lee, and H. L. Owen. "Forensic Implications of Ext4." In: *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*. CSIIRW '10. Oak Ridge, Tennessee, USA: Association for Computing Machinery, 2010. ISBN: 9781450300179. DOI: 10.1145/1852666.1852691. URL: https://doi.org/10.1145/1852666.1852691.

[28] T. Ts'o and S. Tweedie. "Planned Extensions to the Linux Ext2/Ext3 Filesystem." In: Jan. 2002, pp. 235–243.

[29] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier. "The new ext4 filesystem: current status and future plans." In: *Proceedings of the Linux Symposium*. Vol. 2. 2007, pp. 21–33.

[30] F. Hafeez. "Role of File System in Operating System." In: (June 2016). DOI: 10.13140/RG.2.1.4081.8169.

[31] *Microsoft FAT Specification*. Aug. 2005. URL: http://read.pudn.com/downloads77/ebook/294884/FAT32Spec(SDAContribution).pdf.

[32] *Filesystem in Userspace*. Feb. 2005. URL: https://sourceforge.net/p/fuse/mailman/fuse-devel/.

[33] A. Rajgarhia and A. Gehani. "Performance and Extension of User Space File Systems." In: *Proceedings of the 2010 ACM Symposium on Applied Computing*. SAC '10. Sierre, Switzerland: Association for Computing Machinery, 2010, 206–213. ISBN: 9781605586397. DOI: 10.1145/1774088.1774130. URL: https://doi.org/10.1145/1774088.1774130.

[34] B. K. R. Vangoor, V. Tarasov, and E. Zadok. "To FUSE or Not to FUSE: Performance of User-Space File Systems." In: *Proceedings of the 15th Usenix Conference on File and Storage Technologies*. FAST'17. Santa clara, CA, USA: USENIX Association, 2017, 59–72. ISBN: 9781931971362.

[35] Y. Zhu, T. Wang, K. Mohror, A. Moody, K. Sato, M. Khan, and W. Yu. "Direct-FUSE: Removing the Middleman for High-Performance FUSE File System Support." In: *Proceedings of the 8th International Workshop on Runtime and Operating Systems for Supercomputers*. ROSS'18. Tempe, AZ, USA: Association for Computing Machinery, 2018. ISBN: 9781450358644. DOI: 10.1145/3217189.3217195. URL: https://doi.org/10.1145/3217189.3217195.

[36] B. Carrier. *The Sleuth Kit (TSK) Library User's Guide and API Reference*. URL: http://www.sleuthkit.org/sleuthkit/docs/api-docs/4.3/index.html.

[37] E. Alpaydin. *Introduction to Machine Learning (Adaptive Computation and Machine Learning series)*. The MIT Press, Aug. 2014. ISBN: 9780262028189. URL: https://www.xarg.org/ref/a/0262028182/.

[38] *Unsupervised Learning: Foundations of Neural Computation (Computational Neuroscience)*. A Bradford Book, June 1999. ISBN: 026258168X. URL: https://www.xarg.org/ref/a/026258168X/.

[39] S. Russell. *Artificial Intelligence: A Modern Approach (3rd Edition)*. Pearson, Dec. 2009. ISBN: 0136042597. URL: https://www.xarg.org/ref/a/0136042597/.

[40] M. Mohri. *Foundations of Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, Aug. 2012. ISBN: 026201825X. URL: https://www.xarg.org/ref/a/026201825X/.

[41] L. P. Kaelbling, M. L. Littman, and A. W. Moore. "Reinforcement Learning: A Survey." In: *Journal of Artificial Intelligence Research* 4 (May 1996), pp. 237–285. DOI: 10.1613/jair.301. URL: https://doi.org/10.1613/jair.301.

[42] F. Mayer and M. Steinebach. "Forensic Image Inspection Assisted by Deep Learning." In: *Proceedings of the 12th International Conference on Availability, Reliability and Security - ARES '17*. ACM Press, 2017. DOI: 10.1145/3098954.3104051. URL: https://doi.org/10.1145/3098954.3104051.

[43] C. Santos, E. Souto, and E. Santos. "Nudity detection based on image zoning." In: July 2012. DOI: 10.1109/ISSPA.2012.6310454.

[44] C. Platzer, M. Stuetz, and M. Lindorfer. "Skin Sheriff: A Machine Learning Solution for Detecting Explicit Images." In: June 2014. DOI: 10.1145/2598918.2598920.

[45] A. Ulges and A. Stahl. "Automatic Detection of Child Pornography Using Color Visual Words." In: *Proceedings of the 2011 IEEE International Conference on Multimedia and Expo*. ICME '11. USA: IEEE Computer Society, 2011, 1–6. ISBN: 9781612843483. DOI: 10.1109/ICME.2011.6011977. URL: https://doi.org/10.1109/ICME.2011.6011977.

[46] L. Demajo, K. Guillaumier, and G. Azzopardi. "Age group recognition from face images using a fusion of CNN- and COSFIRE-based features." In: Jan. 2019, pp. 1–6. DOI: 10.1145/3309772.3309784.

[47] M. Kaur, R. K. Garg, and S. Singla. "Analysis of facial soft tissue changes with aging and their effects on facial morphology: A forensic perspective." In: *Egyptian Journal of Forensic Sciences* 5.2 (2015), pp. 46 –56. ISSN: 2090-536X. DOI: https://doi.org/10.1016/j.ejfs.2014.07.006. URL: http://www.sciencedirect.com/science/article/pii/S2090536X14000501.

[48] H. Han, C. Otto, and A. K. Jain. "Age estimation from face images: Human vs. machine performance." In: *2013 International Conference on Biometrics (ICB)*. June 2013, pp. 1–8. DOI: 10.1109/ICB.2013.6613022.

[49] R. Angulu, J. R. Tapamo, and A. O. Adewumi. "Age estimation via face images: a survey." In: *EURASIP Journal on Image and Video Processing* 2018.1 (June 2018). DOI: 10.1186/s13640-018-0278-6. URL: https://doi.org/10.1186/s13640-018-0278-6.

[50] G. Guo, Y. Fu, C. R. Dyer, and T. S. Huang. "Image-Based Human Age Estimation by Manifold Learning and Locally Adjusted Robust Regression." In: *IEEE Transactions on Image Processing* 17.7 (July 2008), pp. 1178–1188. ISSN: 1941-0042. DOI: 10.1109/TIP.2008.924280.

[51] E. Eidinger, R. Enbar, and T. Hassner. "Age and Gender Estimation of Unfiltered Faces." In: *IEEE Transactions on Information Forensics and Security* 9.12 (Dec. 2014), pp. 2170–2179. ISSN: 1556-6021. DOI: 10.1109/TIFS.2014.2359646.

[52] A. Deepa and T. Sasipraba. "Age estimation in facial images using histogram equalization." In: *2016 Eighth International Conference on Advanced Computing (ICoAC)*. Jan. 2017, pp. 186–190. DOI: 10.1109/ICoAC.2017.7951767.

[53] Yahoo. *Yahoo Open NSFW*. Nov. 2018. URL: https://github.com/yahoo/open_nsfw.

[54] Clarifai. *Enterprise AI Powered Computer Vision Solutions*. URL: https://www.clarifai.com/.

[55] Hhatto. *Nude.py*. June 2019. URL: https://github.com/hhatto/nude.py.

[56] *NudityDetectioni2v - Algorithm by sfw*. URL: https://algorithmia.com/algorithms/sfw/NudityDetectioni2v.

[57] M. Saito and Y. Matsui. "Illustration2Vec: A Semantic Vector Representation of Illustrations." In: *SIGGRAPH Asia 2015 Technical Briefs*. SA '15. Kobe, Japan: Association for Computing Machinery, 2015. ISBN: 9781450339308. DOI: 10.1145/2820903.2820907. URL: https://doi.org/10.1145/2820903.2820907.

[58] T. Fawcett. "An introduction to ROC analysis." In: *Pattern Recognition Letters* 27.8 (June 2006), pp. 861–874. DOI: 10.1016/j.patrec.2005.10.010. URL: https://doi.org/10.1016/j.patrec.2005.10.010.

[59] Z. Qawaqneh, A. A. Mallouh, and B. D. Barkana. "Deep convolutional neural network for age estimation based on VGG-face model." In: *arXiv preprint arXiv:1709.01664* (2017).

[60] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going deeper with convolutions." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[61] R. Rothe, R. Timofte, and L. V. Gool. "Deep expectation of real and apparent age from a single image without facial landmarks." In: *International Journal of Computer Vision (IJCV)* (July 2016).

[62] K. Simonyan and A. Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. arXiv: 1409.1556 [cs.CV].

[63] M. Mathias, R. Benenson, M. Pedersoli, and L. V. Gool. "Face Detection without Bells and Whistles." In: *ECCV*. 2014.

[64] P. Weiner. "Linear Pattern Matching Algorithm." In: Nov. 1973, pp. 1–11. DOI: 10.1109/SWAT.1973.13.

[65] K. C. Tan. "On Foster's Information Storage and Retrieval Using AVL Trees." In: *Commun. ACM* 15.9 (Sept. 1972), p. 843. ISSN: 0001-0782. DOI: 10.1145/361573.361588. URL: https://doi.org/10.1145/361573.361588.

[66] G. Lavee, R. Lempel, E. Liberty, and O. Somekh. "Inverted index compression via online document routing." In: *Proceedings of the 20th international conference on World wide web - WWW '11*. ACM Press, 2011. DOI: 10.1145/1963405.1963475. URL: https://doi.org/10.1145/1963405.1963475.

[67] V. W. Setzer and R. Lapyda. "Design of Data Models for the ADABAS System Using the Entity-Relationship Approach." In: *Proceedings of the Second International Conference on the Entity-Relationship Approach to Information Modeling and Analysis*. ER '81. NLD: North-Holland Publishing Co., 1981, 319–336. ISBN: 0444867473.

[68] P. J. Pratt. *Database systems: Management and design*. Boyd & Fraser Pub. Co, 1987. ISBN: 0878352279. URL: https://www.amazon.com/Database-systems-Management-Philip-Pratt/dp/0878352279?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0878352279.

[69] D. Merrit. "Volume Testing of Statistical Database Using Model 204 DBMS." In: *Proceedings of the 1st LBL Workshop on Statistical Database Management*. SSDBM'81. Berkeley, USA: Lawrence Berkeley Laboratory, 1981, 380–389. ISBN: 155555222X.

[70] S. Huston, A. Moffat, and W. B. Croft. "Efficient Indexing of Repeated N-Grams." In: *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*. WSDM '11. Hong Kong, China: Association for Computing Machinery, 2011, 127–136. ISBN: 9781450304931. DOI: 10.1145/1935826.1935857. URL: https://doi.org/10.1145/1935826.1935857.

[71] W. Stallings and L. Brown. *Computer Security*. Feb. 2021. ISBN: 9780134794105.

[72] J. S. Plank. "A tutorial on Reed–Solomon coding for fault-tolerance in RAID-like systems." In: *Software: Practice and Experience* 27.9 (1997), pp. 995–1012.

[73] "On the Role of Scientific Thought." In: *Selected Writings on Computing: A Personal Perspective*. Berlin, Heidelberg: Springer-Verlag. ISBN: 0387906525.

[74] jpetazzo. *Using Docker-in-Docker for your CI or testing environment Think twice*. 2020. URL: https://jpetazzo.github.io/2015/09/03/do-not-use-docker-in-docker-for-ci/.

[75] s. GlobalStats. *Desktop vs Mobile vs Tablet Market Share Europe*. 2020. URL: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/europe.

[76] *OpenAPI Specification*. URL: https://swagger.io/specification/.

[77] Forensicsware. *E01 (Encase Image File Format)*. URL: https://www.forensicsware.com/blog/e01-file-format.html.

[78] M. S. Turan, E. Barker, W. Burr, and L. Chen. "Recommendation for password-based key derivation." In: *NIST special publication* 800 (2010), p. 132.

[79] D. A. Patterson, G. Gibson, and R. H. Katz. "A case for redundant arrays of inexpensive disks (RAID)." In: *Proceedings of the 1988 ACM SIGMOD international conference on Management of data*. 1988, pp. 109–116.

[80] N. I. of Standards and Technology. *Forensic Images for File Carving*. URL: https://www.cfreds.nist.gov/FileCarving/index.html.

[81] A. Kim. *NSFW Data Scraper*. URL: https://github.com/alex000kim/nsfw_data_scraper.

[82] C. Alessio. *Animals-10*. 2019. URL: https://www.kaggle.com/alessiocorrado99/animals10.

[83] J. Cheng, Y. Li, J. Wang, L. Yu, and S. Wang. "Exploiting effective facial patches for robust gender recognition." In: *Tsinghua Science and Technology* 24.3 (2019), pp. 333–345.

[84] V. LoBue and C. Thrasher. "The Child Affective Facial Expression (CAFE) set: Validity and reliability from untrained adults." In: *Frontiers in psychology* 5 (2015), p. 1532.

[85] H. Security. *The Sleuth Kit*. 2014. URL: https://www.dhs.gov/sites/default/files/publications/508_TestReport_TheSleuthKit322-Autopsy224TestReport_November2015_Final.pdf.

[86] Y. Tang, J. Fang, K. Chow, S. Yiu, J. Xu, B. Feng, Q. Li, and Q. Han. "Recovery of heavily fragmented JPEG files." In: *Digital Investigation* 18 (2016), S108 –S117. ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2016.04.016. URL: http://www.sciencedirect.com/science/article/pii/S1742287616300512.

[87] S. L. Garfinkel. "Digital media triage with bulk data analysis and bulk_extractor." In: *Computers & Security* 32 (Feb. 2013), pp. 56–72. DOI: 10.1016/j.cose.2012.09.011. URL: https://doi.org/10.1016/j.cose.2012.09.011.