Facial Recognition Prevention

Face web scraping

Alexandre De Groodt

Research and Development project owner:
Prof. Thibault Debatty

Master thesis submitted under the supervision of
Prof. Thibault Debatty

the co-supervision of
Dr. Charles Beumier in order to be awarded the

Degree of
Master in Cybersecurity
Corporate Strategies

Academic year
2024 - 2025

I hereby confirm that this thesis was written independently by myself without the use of any sources beyond those cited, and all passages and ideas taken from other sources are cited accordingly.

02/06/2025

Title: Facial recognition prevention

Author: Alexandre De Groodt
Master in Cybersecurity – Corporate Strategies
Academic year: 2024 - 2025

Abstract

Facial recognition technology has become a central issue in the debate over digital privacy. While it provides powerful tools for security and law enforcement, it also poses serious risks of intrusion into individuals' private lives. With the increasing availability of facial recognition algorithms and the vast amount of imagery available online, the boundary between personal and professional spheres is increasingly vulnerable. This thesis presents an open-source Python script intended for government and companies. The script combines web scraping with facial recognition to assess the online presence of selected individuals based on reference images, and notify them. The tool is designed to operate within legal and ethical boundaries, focusing exclusively on publicly accessible content. It may also be combined with mechanisms such as image takedown requests or obfuscation tools like Fawkes to reduce digital traceability. In addition to the technical implementation, this work explores the legal landscape and societal implications of facial recognition, discussing both its legitimate applications and potential for misuse. Finally, the thesis examines available countermeasures and highlights how tools like this can be used responsibly to raise awareness about digital exposure and promote personal privacy.

Keywords: facial recognition, prevention, privacy, AI, scraping, smart cities

Preface

This master thesis was a wonderful opportunity for me to learn more on the academic world independantly, and a way to bring attention to the subject of privacy that concerns us all.

This is a very interesting subject in the context of cybersecurity, in fact the reason I joined the field was to protect people from the risks they face in this expanding digital world, and I found myself informing many on this very actual subject.

I myself did not realize the extent of this technology and its significant presence, even as a cybersecurity student.

I am unsure whether I should be impressed or scared by these impressive advances in facial recognition.

Finally I'm able to learn to be more professional in my work, to make it easier to collaborate with others in the future, and get more hands on experience in coding, and put my cybersecurity and computer science background in practise in a practical situation. I can only hope this contribution will be useful.

On a more personal note, this thesis was an amazing journey. I started intrigued by the subject, learning about it slowly and doing tests to figure out where I was going.

Then I started talking more about it, getting a clearer picture of it in my head. I had to write it, so I did, but reading the articles was no longer such a daring task. I read them, wrote down my comprehension and used AI to make it clearer by section, so I had a solid base to work on again and improve.

Eventually I became passionate about it, when the coding phase really kicked in. I remembered how fun it is to code, and even writing was in a state of flow, fully absorbed. I let quite a few eggs overcook because of this, but it was worth it.

I now have a clearer vision on the state of the world in terms of AI and facial recognition and how this thesis can hopefully contribute.

Acknowledgements

I thank my supervisor for the help and direction provided as well as my promoter for the opportunity to work on this project and direction in completing it. Great thank comes to my family members for supporting me in the making of this thesis. I also thank the template authors to facilitate the thesis by providing a basis for it.

I have used chatgpt solely for reformatting purpose in order to write in a more structured English, and used that as a new base to rewrite upon. I believe this is an appropriate use of the tool, since it was made to recognize and predict speach patterns, making it more than able to rephrase content in a more structured and formal format. It also proved useful in coding to suggest small code segments and fix bugs.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
API	Application Programing Interface
CNN	Convolutional Neural Network
DSA	Digital Services Act
GAN	Generative Adversarial Network
GDPR	General Data Protection Regulation
NIST	National Institute of Standards and Technology
openCV	Open Source Computer Vision Library

Chapter 1
Introduction

Facial recognition is a rapidly evolving technology at the intersection of biometrics and machine learning. While it promises convenience and innovation - such as seamless device authentication and security - it also raises significant ethical, legal, and societal concerns.

Deployment One concerning development is the emergence of public-facing tools like **Pimeyes** and **Facesearch**, which combine facial recognition with web scraping. These platforms allow anyone to trace an individual's online presence using only a single photo, effectively linking public and private personas without consent. [22] Such capabilities represent a growing privacy risk in a world where images are shared ubiquitously.

Meanwhile, facial recognition is also being deployed at scale by private companies and governments. Firms like **Clearview AI** offer advanced tools capable of identifying individuals in real time across vast image databases. The acceleration of these technologies, fueled in part by global events such as the COVID-19 pandemic and heightened security concerns, has outpaced existing legal safeguards that now have to decide how to respond to it [37, 44].

Social Media. One of the most critical concerns in the digital age is the extensive presence of individuals on social media platforms. These platforms often contain a wealth of personal information, frequently shared with limited awareness of the associated privacy implications. As such, social media represents not only a valuable source for information gathering but also an important source of privacy breach. Even if we ourselves do not post directly on social medias, we may be on other's pictures, and considering that "social media users share at least 3.5 billion images everyday", this is not a risk we can ignore. [18]

Legal aspect In this context, this thesis explores the technical and legal feasibility of building a facial recognition-based search system within the bounds of European data protection law. The goal is to create a free and open-source framework that allows users to assess their own exposure online. Such a tool could support takedown or obfuscation strategies using technologies like Fawkes, which make the face unrecognizable to machines whilst keeping it roughly the same for humans. Fawkes is a promising means to help individuals defend against unauthorized facial recognition by modifying the images for IA agents but not for the human eye [49].

1.1 Motivations

Facial recognition technology is advancing rapidly, but public understanding, legal frameworks, and technical countermeasures have not kept pace in most countries. While companies and state actors increasingly deploy these systems for surveillance and commercial purposes, individuals and civil society are left without effective tools to assess or resist such tracking. `robot.txt` files to enforce the terms of service can be ignored, and defences in place against web scraping can be avoided, such as was the case with youtube. [**?**]

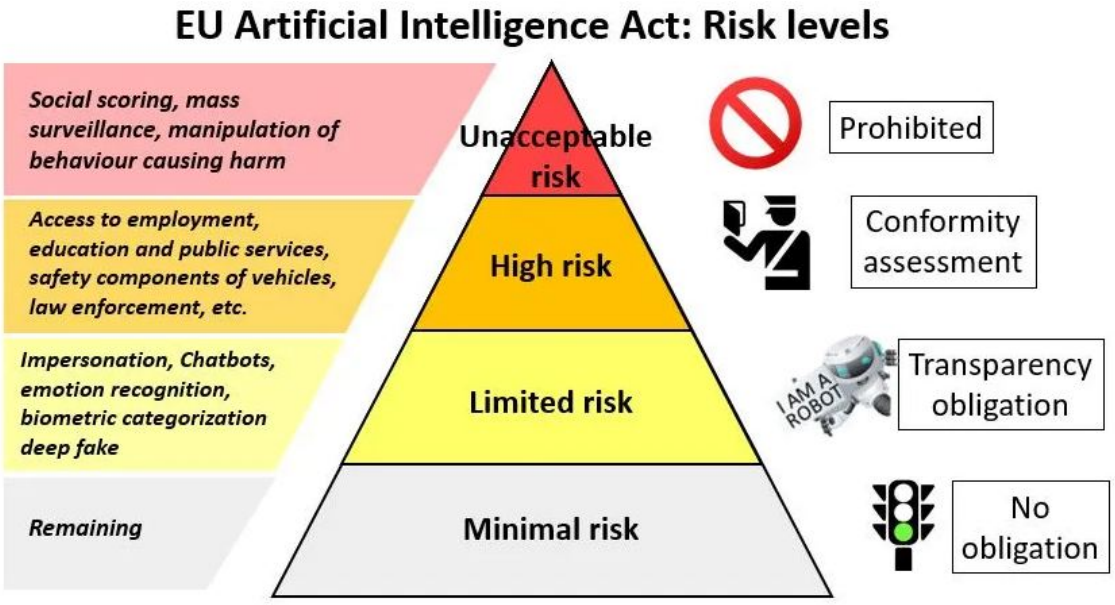


Figure 1.1: AI act overview [47]

→ This asymmetry creates a significant privacy threat. A single photo can now enable the identification and tracing of individuals across the web without their consent, through systems powered by massive datasets and machine learning. The fact that even small, widely available camera devices can now perform real time facial recognition using micro controllers highlights the urgency of addressing this issue, given they can be used to identify us.

Despite growing regulatory efforts such as the **GDPR** and **AI act** in Europe, enforcement is often behind when compared to the speed at which private companies innovate. Public-facing tools are costly, offering little transparency to the average person, who often ignore their very existence, and their right to request takedowns.

Awareness This thesis aims to raise awareness of these challenges and contribute a practical prototype for facial recognition-based image discovery, still for use for resourceful individuals, but mostly for internal use by companies and governments who can raise awareness. This means no longer limiting the use of such tools to the few individuals that are aware enough of the risk linked with their online presence. The goal is not only to make these mechanisms understandable and visible, but also to empower users to take action - whether by requesting image removal or applying obfuscation techniques like Fawkes.

Understanding the inner workings of these algorithms is key. Only by exposing how facial recognition functions, and where it fails, can we have meaningful discussions about its regulation, ethical use, and role in society. Our work seeks to bridge the gap between technical capability and individual rights - especially those at high risk - laying the foundation for more awareness and better safeguards.

1.1.1 Context

Facial recognition technologies are rapidly being deployed across the globe, raising critical questions about privacy, consent, and surveillance.

- In **China**, these systems are deeply embedded in public infrastructure, enabling real-time tracking and social monitoring of citizens by identifying them in the many camera systems, and linking all this information together, and keeping track of people on watch lists. This "sharp eyes" program was put in place in 2015. [11]

- In the **United Kingdom**, law enforcement has trialed facial recognition vans in public spaces, prompting public backlash over issues of algorithmic bias and lack of transparency. [4]

- In the **United States**, the NYPD and other police departments accross the states have adopted facial recognition systems for crime prevention with Clearview since 2019 [55], while some states like Ohio have begun to push back, due to cases of miss-identifications, and multiple companies have chosen to stop selling facial recognition tools to police departments [50].

- The Royal Canadian Mounted Police in **Canada** has been using facial recognition to help in their investigations, with Clearview [37].

- Even in Serbia, in **Europe**, security cameras equipped with facial recognition and bought from China have been installed. [46] The watch lists can be consequent, and we do not know who is being watched. For instance, whilst this technology is banned for mass surveillance, in France it can be used by the police in order to find individuals from their databases. [1]

Thankfully laws and comitees exist to keep this in check. Facial recognition is not inevitable, but it is certainly increasing in presence around the globe, in "smart cities".

Societal context The convergence of ubiquitous cameras and powerful machine learning algorithms has made continuous population monitoring a real possibility. This capability is no longer limited to state surveillance; it extends to what is often referred to as surveillance capitalism, where personal data is harvested, analyzed, and monetized by private companies for targeted advertising and behavioral profiling, learning the preferences of everyone so that they can be sold products more effectively. Hence data is worth money, meaning there is a risk of facial recognition being used for profiling.

Regulations such as the European Union's **General Data Protection Regulation** (GDPR) offer some degree of protection, but they were not designed with the full implications of facial recognition in mind. Traditional legal concepts like the "right to image" are difficult to enforce in a context with facial-enabled surveillance cameras, where we have to distinguish between the will to keep citizens safe by monitoring with protecting their private lives. Thankfully a new comitee has been formed in 2024, and a new **AI act** was voted in june of that year, notably banning "untargeted scraping of the internet and closed-circuit television (cameras) material to create or expand facial recognition databases". Europe is ahead in this regard, although laws always take 2 years before being enforced [17, 20].

Tools like **Pimeyes** and **Facesearch** demonstrate the accessibility and power of this technology. With just a single photo, users can discover where a person appears across the web. Despite nominal opt-out mechanisms, the barrier to entry for building similar facial recognition and scraping systems remains low, thanks to open-source libraries such as **OpenCV** and widely available online tutorials. This democratization of surveillance

tools creates significant risk and highlights the need for public awareness, legal reform, and technical countermeasures.

1.2 Project Statement & Contributions

1.2.1 Problem Statement

Despite regulatory efforts such as the **GDPR** and emerging discussions about banning facial recognition surveillance (e.g., in Canada [37]), enforcement often lags behind technological progress, although Europe is ahead with the **AI act** mentionned before. This gap is especially evident in the case of automated facial recognition systems that scrape and analyze web-based images, frequently operating in legal grey areas—as shown by Clearview AI, which continues to function despite cease-and-desist orders from major platforms like Google [8].

New threats With off-the-shelf tools and minimal technical skills, anyone can build systems to collect facial data from public sources and analyze it at scale, or use existing tools to do so. These capabilities pose serious privacy risks, as individuals can be tracked and profiled online without their knowledge or consent. Furthermore, said individuals can lack knowledge of their rights to ask the pictures to be removed, of the privacy risks they pose, or even of the picture's very presence, and the existence of tools to find it. This is where a tool with transparent code aimed at companies can make sense. This is especially true for important companies like banks, military organisation,or any other company or organization operating in a strategic domain, where employee profiling in order to find their whereabouts to deceive or impersonate them is most troublesome, potentially breaching the gap between private and public life. **Online presence** If we want protection, we may

try to control our online presence, but unfortunately sites like clearview keep a local copy of all pictures, meaning that once a picture has been online, it is already too late, and modifying or removing it would not be effective. If we instead wish to use the online facial recognition tools to protect ourselves by checking our presence on their website, we need to trust that these companies are not going to keep the images, as claimed, and will also have to spend a significant sum in order to get access to advanced searches, on an individual level.

Project This project investigates the feasibility of creating a local facial search system while staying within European legal frameworks. The goal is to identify publicly available images of individuals, especially on social medias but on the web as a whole, with the intent to be aware of this presence, and having the option to either remove them or make them unrecognisable to facial recognition systems. Finally, we aim to highlight the risks of uncontrolled facial data exposure and underscore the need for stronger regulatory and technical safeguards.

1.2.2 Contributions

This thesis makes both conceptual and practical contributions:

- **Insight:** We present the concepts in a concise way to give the necessary context to understand facial recognition, the code and reasoning behind it.

- **Prototype Tools:** We present a suite of scripts for image gathering and face detection, demonstrating the feasibility of building a custom recognition pipeline using public data and tools, usable for companies, state actors and individuals.
- **Legal and Ethical Framing:** We contextualize our technical findings within existing European laws (e.g., **GDPR**), emphasizing the limitations of current privacy protections.
- **Awareness and Empowerment:** We explore methods for defending against recognition, such as image obfuscation, and discuss how individuals and institutions can audit their online exposure.

Due to the terms of service of the various social media platforms preventing scraping, the project does not include a complete end-to-end implementation of public search, but we detail the system components and their interactions for transparency and reproducibility, as well as future work.

1.3 Organization of this Document

- **Section 1 – Introduction:** The reasoning behind the thesis was presented, and keys notions were mentionned.
- **Section 2 – Background & Technical Foundations:** An overview of facial recognition algorithms and web scraping techniques and tools, along with their legal and ethical implications.
- **Section 3 – State of the Art:** A review of the existing literature on facial recognition from a technical as well as a legal perspective.
- **Section 4 – Planning:** Presentation of the code design, legal constraints, and prototype functionality.
- **Section 5 – Implementation:** Explanation of the code and testing.
- **Section 6 – Conclusion:** An overview as well as a reflection on the risks this can cause, and direction for future work in this matter.

Chapter 2
Background

This section introduces the technical and contextual knowledge necessary to understand the rest of the thesis, into which we will dive in greater details in the state of the art. As these are not very advanced notions by themselves, this section will consist more in general internet research rather than research papers. This should serve as a refresher into these notions, and make the thesis more accessible.

We concisely cover facial recognition algorithms, as well the web scraping process that powers large-scale image harvesting, and current legal protections. These topics form the foundation for evaluating the feasibility and ethical implications of our proposed system.

2.0.1 Facial Recognition

Facial recognition systems attempt to identify individuals by analyzing their facial features. These systems rely on the assumption that every face is unique and can serve as a biometric identifier.

Biometrics

Biometric systems authenticate individuals based on inherent physical or behavioral characteristics — "what we are" — rather than "what we know" (passwords) or "what we have" (tokens). Fingerprint scanners are a common example; facial recognition is increasingly used in consumer electronics for device unlocking. [54].

The other aspect is that the biometric data must be stored safely, thankfully there are methods such as homographic encryption that allow operations to be performed on encrypted data, such as verification. [2] However, identity verification is not the focus of this thesis.

Machine Learning

Modern facial recognition systems rely heavily on deep learning, particularly Convolutional Neural Networks (CNNs), which are capable of learning and generalizing complex visual patterns, using large datasets. Generative Adversarial Networks (GANs) also play a role in augmenting training datasets or generating realistic synthetic faces [34].

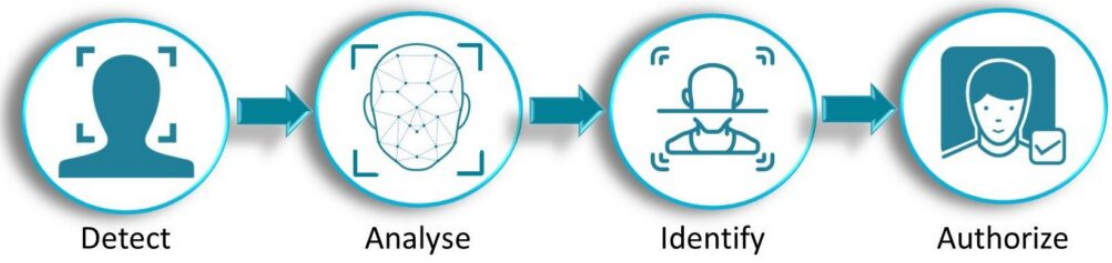


Figure 2.1: Facial recognition pipeline [38]

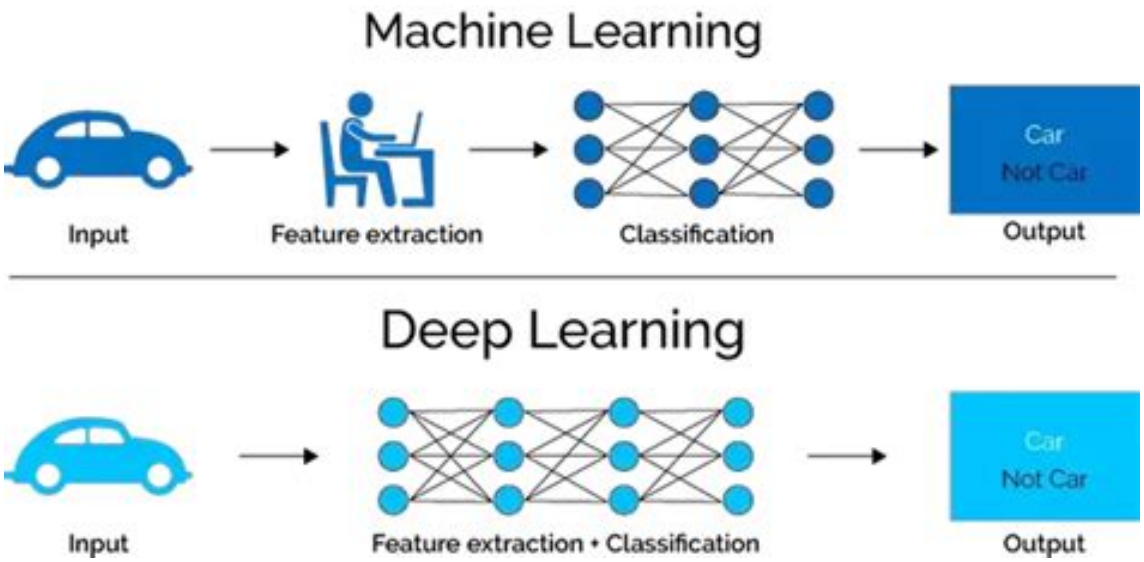


Figure 2.2: Neural network architecture used in recognition [7]

Deep learning models are trained by reinforcing connections that yield correct predictions during the learning phase. Once trained, these models can be reused to detect and match new faces, often by comparing feature vectors and calculating similarity scores.

However, deep learning is probabilistic in nature - false positives and false negatives can occur, sometimes with serious consequences in law enforcement, and even in shops where the technology was installed against potentially thieves. Although systems like Clearview claim near-perfect performance [30], they should never be used in isolation for legal decision-making.

Algorithmic Workflow

Typical facial recognition workflows involve:

1. **Detection:** Locating face regions within images.
2. **Feature Extraction:** Encoding facial features into vectors.
3. **Matching:** Comparing the query vector with a database of known identities.

Different systems may use different models for detection and recognition, optimizing each for speed, accuracy, or resource constraints [40].

2.0.2 Web Scraping

Web scraping involves the automated retrieval of online content using scripts or software. While scraping text is relatively common, gathering images and videos presents challenges due to bandwidth, data volume, and legal restrictions.

Major social platforms and other websites often prohibit scraping in their terms of service and provide APIs as controlled alternatives. However, these APIs are intended for advertising and simple apps, and they often do not give access to images. This makes scraping an attractive workaround due to limited enforcement of the policies.

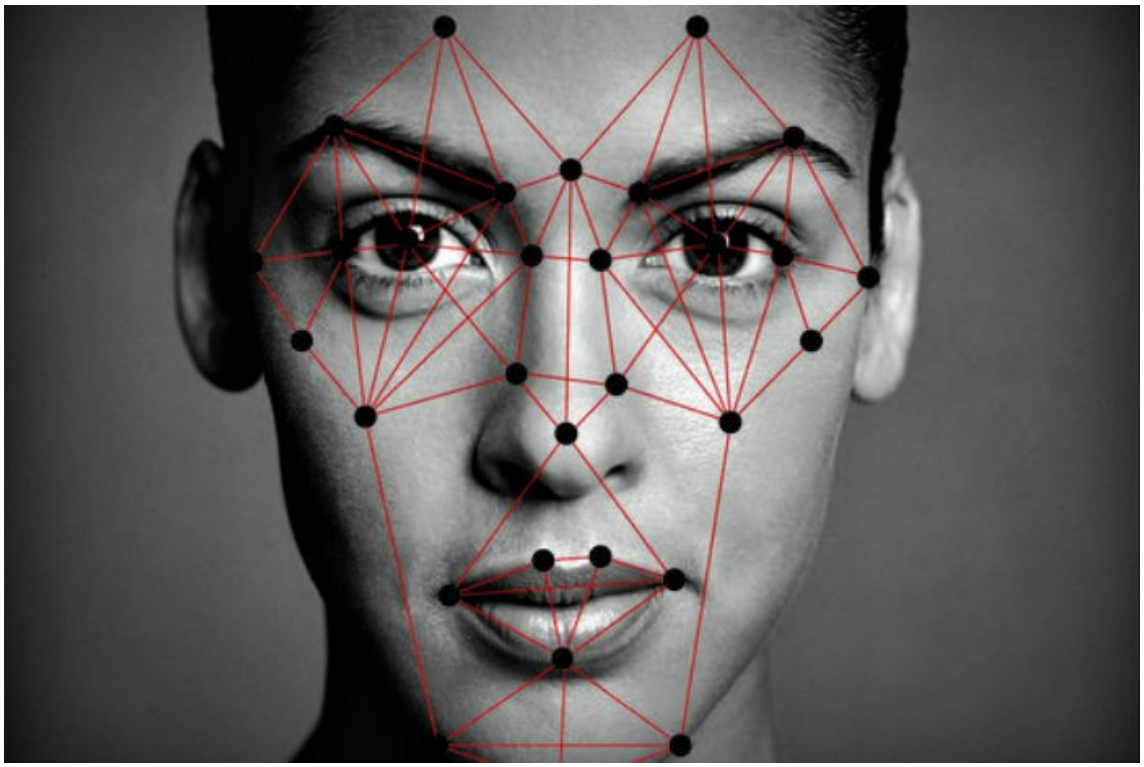


Figure 2.3: Facial recognition extracted key points example [3]

2.0.3 Online Recognition Platforms

Several public tools already implement facial recognition search engines, the most well known ones being:

- **PimEyes:** Allows users to search for appearances of a face across the web, but excludes social media, probably due to **GDPR** restrictions.
- **FaceSearch:** Targets social media content and includes features such as sex offender checks. It excludes European users, likely due to **GDPR** restrictions.

Both platforms offer free limited search previews, but require payment for full functionality. For **Pimeyes**, this includes the possibility to send take down requests to the websites that hold our pictures, however this comes at a price which would scale quite rapidly for an entire company. This is rather intended for individual use. Free opt-out mechanisms also exist for both tools, which are either image-specific, or requiring identity verification. We have tested both the procedure to remove images one by one and the one to remove a person by providing an anomyzed ID card with **Pimeyes**, and can confirm they do work. However these free options only remove someone from this one specific tool's search results, leaving them on other platforms.

There are free alternatives for face searching, notably [14], but none fit our requirements. Some only look for similitude in images, are from a non trustworthy source and most importantly they lack transparency or the implicit assurance that comes with a company having a reputation to maintain.

Clearview AI is a more powerful tool marketed to law enforcement. It claims unmatched accuracy, backed by a vast training dataset, though independent verification is lacking [16]. Clearview and similar tools pre-index massive datasets to improve search speed, likely constructing internal dossiers for each detected face [22].

2.0.4 Legal Considerations

Legal frameworks such as the General Data Protection Regulation and **AI act** aim to protect individual rights in the EU, but enforcement remains challenging, since some of these laws are subject to interpretation and good understanding of the context they exist in. Key **GDPR** principles include:

- **Proportionality:** Data collection must be justified and minimal. For instance, it would be hard to justify tracking the whole country through the means of facial recognition in order to catch a few criminals, especially if this is maintained over time.
- **Consent:** Individuals must be informed and, when applicable, provide explicit consent that their data is used in this way. There is more lenience in research contexts, and the law applies differently for criminal matters, as we'll explore in greater details [44].

The **AI act** entered into force on 1 august 2024, and will be fully applicable as of 2 august 2026, two years later. It is more focused on AI in particular. It classifies multiple levels of risks, and many within the unacceptable risks are of high interest to us. These are completely banned, whilst for others there can be more leniance. They include, (quoting): [17]

- Social scoring
- Individual criminal offence risk assessment or prediction untargeted scraping of the internet or CCTV material to create or expand facial recognition databases
- Emotion recognition in workplaces and education institutions
- Biometric categorisation to deduce certain protected characteristics
- Real-time remote biometric identification for law enforcement purposes in publicly accessible spaces

There are other laws that aim to protect individual's privacy, for instance in Canada, but we have chosen to focus on Europe in this thesis.

We should already note that Clearview's activities have drawn criticism for violating these principles, especially regarding mass data collection without consent. [44]

2.0.5 Defensive Techniques and Limitations

To counter facial recognition, two main strategies exist:

- **Detection and Removal:** Find and eliminate existing online images, according to the **GDPR**, **Digital Millennium Copyright Act**, and **Takedown Notices**. Our presence in the picture is enough to request its takedown. [29]
- **Obfuscation:** Prevent recognition by altering images as our face is being captured, for instance with physical disguise or specific make-up techniques. This can also be used on already existing online images, although if the company doing facial recognition already has a local cache of our picture, it will prove to be ineffective.

Tools like **Fawkes** [39] can manipulate existing images to disrupt recognition algorithms, though success depends on the strength of the obfuscation and the resilience of the recognition model. Adversarial make-up may be able to fool a face recognition equipped basic camera but not a full script.

It is important to note that no single strategy offers full protection. Advanced recognition systems can adapt to new defences (e.g., bypassing COVID mask interference [53]), and not all individuals will employ such methods. Thus, protective techniques must evolve alongside threat models.

Chapter 3

Literature Review, State of the Art, Definitions, and Notations

3.1 Introduction

The purpose of this chapter is to explore the scientific and technological foundations relevant to the use of facial recognition algorithms, and to establish the context for their real-world application, limitations, and legal implications.
The approach follows a series of research sub-questions derived from the main problem, ensuring comprehensive coverage of the technical, ethical, and societal issues at stake.

3.2 From Research Problem to Sub-Questions

To systematically address the main problem, we decompose it into specific sub-questions:

Research Sub-Questions

- How do facial recognition algorithms work?
- What already exists to satisfy our use case in terms of facial recognition algorithms?
- What are the limitations of these algorithms? Their mitigation?
- To what extent are facial recognition systems discriminatory?
- How does it connect to web scraping?
- What are the good practices for web scraping?
- What are the current real-world applications of facial recognition?
- What impact do they have on human performance?
- How can individuals protect themselves against facial recognition?
- What about the impact on society as a whole?
- What legal restrictions govern the use of this technology?
- What broader implications does facial recognition have for privacy?
- What means can we use to combat this technology?

These sub-questions will guide the development of subsequent sections.

3.3 Definitions

- **API (Application Programming Interface)**: A set of protocols and tools that allow different software applications to communicate. In the context of face recognition, APIs provide developers with access to pre-built recognition models and services.

- **Bias**: Systematic deviation in algorithmic performance that disproportionately affects certain demographic groups (e.g., by age, gender, or ethnicity), often due to imbalanced training data or design flaws.

- **CNN (Convolutional Neural Network)**: A deep learning architecture particularly effective at processing grid-like data such as images, widely used in face detection and recognition tasks.

- **ANN (Artificial Neural Network)**: A computational model inspired by the human brain, consisting of interconnected nodes (neurons) that can learn patterns from data.

- **GAN (Generative Adversarial Network)**: A class of neural networks used to generate synthetic data. It consists of two competing networks (generator and discriminator), and is often used to create deepfakes.

- **GDPR (General Data Protection Regulation)**: A comprehensive data protection law in the European Union that governs how personal data must be collected, stored, and used. It includes specific provisions on biometric data, including facial information.

- **DSA (Digital Services Act)**: A regulatory framework by the European Union aimed at increasing transparency and accountability of online platforms. It indirectly affects facial recognition by setting content moderation and algorithmic transparency requirements.

- **Eigenfaces**: Eigenfaces are a set of eigenvectors derived from the covariance matrix of facial images, typically used in Principal Component Analysis (PCA) for face recognition. They represent the principal components of facial variation in a training set, allowing for dimensionality reduction and simplified facial comparison in early facial recognition systems.

- **Layers (in Neural Networks)**: Layers are the fundamental building blocks of neural networks, comprising groups of interconnected neurons. They typically include input layers (receiving data), hidden layers (performing transformations), and output layers (producing predictions). In Convolutional Neural Networks (CNNs), layers can include convolutional layers (for feature extraction), pooling layers (for dimensionality reduction), and fully connected layers (for final classification).

- **OpenCV (Open Source Computer Vision Library)**: OpenCV is an open-source software library designed for real-time computer vision and machine learning applications. It provides a wide array of tools for image processing, object detection, facial recognition, and camera calibration, and supports integration with deep learning frameworks such as TensorFlow and PyTorch.

- **Computer Fraud and Abuse Act (CFAA)**: The Computer Fraud and Abuse Act is a United States federal law enacted in 1986 to combat cybercrime. It prohibits unauthorized access to computers and networks, including the theft of sensitive data or misuse of digital systems. The CFAA has been criticized for its broad language, which some argue can criminalize minor or ethical activities like security research or data scraping.

- **Face Decoy (Foggysight)**: The face decoy method, proposed by the Foggysight project, is a privacy-preserving technique where users upload images containing synthetic or manipulated faces to confuse facial recognition systems. These decoy faces act as adversarial noise in datasets, thereby reducing the effectiveness of automated face identification by contaminating training or inference inputs with misleading data.

- **ArcFace Loss**: ArcFace loss is a loss function designed specifically for face recognition tasks. It introduces an additive angular margin penalty to the softmax loss function, which enhances the discriminative power of face embeddings by enforcing a larger angular separation between identities. This approach ensures that embeddings for the same person are clustered more tightly together in the feature space, while embeddings for different people are pushed further apart. ArcFace is mathematically grounded in hypersphere geometry, improving both the accuracy and robustness of recognition systems, especially under unconstrained conditions.

3.4 Principles and Operation of Facial Recognition

By introducing the vulnerabilities of facial recognition with basic machine learning notions, we will be able to understand its limitations, and avenues to address them.

3.4.1 Overview of Deep Learning Approaches

From Pixels to Faces. To a computer, an image is simply a matrix of pixel values. Extracting meaningful information such as the presence of a face requires several processing steps. All faces share structural similarities, such as the relative position of eyes, nose, and mouth - which machine learning models can exploit to perform detection and recognition.

Face Embeddings. Most recognition systems transform faces into fixed-length feature vectors called embeddings. These vectors capture the essential characteristics of a face in a compressed format. Recognition is performed by comparing these embeddings using similarity metrics such as cosine distance or Euclidean distance. The lower the distance between two embeddings, the more likely they represent the same individual.

Machine learning Modern facial recognition systems predominantly use deep learning, especially Convolutional Neural Networks (**CNNs**), which are well-suited for visual pattern recognition. These networks are composed of layers that convolve the image to detect features such as edges, textures, and shapes. Through training on large datasets, **CNNs** learn to distinguish subtle differences between faces.

Unlike traditional neural networks, **CNNs** include pooling and fully connected layers that allow the models to retain spatial hierarchies while reducing computational complexity. They are trained using backpropagation, where the network adjusts its internal parameters based on the error between predictions and ground truth.

Traditional vs Deep Learning Approaches. Before the advent of deep learning, techniques like eigenfaces and Local Binary Patterns (**LBP**) were popular. These relied on dimensionality reduction methods such as Principal Component Analysis (**PCA**) to compress facial data into representative features. However, these methods often failed under real-world conditions like poor lighting or varied expressions. **CNNs**, in contrast, are more robust and generalizable. [2]

Generative Adversarial Networks (GANs). Recent advances have introduced **GANs**, which consist of two networks: a generator and a discriminator. These compete to create increasingly realistic synthetic images. In facial recognition, **GANs** can augment training datasets, improving model performance when data is scarce or imbalanced [34].

Our Approach: Pre-trained Models. In this project, we rely on pre-trained models, which have been trained on large-scale facial datasets and can generate embeddings from input images without additional training. This approach is both practical and legally safer, as it avoids the need to collect extensive amounts of personal data. While data augmentation using **GANs** is possible, it was not used here due to the project's privacy considerations and resource constraints.

3.4.2 Facial Recognition

There is no universally optimal algorithm or model for facial recognition. Performance depends on a range of factors including available computational resources, time constraints,

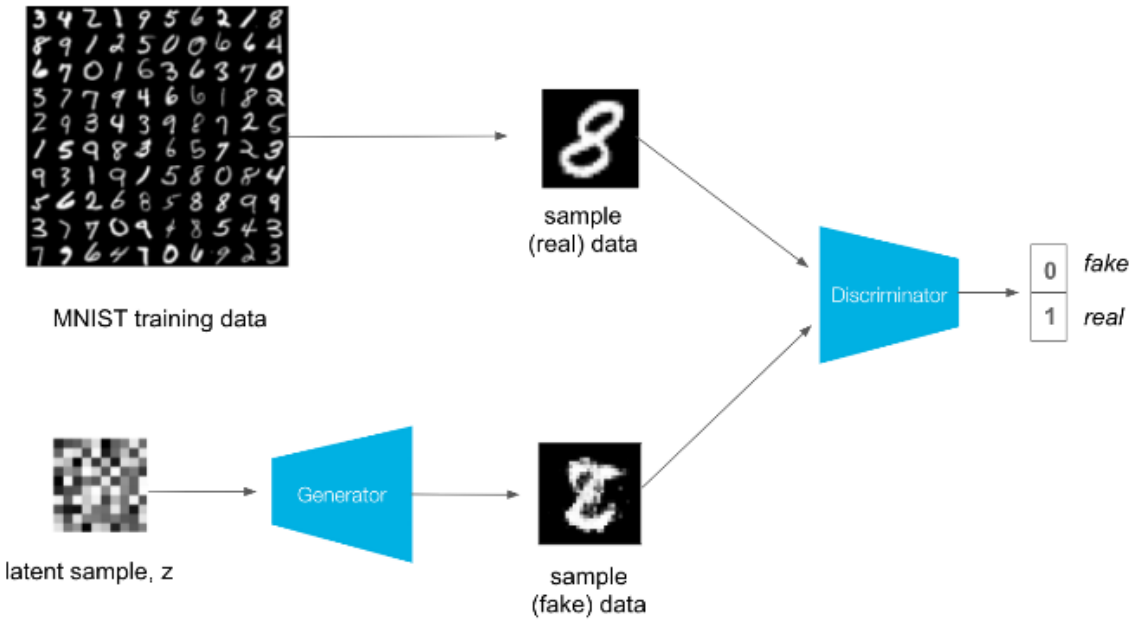


Figure 3.1: Generative Adversarial Network view [32]

dataset size, and the expected variability in image quality and content. Each model presents **trade-offs between speed, accuracy, memory usage, and robustness to adversarial conditions**. These performance differences often stem from the choice of training data and architecture-specific optimizations.

As such, it is common practice to combine different models, using one optimized for fast and reliable face detection and another one for more computationally intensive face recognition. For this project, we consider several modern approaches:

- **YuNet:** A lightweight, high-speed face detection model integrated with **OpenCV**'s deep learning module. It is particularly well-suited for real-time video processing and low-power devices [56]. **OpenCV** holds other models and also works with many different data formats and allows video capture, helping us get the pictures we need for the database.
- **YOLO (You Only Look Once):** A family of object detection models that can be adapted for face detection. While more computationally demanding than **YuNet**, it may provide improved accuracy and flexibility in complex scenes.
- **InsightFace:** A high-performance face recognition framework that employs ArcFace loss to generate highly discriminative face embeddings. It is considered one of the most accurate open-source solutions available.
- **DeepFace:** As discussed in [34], DeepFace offers strong performance with relatively modest resource demands. It represents a balance between accuracy and efficiency, although its preprocessing requirements make it less suitable for large-scale, resource-constrained tasks like web scraping.
- **Dlib:** This option has more false negatives in terms of recognition, and runs slower than both InsightFace and OpenCV, notably due to it being older, thus making it a less interesting choice [25]. It can also be used for detection, but will be slower than more recent options such as yunet.

According to [40], combining YOLO for detection with InsightFace for recognition provides strong results, benefiting from YOLO's versatility and InsightFace's embedding precision. However, given our need to process large numbers of images at scale - similar to adversaries like **Clearview AI** - we may prefer faster alternatives such as **YuNet**. We intent to evaluate both detection options by comparing their speed and accuracy in realistic scenarios before making a final selection.

New developments While this study focuses on these specific models, it is worth noting that the field is evolving rapidly, with many options available. This means that our system design should be as modular as possible to accommodate future advances or changes in performance requirements. The issue that comes with this these can come in different formats (for instance, facial points for yolo) for detection and recognition, sometimes they expect a given size for detection or recognition, and we will have to make sure to interpret the images in the different model specific ways. This could cause complications especially in the clarity of the code if we add many different models with different expectations.

3.4.3 Models Limitations

Despite advancements in performances and accuracy, facial recognition systems face inherent limitations [2]:

- **Sensitivity to illumination, posture, and occlusion**: Especially for 2D models (as opposed to 3D ones [43]), small changes in conditions can significantly impact performance, due to the heavy reliance on some areas of the face being immobile, and using light in order to assess depth and recognize the shape of the face. The posture importance is due to having to normalize the faces for comparison. This can be exploited, but modern models perform better in this way.
- **Performances**: A lot of resources are still required to run the model, making it less practical on smaller devices, where simpler versions have to be run, leading to lower accuracy [41].
- **Generalization issues**: Performance may drop when encountering faces or demographics under-represented in the training data. We should prioritize the models that perform better for diversirty, such as openCV and InsightFace, and avoid relying on other algorithms in contexts like criminal investigations [13, 48].

3.5 Mitigations

A good method to help with posture and light limitations is to build a better representation of the person we are trying to identify, by capturing their face under different poses and ambient light, or at least with sufficient quality. Note that there is no need to have a studio to take the pictures, but the higher quality and more varied they will be, the easier it will be to differentiate the person.

Pre-processing. It is also possible to pre-process data in order to make recognition easier, however we will not do this because of performance concerns.

The resource concerns can be mitigated by being careful not to over-use our model, for instance by only running the face recognition once on every picture online, remembering it, and then comparing them with all the faces in our database at once. Big companies

like **Clearview** AI hold the data in their data centers, allowing them to already pre-sort faces of people, reducing how many comparisons are necessary [16].

3.5.1 Algorithmic Bias in Facial Recognition

Despite their technical sophistication, machine learning models are not inherently neutral or objective. The accuracy and fairness of facial recognition systems are deeply influenced by the potentially unbalanced datasets they are trained on. Often white adult males-models tend to perform significantly better on those groups, while underperforming on others, including women, children, and individuals with darker skin tones.

This disparity is often not the result of explicit intent, but rather a reflection of systemic biases in data collection practices and the composition of research teams [48].

Implications The implications are particularly serious in high-stakes domains like law enforcement, where misidentifications can lead to wrongful arrests, surveillance, or denial of services. In several countries, including the U.S., law enforcement agencies have adopted facial recognition tools without robust oversight, prompting concerns about legal accountability and civil liberties.

It is also important to note that performance gaps are not always consistent across demographic categories. For instance, some systems may accurately distinguish gender for one ethnic group but struggle with others, revealing the nuanced and multi-dimensional nature of algorithmic bias.

Evolution Comparative evaluations, such as that by Fabian et al. [25], indicate that some modern frameworks, such as **OpenCV** and **InsightFace** - which we will be using - exhibit more balanced performance across demographic groups than older algorithms such as Dlib. Nevertheless, disparities persist, especially under challenging conditions such as poor lighting or non-frontal facial angles.

Finally, evaluations by the U.S. National Institute of Standards and Technology (**NIST**) have shown that the most accurate facial recognition algorithms often also exhibit the least bias [13], they have also shown that the most disparity is for african women, for which the false match rate was 100 times worse than for white men. We recommend the reading of the 2019 report on demographic effects [27].

3.5.2 Mitigation Strategies

There are a few ways to address the remaining biases, focusing on the training of said algorithms:

- **Balanced and representative training datasets:** Increasing diversity in datasets across ethnicity, gender, age, and geographic origin helps ensure that models generalize well across populations.
- **Corrective learning objectives:** Some training algorithms now incorporate constraints or loss functions that penalize disparities in performance across groups, encouraging more equitable model behavior.
- **Independent evaluation and auditing:** This makes sense before employing the technology on a greater scale, especially for law enforcement. As [48] points out, there were legitimate concerns around DNA identification at first as well.

While human oversight remains a critical layer in mitigating the consequences of algorithmic errors, the influence of algorithmic outputs on human judgement should not be underestimated, and go beyond simple confirmation bias, as we will detail further.

3.5.3 Web Scraping for Facial Recognition: Practical and Ethical Considerations

In the context of facial recognition, scraping images from the web can be a valuable method for assembling large datasets necessary for training and evaluating algorithms. However, this practice presents several technical, legal, and ethical challenges that must be considered, as guided in [9]. This paper's first focus was the USA, but they also mention the case of the GDPR, as we will expand upon later.

Scraping as a technology can be used in many different ways, notably for profit by analyzing market trends and more, but also for research and the training of various machine learning algorithms. It can be done by simply making sense of the html code of websites or using custom APIs.

Legal and Ethical Considerations

The legality of web scraping is nuanced and varies across jurisdictions. In the United States, for instance, the Ninth Circuit Court ruled in *hiQ Labs, Inc. v. LinkedIn Corp.* that scraping publicly accessible data does not violate the Computer Fraud and Abuse Act (CFAA) [35].

Scraping for research. This decision does not grant freedom for all scraping, given that websites often include terms of service that explicitly prohibit automated data collection. Violating these terms can lead to legal repercussions, and should normally be informed within the robot.txt, which can explicitely give permission to third parties for scraping, often corresponding to legal contracts. Twitter used to give access to researchers, but has transitioned to a paying system since April 2023, and access to it must now be granted per project. In August 2024, Meta shut down CrowdTangle. This is making it much harder to use that data for research purposes [35] .

There is also the matter of copyrights and individual property of images, that can be ignored by the various scraping tools or even websites which may repost content without particular attention.

Ethical issues of scraping Ethically, scraping images for facial recognition raises significant privacy concerns. Individuals whose images are collected have most often consented to such use, especially when images are sourced from social media platforms or personal websites. The case of Clearview AI exemplifies these issues: the company scraped billions of images from the web to build a facial recognition database used by law enforcement, sparking much criticism, and legal challenges in the EU [44], leading to fines in France and the Netherlands. With the new AI act such scraping in order to make facial recognition models will be banned, and although ClearView might try to get it to work before it is implemented, this seems unlikely to work [17].

Researchers and practitioners must also consider the potential for algorithmic bias introduced by these unbalanced online datasets. It is reasonable to assume the different demographics do not post in the same frequency, neither are they of the same proportions, leading to imbalances. If certain demographic groups are under-represented in the scraped

data, the resulting facial recognition models may exhibit reduced accuracy for those groups, leading to discriminatory outcomes [9].

Best Practices for Ethical Scraping

These are practices we should consider, to avoid legal and ethical problems when doing web scraping in this research context:

- **Respect Website Policies:** Always review and adhere to a website's terms of service and robots.txt file to determine permissible scraping activities.
- **Safeguard Data:** Protect the collected data against unauthorized access or breaches. Indeed our prototype will produce a file that will contain the results of the search, and should be safely guarded once it is deployed.
- **GDPR:** For web scraping researchers are left with more freedom than most, so long as no personal information has been collected, they do not have to send privacy notice under the **GDPR**. Ironically doing so would require identifying the persons in the pictures. They should be able and willing to demonstrate the proportionality [9]. Researchers are also able to request large content provider platforms to provide them with access under certain conditions, with the DSA in Europe.
- **Privacy:** For now we do not keep the images, still we should never identify persons not from our database or not having given consent for us to do so. However, we should also add that the data collected should be of justifiable size and purpose. Under these considerations, the **GDPR** states that we do not have to inform and ask for consent when looking for information, due to the enormous effort that would be required. Indeed, this is not considered human subject research, but more like observation, but this remains open to interpretation.
- **Picture context:** We should consider the context in which users will be placing pictures online, and not try to access any private pictures or "hot" websites. With all of these concerns, we should keep in mind we may miss some pictures, but there is so much data to find that we should have enough information already. We will also not be looking on the dark web.
- **Website scale:** We should also be mindful not to place too much on a drain on websites themselves, and instead place most of the load on big websites if possible (for instance avoid scraping a small video platform). This is also a concern for us, which will be explored in the next section.
- **Variability:** Even though we are not doing machine learning we still have to concern ourselves with taking data from varied website sources in order to have a representative web presence.

This makes a scraping strategy that should be kept transparent by adding it to the script's readme.md. It should be kept in mind when performing web scraping, or using the prototype, and updated. Furthermore, the script itself should also remain as human understandable and well explained as possible, to make it possible to review it by a more varied audiance.

Technical Challenges

While scraping text is relatively straightforward, collecting images poses issues. High-resolution images consume bandwidth and/or storage resources. In order to optimize the scraping process for facial data collection, focusing on saving bandwidth, we can think of multiple mitigation strategies:

- **Storage:** Store processed image URLs to avoid looking at them multiple times over different searches if no new person has been added to the database. Eventually we could even store the pictures within a database of consequent size, to not have to query them anymore, as is the case for Clearview, according to its representative [16].
- **Frequency:** Limit the frequency of scans to reduce server load, for instance by doing weekly updates and only checking for new online pictures then.
- **Videos:** The bandwidth cost of videos is much worse than for pictures, one mitigation would be to only look at key frames from videos, every few seconds for instance, but we will most likely still have to download the whole video from the website, and can only save on performance costs by reducing the amount of face scanning performed. This could be improved later on, for instance by choosing the frames better. 'Thankfully' for us, youtube and twitter refuse such scraping, reducing the total load.
- **Metadata:** Filter images using available metadata, such as geographic location, to ensure dataset diversity - or the opposite, if we want to focus on a specific country for instance. The issue is that this information is often missing from pictures, and could easily be manipulated, preventing us from relying on it. Instead, we can choose to scrape the websites that are relevant to the location of interest to us.

3.6 Real-World Applications and Effects

Before addressing the legal frameworks that govern facial recognition technologies, it is essential to understand the current landscape of their deployment and the real-world impact they exert across different sectors.

3.6.1 Deployment Contexts

Facial recognition technologies are being increasingly integrated into a wide variety of applications, both commercial and governmental:

- **Law enforcement:** Police use facial recognition to identify suspects, locate missing persons, and corroborate investigative leads [1].
- **Public surveillance:** In countries like China, facial recognition supports comprehensive citizen monitoring systems; in the United States, numerous law enforcement agencies have adopted it for crowd scanning and suspect identification [11].
- **Commercial use:** Biometric authentication such as Apple's Face ID demonstrates the growing consumer-facing deployment of facial recognition. Some stores use the technology to identify repeat customers or known shoplifters, sometimes raising concerns about consent and profiling.

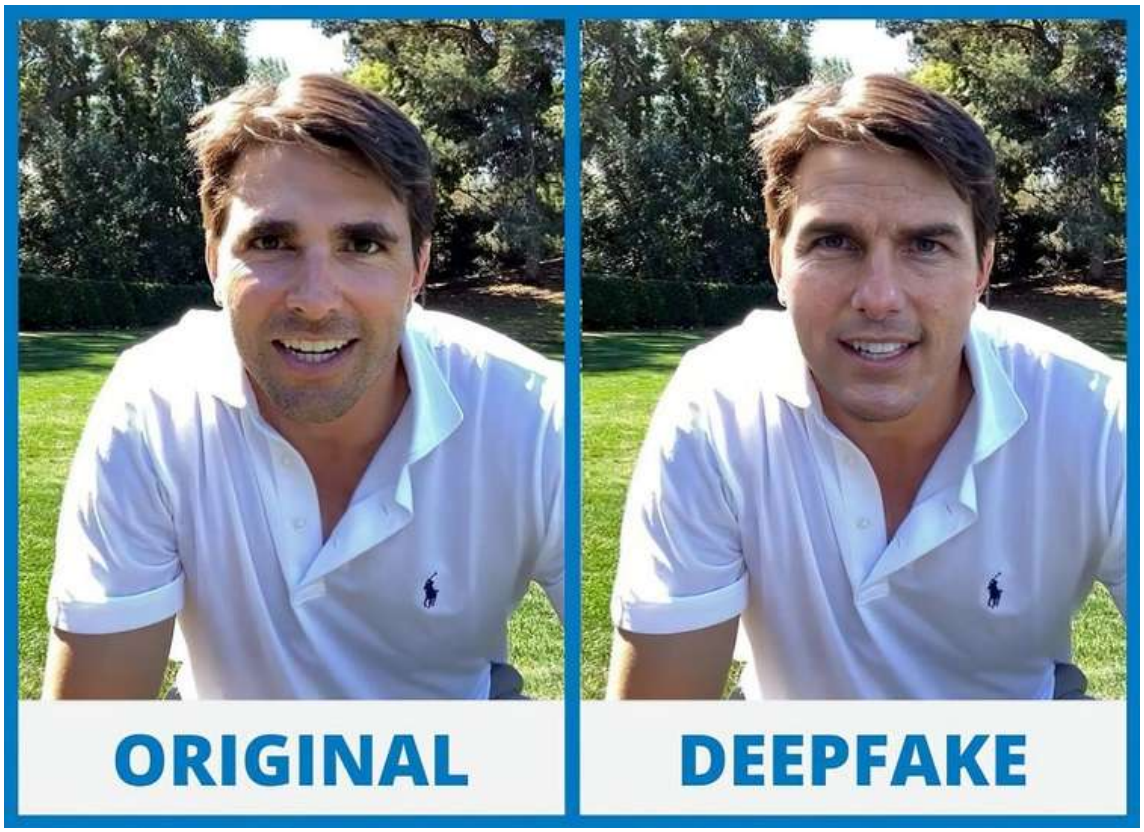


Figure 3.2: Deepfake example [36]

- **Border control and airport security:** Facial recognition streamlines passenger processing and improves security at international transit hubs [2].

In addition to identification tasks, research has explored future applications such as facial reconstruction in medical contexts [41], as well as emotion and age estimation based on facial data [43].

3.6.2 Deepfakes and Their Interplay with Facial Recognition

Deepfakes, primarily generated through Generative Adversarial Networks (GANs), are synthetic media in which a person's likeness is manipulated or entirely fabricated. These technologies can produce highly realistic facial images and videos, often indistinguishable from genuine ones to the human eye [52]. While deepfakes have legitimate applications in entertainment and accessibility, they are increasingly used for malicious purposes, including defamation, identity theft, and political misinformation. We will focus on facial

deepfakes, synthetic content that manipulates facial expressions, replaces identities with face swapping, or animates still images. Such alterations can be performed starting from a single image, using techniques like Video-Driven Facial Re-enactment, in which a source actor's facial movements are transferred to a target individual. These methods can produce videos that appear authentic but are entirely fabricated.

Combined risks The integration of deepfake generation with facial recognition technologies introduces a critical risk. If law enforcement or forensic investigators rely on facial

recognition as evidence, the existence of convincing deepfakes undermines the reliability of visual data. As facial recognition is increasingly used by police and border control, the potential for evidentiary manipulation or wrongful identification grows in parallel [23]. The use of deepfakes in criminal or adversarial contexts - such as framing individuals, impersonating suspects, or fabricating alibis - poses a serious threat to the integrity of digital evidence.

To mitigate these risks, one potential enhancement to our facial recognition prototype would be the integration of a deepfake detection module, which typically works by detecting small perturbations made with the generation. The verification system could be used to verify whether a given input has been synthetically manipulated, however this should be done manually due to the performance trade-off, and will be left for future optional development if the technology develops further. Several detection methods have been proposed in the literature, often based on inconsistencies in eye movement, head pose, illumination artifacts, or subtle temporal anomalies not present in real recordings.

Future development The proliferation of tools capable of generating convincing deepfakes means that the prevalence of manipulated facial data is likely to rise significantly in the coming years. As such, any serious deployment of facial recognition in public or legal contexts must consider not only the risks of bias and surveillance, but also the growing possibility of synthetic deception.

3.6.3 Impact on Human Performance

Facial recognition is not intended to replace human judgment but to augment or guide it. However, several studies have shown that this combination can paradoxically degrade human performance. Operators tend to place too much confidence in algorithmic suggestions, which introduces cognitive biases. [15] shows that identification is more difficult when individuals look similar, and that pre-selection by an algorithm can obscure human discernment, reducing overall accuracy.

Recognition experts. [24] argues that identity verification is a complex and expert-driven task. Not all operators are equally trained, and many lack the visual expertise required to make reliable decisions based on facial similarity. Judges and legal practitioners should recognize that identification from images is inherently subjective and must be treated with corresponding caution.

Framing [19] emphasizes that algorithmic framing shapes human decision-making. If a system suggests a match, users are more inclined to confirm that match - even if uncertain. The authors argue that how the system is used matters more than its technical specifications. They also highlight the role of surveillance infrastructure: camera angles, image quality, and coverage zones all affect the reliability of identification. While these technologies can help identify suspects who might otherwise evade detection, they simultaneously raise the risk of misidentification and over-reliance.

Growing concerns. As surveillance becomes more pervasive, it fosters a shift in societal attitudes, encouraging suspicion and redefining the boundary between legitimate investigation and invasive monitoring. This evolution raises deeper concerns about bias, over-policing, and the erosion of civil liberties.

3.6.4 Clearview AI and the European Perspective

A particularly controversial example of real-world deployment is **Clearview AI**. This U.S.-based company scraped billions of facial images from publicly accessible websites, including social media platforms - without user consent. The collected data was used to develop a facial recognition platform, which has since been adopted by over 600 law enforcement agencies in the U.S [51].

Clearview's approach raised significant ethical and legal concerns. Besides breaching the terms of service of several major platforms (e.g., Google, Facebook), the company's practices drew criticism for their potential to facilitate mass surveillance. A data breach in 2020 further exposed the risk of large-scale misuse, though only the client list was leaked at the time.

European context In the European Union, such practices are subject to stricter legal scrutiny, even before the **AI act** comes into play. Under the General Data Protection Regulation (GDPR), facial images may be considered biometric data if used for identification purposes. In addition to the GDPR, the EU has a dedicated law enforcement data protection directive (LED), which governs the processing of personal data for crime prevention and prosecution. According to [44], the use of **Clearview**'s technology in Europe would likely contravene these frameworks. The study emphasizes that unlike ad hoc data requests traditionally made by law enforcement to social media platforms, **Clearview**'s systematic data scraping and storage introduces a new paradigm of persistent biometric surveillance.

Company stance **Clearview**'s CEO has claimed that the company's methods are lawful in the U.S. and have contributed to solving crimes. However, European regulations require that any processing of biometric data must be necessary, proportionate, and carried out with clear safeguards. Additionally, **Clearview**'s business model - a for-profit service provider - complicates its legal standing. The study notes that "facial images per se do not constitute biometric data under the EU data protection legislation," but the moment they are used for identification purposes, they fall within the scope of GDPR protections.

The **Clearview** case illustrates the tension between technological capabilities and legal frameworks. It also highlights the regulatory divergence between the U.S. and the EU, where the latter places a stronger emphasis on data protection, consent, and proportionality.

3.6.5 Potential Solutions and the Path Forward

Addressing the risks posed by facial recognition technologies requires a multi-pronged approach:

- **Public education and digital hygiene:** Individuals should be made more aware of the implications of uploading personal photos online. The "privacy paradox", the gap between privacy concerns and actual behavior, remains a central challenge, especially for younger generations who are more active on social media.
- **Legislative reform and moratoria:** As suggested by [37], temporary bans on facial recognition deployment may be necessary until proper regulations are enacted.

Public consultation and democratic oversight must play a central role in determining acceptable uses.

- **Corporate responsibility and transparency:** Private companies should adopt clear policies about how facial data is collected, stored, and used. This includes transparency around partnerships with law enforcement.

Ultimately, unregulated facial recognition turns individuals into "walking ID cards," as McSorley puts it. This metaphor captures the fundamental risk: that constant identification becomes normalized, while the technology itself remains flawed and biased. Even companies like Facebook have withdrawn from facial recognition deployment in November 2021, they say that with concerns on the impact on society they have chosen to remove it for now [26]. Facebook used to give the option to recommend who to tag in pictures, and be notified when they appeared in photos, based on opt-ins.

As this technology advances, the need for a rights-based, accountable, and transparent approach becomes more urgent - especially in democratic societies committed to individual autonomy and rule of law.

3.6.6 Techniques for Avoidance

By understanding the inner workings of facial recognition we can surmise countermeasures. For instance, the algorithms rely most on parts of our face that do not move to perform recognition, and on light in order to assess depth, making it possible to deceive less advanced ones with the right technique, as shown in this video [28].

Several methods have been developed to reduce detectability by facial recognition systems:

- Adversarial makeup.
- Use of infra-red LEDs to interfere with image capture.
- Strategic occlusions (e.g., hats, scarves) to disrupt key facial landmarks.

Naturally this poses the issue that most of these cannot be worn in public without attracting attention, which is the opposite of what we would like to do. Modern facial recognition tools like face++ are also very resilient against these methods [49]. The only solution may be to dress with masks like robbers, or else, which could be imagined for the police for instance, although it is not the most elegant solution, and even then the models might be able to adapt eventually.

Machine Learning Model Attacks

Even with increased public awareness about digital privacy, users need effective technical tools to protect themselves, beyond simply refraining from posting images online or resorting to adversarial make-up techniques.

Protection methods. There are preventative protection techniques, to be applied **before** the image is posted. Once an image is online, it can be scraped and incorporated into facial recognition databases, making post hoc protection far less effective. These methods aim to preserve human interpretability i.e. humans should still be able to recognize the individual in the modified photo, while misleading machine learning (ML) models.

Training data One class of countermeasures involves **poisoning** the training data. This refers to the manipulation of images found online so that, during model training, the algorithm learns misleading or incorrect features for an individual's face. This attack can be highly effective due to the sensitivity of ML models to even minor perturbations. However, research in this area is limited and often theoretical, as noted by [21].

Fawkes More attention has been given to **post-training perturbation techniques**, which require some knowledge of the facial recognition model (a white-box setting). These techniques exploit the fact that machine learning models sometimes memorize specific aspects of their training data, leading to privacy leakage. One defense mechanism proposed in this context is **differentially private model training**, which aims to limit such memorization [49].

Foggysight Approach Foggysight proposes a decoy-based approach, where fake or modified images resembling other individuals are uploaded in bulk. This confuses the model during inference by presenting multiple potential matches. They also propose collaborative systems for automating this type of protection. However, the effectiveness of this method is conditional:

- It requires a significant number of decoy images, about five times more than the unmodified ones to reduce model accuracy below 50%.
- It depends heavily on wide user adoption.
- There is a risk that users may start being misclassified as someone else, especially if decoys flood the online space.

Fawkes Cloaking Tool The **Fawkes** project offers a more targeted approach, modifying facial features in a way that disrupts how the model encodes identity. The cloak works by shifting the model's feature extraction to a different class label - essentially making the AI "think" the person is someone else. Key points include:

- Claims of 80% protection even when public photos already exist, and up to 95% under certain conditions.
- The transformation remains recognizable to humans but is misinterpreted by the algorithm.
- According to the authors, cloaks are resilient to conventional image transformations (cropping, scaling, compression).
- Fawkes allegedly evades detection by outlier detection algorithms used to identify tampered images.

3.6.7 Effectiveness and Limitations

While promising, these defence mechanisms come with important caveats and limitations that affect their real-world viability:

- **Limited effectiveness against advanced models**: Most avoidance techniques work better against less robust or poorly trained systems. Highly optimized models such as those used in law enforcement or corporate applications are significantly harder to fool, especially in controlled environments.

- **Existing images cannot be retroactively protected**: Any image that has already been scraped and stored in a facial recognition database cannot be cloaked after the fact. These images contribute to the model's accuracy and can still be used for identification.
- **Model retraining circumvents many countermeasures**: Due to the adaptive nature of machine learning, even if a technique temporarily reduces recognition accuracy, a retrained model can often learn to ignore or bypass the perturbations. This was evident in the development of facial recognition models that work despite face masks [53].
- **Risk of reverse identification**: If cloaked and uncloaked images of the same person are available, the model may eventually learn to associate them, thereby reducing the effectiveness of the defence.
- **Detection risk**: As mentionned in the paper, cloaking methods such as Foggysight and Fawkes may be detected, in which case they will most likely be ignored - in the paper they showed that existing anomaly detection would likely not work. In case of detection, this would mean the images would not be used to produce face embeddings.
- **Dependence on mass adoption**: In the case of decoy-based methods like Foggysight, effectiveness is contingent on a high participation rate. Without widespread use, their ability to meaningfully lower recognition accuracy is diminished.
- **Multiple methods approach**: Given the arms race between privacy defenses and recognition technologies, no single method is sufficient. Effective protection likely requires a combination of approaches such as cloaking, adversarial perturbations, legal takedown tools, and differential privacy.

In conclusion, while technologies like Fawkes and Foggysight show promise in limiting facial recognition, their success depends on many variables: adversary capabilities, deployment scale, and user cooperation. Furthermore, ongoing improvements in machine learning mean that privacy-preserving tools must continually evolve to remain effective.

3.7 Review Methodology

3.7.1 Literature Review Methodology

The literature review was conducted to establish a solid understanding of the technical, legal, and ethical dimensions of facial recognition technology. Primary sources were retrieved using the ULB Cible+ academic portal, prioritizing peer-reviewed articles, conference papers, technical reports, and white papers.

Searches were performed using keywords including but not limited to: *"facial recognition"*, *"facial recognition bias"*, *"facial recognition algorithms"*, *"Clearview AI"*, *"PimEyes"*, *"web scraping"*, *"facial recognition countermeasures"*, *"deepfake detection"*, and *"face verification pipeline"*.

In addition to academic databases, relevant industry reports and technical documentation were also reviewed, especially where they offered practical insights into existing implementations (e.g., InsightFace, OpenCV, or YOLO-based detectors). Where access

was limited due to paywalls, Google Scholar was used to locate openly available versions. ChatGPT was also used as a supplementary tool to clarify concepts, track citations, and explore related research directions that emerged during implementation, although any factual claims were verified via peer-reviewed sources. Zotero was used to keep track of the papers and review them later for more information, as well as putting important parts in evidence and generating the bibliography.

The selection criteria emphasized recent publications (mostly post-2020) and included works that offered comparative evaluations of algorithms, legal and regulatory analysis (e.g., GDPR, AI Act), or proposed novel methods of privacy protection (e.g., Fawkes, FoggySight). Sources were documented in a BibTeX database and cited consistently throughout the thesis.

3.8 Summary of the State of the Art

This chapter has provided a comprehensive overview of facial recognition technology in both its technical and societal dimensions, offering essential context for the development of the proposed prototype.

We began by examining how facial recognition systems function, including the deep learning models that enable them, which provided a foundation for understanding their strengths and limitations. From this, we identified promising algorithms to integrate into our prototype and outlined approaches to optimize their performance while mitigating known biases, particularly those related to demographic disparities.

We then turned our attention to web scraping, a critical component of data acquisition for such systems. This included an analysis of its technical implementation and the ethical and legal considerations that govern its use, especially under European data protection laws.

Special attention was given to the risks posed by deepfakes, which could both challenge recognition systems and be countered by them. While not the primary focus of this work, such risks remain relevant to the broader context of identity verification and digital manipulation.

Real-world applications across law enforcement, surveillance, and commercial platforms were reviewed, revealing both the capabilities and consequences of deploying such technology at scale. Importantly, we considered the human factors involved—such as operator trust in automated systems—and the societal implications of widespread recognition deployment.

Finally, we reviewed current legal frameworks, particularly within the European Union, and highlighted existing and emerging countermeasures individuals can take to protect their facial privacy, including adversarial tools like Fawkes and Foggysight.

Together, these insights inform the design choices of our prototype and frame its ethical boundaries. They also underscore the urgency of providing transparent, user-empowering tools in a space dominated by opaque, proprietary systems.

Chapter 4
Project Mission, Objectives, and Requirements

4.1 Overview and Objectives

The primary objective of this project is to design a script capable of scanning publicly accessible areas of the web, identifying online images and videos containing faces, and detecting the presence of specific individuals based on a reference dataset. The script should function as an investigative tool, enabling visibility into a person's online presence via facial recognition. Later on it can be expanded to faciliate sending takedown requests, or requests for passing the picture through an algorithm making facial recognition more difficult - such as Fawkes, if a company or government organisation wishes to protect its members.

State-of-the-art facial recognition models already provide robust tools for identity matching, and various web scraping techniques allow automated media retrieval from online sources. However, an open-source integration of these components - built to prioritize scalability, privacy protection, and usability - is not yet widely available. Commercial solutions like Pimeyes and Facesearch exist but are financially inaccessible for large-scale monitoring or activist-oriented initiatives, often costing up to €40 per person per month. There are other tools available, but none fit our specific use case effectively, due to only using generic recognition or looking only within a set database, or for social media profiles. We would also prefer the algorithm to be open source and transparent.

This project assumes the user will obtain and supply a dataset of known individuals (e.g., images from a webcam using a provided script, or other photos) to initialize the recognition process. Legal constraints and terms of service limit the scraping of certain platforms, meaning Meta and Twitter are the only available social media platforms with their respective APIs. Naturally the limitations are respected in our design.

4.2 Anticipated Risks and Limitations

There are multiple challenges we will have to overcome or find our way around when scanning most of the web for public pictures:

- **Bandwidth and Scalability:** The volume of data to be scanned is large. Unlike commercial providers, we do not cache the entire web or maintain a centralized media database. I will try to use multi-threading to improve performances.
- **Scraping Limitations:** Social media platforms increasingly restrict scraping via technical (e.g., rate limiting, obfuscation) and legal mechanisms. The project will only target legally accessible and permitted sources.
- **Legal Compliance:** Compliance with GDPR and platform-specific terms of service is critical. Takedown or media obfuscation requests will be left to the user to initiate, based on local laws.

- **Ethical concerns:** The script could be misused, for instance in order to perform some form of background checks on employees. In order to address this, as much transparency as possible is advised, and a guide provided with the script. If the database is protected, the script will always be run in the same way and cannot be targeted at anyone who did not give consent to be present in the database. It should not be ran by anyone with conflict of interests such as asserting employee's reliability. A potential system would be for the script to run automatically and directly contact the concerned employees.

- **False Positives, False Negatives:** Face recognition systems are not infallible, forcing the presence of a human double checking the script's results. We will not be able access every pictures, nor will we be able to find and accurately recognize all faces.

- **Inaccuracy issues:** We may lack enough images, or in bad conditions, not sufficient to recognize the given individuals in different light conditions and poses. This means we may have a lot of pictures with people identified with an accuracy too low to warn them, but also too high to ignore. An auditing system may be required to have a human verify the results.

- **Performance limitations:** The facial recognition algorithms take a lot of resources to run. There is however a possibility of relying on cloud resources for running this script for deployment, and a modern computer is sufficient for testing of the prototype.

4.3 Requirements

4.3.1 List of Requirements and Dependencies

The project's functionality depends on satisfying the following key requirements, listed in their logical flow:

1. **Input Dataset Collection:**
    - Collect decent quality images of each target individual, ideally in different lighting and poses.
    - Normalize and crop these images to feed the recognition model.
2. **Web Scraping Engine:**
    - Extract publicly accessible media content from websites (news archives, public forums, media repositories).
    - Filter for image and video content.
3. **Face Detection Algorithm:**
    - Identify faces in static images or video frames with low latency and acceptable accuracy.
    - Prioritize lightweight models.
4. **Face Recognition Algorithm:**

- Match detected faces against the whole known dataset.
- Return confidence scores to support manual validation.

5. **Result Logs:**
   - Aggregate matches in a structured output format.
   - Offer logs for the one running the script and the concerned individuals.
6. **Best Practices:**
   - Include a text on how this code should be used.
7. **Legal Compliance Layer (optional):**
   - Provide metadata to support takedown or scrambling requests (e.g., URL, timestamp, matched identity).

4.3.2 Requirements Covered by the State of the Art

The following components are well-supported by existing tools and research:

- **Media scraping:** Python libraries such as `requests`, `BeautifulSoup`, and `yt-dlp` support scraping and downloading.
- **Face detection:** Tools like YuNet and YOLO provide fast and lightweight face detection.
- **Face recognition:** InsightFace and similar models offer high accuracy using deep embeddings and cosine similarity.
- **Best Practices:** We have seen the legal requirements to keep in mind as well as potential effects on human operators and what to be mindful of.

4.3.3 Requirements Not Fully Covered by the State of the Art

The innovation of this project lies in integrating the above components:

- **Pipeline Integration:** No existing open-source framework combines scraping, detection, recognition, and export in a user-friendly, legal-compliant and free way.
- **Scalability Optimization:** Handling multiple identities within the database in an efficient manner.
- **Ethical Controls:** Most facial recognition libraries do not embed safeguards for abuse prevention or usage audit.
- **Robust Output Interface:** Aggregated, human-readable summaries of results for end users are missing in low-level tools.

4.4 Project Scoping

4.4.1 Mission Statement

To build a lightweight and modular script that can identify the presence of specific individuals in publicly available media on the internet, using existing deep learning models for face recognition and legally permissible scraping methods.

The script will be aimed at organisations and come with a guide for best practices. The guide will include legal information as well as intended use, and what should be kept in mind when using it.

Compliance

This project was made taking into account legal and ethical requirements. As such, it will come with a guideline for best use within the readme.md, and built-in safegards such as verifying scraping permissions of the various websites.

GDPR We will be performing identification for the persons present in the database, that will have to explicitly state their agreement. As we will not try to identify anyone outside of these, and respect the website's scraping policies as defined in their robot.txt, this project should be compliant with the law.

We will not be keeping the scraped pictures any longer than required for research purposes, and only keep the ones that are likely to contain one of our protectees for auditing. Naturally consent for the pictures used to scan will be taken in full awareness and can be withdrawn.

AI act As we are not building a better facial recognition database or training an AI model, our judgment is that we are compliant with this law.

4.4.2 Explicit Out-of-Scope Elements

- No scraping of websites that prohibit such access via their Terms of Service (e.g., Google).
- No integration with APIs that require corporate registration or paid access (e.g., Twitter API, limited meta integration).
- No legal automation: Takedown requests or enforcement actions are left to the user.
- No ongoing monitoring infrastructure: the tool is intended for ad-hoc, on-demand analysis only.

Many websites refuse acces to crawlers, and the API provided by most social media companies often do not include image gathering (such as the one from Tiktok and Linkedin). Instead it is limited to other informations more useful for advertisers. Only meta allows restricted access to public groups, provided we have a corporate registration.

However, like mentionned in the state of the art, it is possible to get access to the twitter API for research purposes thanks to the DSA act, by using this form. However, this is limited in time and goal, meaning eventually one would have to pay for a more advanced version.

4.5 Testing

Once the code is functional, we will experiment with the sufficient amount of protectee pictures for proper recognition, as well as investigate the requirements for specific light conditions or varied ones.

Once this has been found, we will acquire a dataset to test our recognition on, preferably including unusual cases like partial occlusion or lighting to test the limits of our algorithm.

Testing will be made easier by an added logging system. Throughout the development of the script, we will focus on keeping the code readable and well organised.

Once we reach the final versions, we will end by experimenting with the scraping mechanisms to get a concrete idea of how much time the process will take, and how we can improve it.

Chapter 5
Good Practice Guide

As discussed in the preceding sections, the software accompanying this thesis will include a usage guide. This guide will be integrated into the README.md file and will provide essential instructions and ethical considerations for operating the script.

It is strongly recommended that users first read the associated master's thesis and the "Web Scraping for Researchers" guide by Brown (2024) [9] to gain a foundational understanding of both web scraping and facial recognition technologies, as well as the context, objectives, and limitations of the tool. The important points of the state-of-the-art web scraping discussion (see Section 3.5.3) will also be reproduced in the guide for quick reference.

Intended use The purpose of the script is to scan publicly available websites to detect the presence of individuals listed in a predefined facial database. However, it is important to consider how to manage the outcomes of such detection. While it is technically feasible to compile all results into a single comprehensive report, doing so over a wide scope (e.g., crawling the open web) may yield an excessive number of findings, leading to information overload. Furthermore, if the review process is handled by a single individual or centralized body, this may introduce bias or conflicts of interest.

To mitigate these risks, we recommend that the script be executed in smaller, controlled batches, and that the resulting reports be distributed directly to the relevant individuals. This decentralized approach empowers the data subjects to assess the findings and determine appropriate actions for themselves. The individuals should also be trained in how to use the tool, but also be made more aware of the privacy risks that come with our online lives, and be able to ask questions on these critical subjects to really grasp the consequences. It may seem like a stretch at first to imagine someone tracking our location from pictures, but when considering the amount of photos uploaded per year it sounds more plausible.

Consent and awareness Crucially, explicit consent from all individuals included in the facial recognition database is required prior to deployment. To remain compliant with the General Data Protection Regulation (GDPR) and to uphold fundamental principles of cybersecurity, the database must be secured and encrypted. The use of an unprotected directory structure—as presented in this proof-of-concept implementation—is insufficient and poses a significant privacy risk. Without adequate safeguards, the tool could be misused to facilitate facial recognition surveillance against the very individuals it is meant to protect, or even contribute to the generation of deepfakes and other forms of synthetic identity abuse.

In summary, ethical deployment of this system requires informed consent, robust data protection, and responsible operational practices.

Chapter 6
Implementation & Testing

This chapter details the development process, environment setup, and testing of our facial recognition prototype, as described earlier. The implementation is broken down into logical components to ensure modularity and simplify debugging and performance evaluation. Furthermore, the code was clarified after each phase, to ensure it remained clear and easy to modify and improve later on, with proper commenting, file and function separation. This means the code will be easier to maintain and improve.

Another key objective in the implementation of this tool was to ensure ease of use, even for users without programming expertise. To this end, the system was designed to minimize the need for direct interaction with the underlying code. Usability is facilitated through comprehensive installation guides, inline instructions displayed during execution, and support for command-line arguments following standard conventions. Users can invoke the `-help` flag to receive additional contextual information on script usage. The tool's workflow is streamlined, requiring manual interaction with only two primary scripts. Additionally, a detailed 'README.md' file is provided, offering clear, step-by-step instructions for both Windows and Linux platforms, thus enhancing accessibility and promoting wider adoption.

6.0.1 Methodology

Foundational knowledge was gained through a combination of online courses and reputable websites, which will be properly referenced, including this course to acquire the necessary basis to understand facial recognition in python, with openCV [33]. Additionally, review of official documentation, along with the use of Visual Studio Code's autocomplete and function insight features supported the development process.

ChatGPT was used as an aid for debugging and to generate initial drafts of code segments.

6.1 Planning

The project was developed incrementally to ensure each component functioned independently before integration. The planned development stages are as follows:

1. **Face Acquisition and Pre-processing:** Collect facial images and convert them into a machine-readable format. This is accomplished using the OpenCV library and converted with InsightFace.

2. **Local Testing:** Perform facial recognition tasks on local images to validate recognition before incorporating any online data or APIs.

3. **Script Customization:** Implement script modularity, allowing users to switch between different models, adjust granularity, and process video input in addition to images.

4. **UI and logging:** Add UI functionalities to analyze the pictures for which we are not sure, and include a proper logging system to keep track of operations and make testing easier.

5. **Web Integration:** Add web scraping functionality and deploy the full tool. Performance will be measured in terms of image processing time (excluding download latency, as it can vary depending on proxy usage and network conditions).

This stepwise methodology allows for systematic testing, making it easier to isolate issues and verify the success of individual components.

6.2 Setup, Requirements, Environment, Tools & Materials

The tool was developed and tested primarily on Kubuntu 24.04 and Windows 11, using a Samsung Galaxy Book2 4050 laptop with the appropriate nvidia graphics drivers installed, 16gb of RAM, and using intel processor. Note that webcam functionalities were unavailable on this version of Kunbuntu due to lack of appropriate drivers, and the setup for GPU usage was only done on Linux.

This section outlines the necessary setup steps and required dependencies for running the script.

6.2.1 System Requirements

Before proceeding, ensure that the following tools are installed on your system:

- Python 3.x
- pip (Python package manager)
- CMake

Installation instructions vary depending on the operating system and are included in the readme.md of the code.

6.2.2 Virtual Environment Setup (Recommended)

To maintain dependency isolation, it is recommended to use a virtual environment:

```
sudo apt install python3-venv python-is-python3
python -m venv venv
source venv/bin/activate
```

6.2.3 Package Installation

Install the required Python packages using the following commands (the second one is only for linux):

```
pip install opencv-python numpy insightface onnxruntime imutils bs4 requests
sudo apt-get install python3-tk
```

Alternatively we can use `pip install -r requirements.txt`

Facial recognition models are provided with the script. However, alternate models can be specified as arguments if needed.

6.3 Implementation Overview

This section presents the implementation details of the main components in the project.

6.3.1 Script and Folder Structure

The project structure is organized as follows:

- **offline_files**/: Contains 38 test images and one video used for offline experiments.
- **protectees**/: The facial database of individuals to be protected. This folder should be encrypted and access-controlled. It also stores audit logs and detection events.
- `results_YYYYMMDD/`: Results are stored in date labeled folders. By default, each run gets a unique subfolder. Face matches are placed in the *found/* subfolder; unidentified detections can optionally go to a different *miss/* folder. When the option is turned on, all detected face crops are stored in *detected_faces/*. Videos receive their own named subfolder, that follow the same logic but are separated for clarity. Finally, these folders include `log.txt` files that detail the proceedings to be reviewed later, and `audit.txt` files that detail the pictures needing auditing.
- **face.py**: Used to populate the protectees/ database via webcam, one person at a time. It captures 50 face images by default.
- **main.py**: The main script entry point. Accepts arguments for model selection, online/offline modes, urls to scan, and more.

  The following are used by the main script during execution:

  - **settings.py**: Handles all the settings for the main script and their explanation, and will be used to parse the arguments used to run it.
  - **ui.py**: Contains the audit interface, shown at the end of the results .
  - **scraping.py**: Has a few functions necessary for web scraping whilst taking *robot.txt* rules into account. The root of the folder also includes a `scanned_url.txt` file that details the urls that have been scanned by this script.

6.3.2 OpenCV Integration

OpenCV is utilized as the core library for image processing, video capture, and display operations. Key functions include [5]:

- `cv2.imread()`: Load images from disk. A similar function exists for videos.
- `cv2.imshow()`: Display the images. We use this to show the progress of the main script and the facial capture of `face.py`.
- `cv2.imwrite()`: Saves images to the disk at the specified location, after desired modifications such as extra annotations and cropping.

OpenCV's ease of use and pre-trained model support make it ideal for rapid computer vision prototyping.

6.3.3 Initial Face Capture

Face detection is initialized via a pre-trained model. Initial trials using `haarcascade_frontalface_default.x` proved insufficient, particularly in cases of partial occlusion (e.g., hair, head angles). As a result, the more robust `yunet.onnx` model was adopted, offering improved performance and real-time detection.

Model Initialization

The `yunet.onnx` detector is initialized as follows:

```
face_detector = cv2.FaceDetectorYN.create(
    "yunet.onnx", "", (640, 480),
    score_threshold=0.6, nms_threshold=0.3, top_k=5000
)
```

Parameter Description:

- `"yunet.onnx"`: ONNX model for face detection using OpenCV.
- `(640, 480)`: Detection window size. Faces larger than this may not be captured entirely.
- `score_threshold`: Minimum confidence for a valid detection.
- `nms_threshold`: Non-Maximum Suppression threshold to eliminate overlapping boxes.
- `top_k`: Limits the number of bounding boxes processed; 5000 is a safe upper limit.

The detector is applied to frames captured via `cv2.VideoCapture(0)`. A rectangle is drawn around detected faces, and images are only saved every 10 frames to avoid redundancy.

Captured faces are resized to 150×150 pixels whilst maintaining aspect ratio (by adding white borders) in order to improve the facial recognition capabilities, as they perform better under a consistent format.

Two individuals were recorded under consistent lighting, with and without glasses, resulting in 50 images per person for training and validation purposes. This number was deemed necessary due to many unpredictabilities in the capture process, mostly due to operator movements.

6.3.4 Basic Facial Recognition

This section describes the second stage of the system, which evaluates whether any of the protected individuals appear in the analyzed images. It continues to use `yunet.onnx` for face detection and introduces InsightFace for face recognition.

While the detection model remains the same, its parameters are adjusted slightly for this task. In particular, the detection resolution is reduced to 320×320 and the confidence threshold is set to 0.6 but can be adjusted further if too much effort is spent on faces. Furthermore we have added a size check for faces smaller than 32×32 as we deem them too small for accurate recognition.

The InsightFace recognizer is initialized as follows:

```
app = FaceAnalysis(name="buffalo_l",
    providers=["CPUExecutionProvider"])
app.prepare(ctx_id=0, det_size=(320, 320))
```

The `"buffalo_l"` model was selected for its balance of accuracy and efficiency in real-time settings, and for being well-supported within the InsightFace framework.

Model Performance and Similarity Metrics

The recognition process computes the similarity between detected faces and those stored in the reference database. Both input and reference faces are transformed into 128-dimensional embedding vectors. These embeddings encode facial features and enable comparison.

Two common similarity metrics are [42]:

- **Cosine Similarity**: Measures the angle between two embedding vectors.
- **Euclidean Distance**: Measures the straight-line distance between two vectors.

In this implementation, we adopt cosine similarity, which normally gives a value between -1 and 1, normalized to a percentage scale to improve interpretability:

```
similarity = ((np.dot(embedding, ref_embedding) /
               (np.linalg.norm(embedding) *
                  np.linalg.norm(ref_embedding))) + 1) * 50
```

This calculation returns values between 0% and 100%, more easily interpretable. A configurable threshold determines when two embeddings are considered to represent the same individual.

6.3.5 Picture Annotation

Each detected face is annotated on the image using a color-coded bounding box, according to its similarity to the most likely match in the database:

- **Red**: Similarity score below 60%. No further action is taken, as the likelihood of a match is low.
- **Green**: Similarity score between 60% and 70%. The detection is subject to human auditing before any notifications are sent.
- **Blue**: Similarity score above 70%. These are considered strong matches and flagged for review.

In all cases, the annotation displays the identity of the individual in the database with the highest similarity score. For simplicity, we do not currently account for potential matches with other individuals in the database who may have slightly lower similarity values.

This process ensures that protectees are only alerted when the resemblance is significant, reducing the likelihood of false positives while maintaining a human-in-the-loop validation strategy.

6.3.6 Dataset

Facial recognition was evaluated using a curated dataset of 38 images. Within this dataset, **Protectee One** appears in 20 images, and **Protectee Two** in 18. Several images contain both individuals, while others feature unrelated subjects, including family members and strangers. The inclusion of family members is intentional, as it introduces subtle similarities that may challenge the recognition algorithm and better simulate real-world conditions from a limited dataset.

While a larger dataset would allow for more robust statistical conclusions, it would also complicate manual analysis. For this reason, we reserve large-scale testing for the web scraping phase. Not focusing too much on our own database should also helps prevent overfitting by avoiding over-adjustment of recognition thresholds to a limited dataset. Once deployed, the system's ability to process high volumes of images will naturally expose any false positives and inform future tuning.

Initial recognition was performed using the InsightFace model, which demonstrated promising performance:

- Both protectees were correctly identified in nearly all cases, including challenging scenarios such as reflections in mirrors, variable lighting conditions, different facial angles, partial occlusion (e.g., facial mask, sunglasses, clay mask), and even older photographs.

- No false positives were recorded when a similarity threshold of 60% was applied with the InsightFace model.

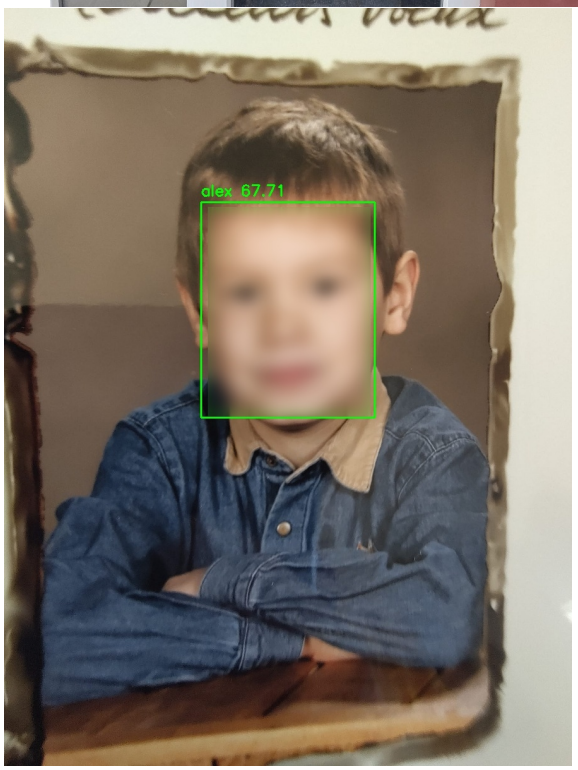
- The average runtime to analyze the full dataset of 38 images was approximately 20 seconds (201.5 seconds over 10 trials).

These results suggest that, on this dataset, the `Yunet` detector reliably identified facial regions, while the `InsightFace` recognizer achieved a high degree of precision. Notably, initial iterations of the algorithm failed to detect certain faces, but performance improved significantly after resizing images to standardized dimensions.

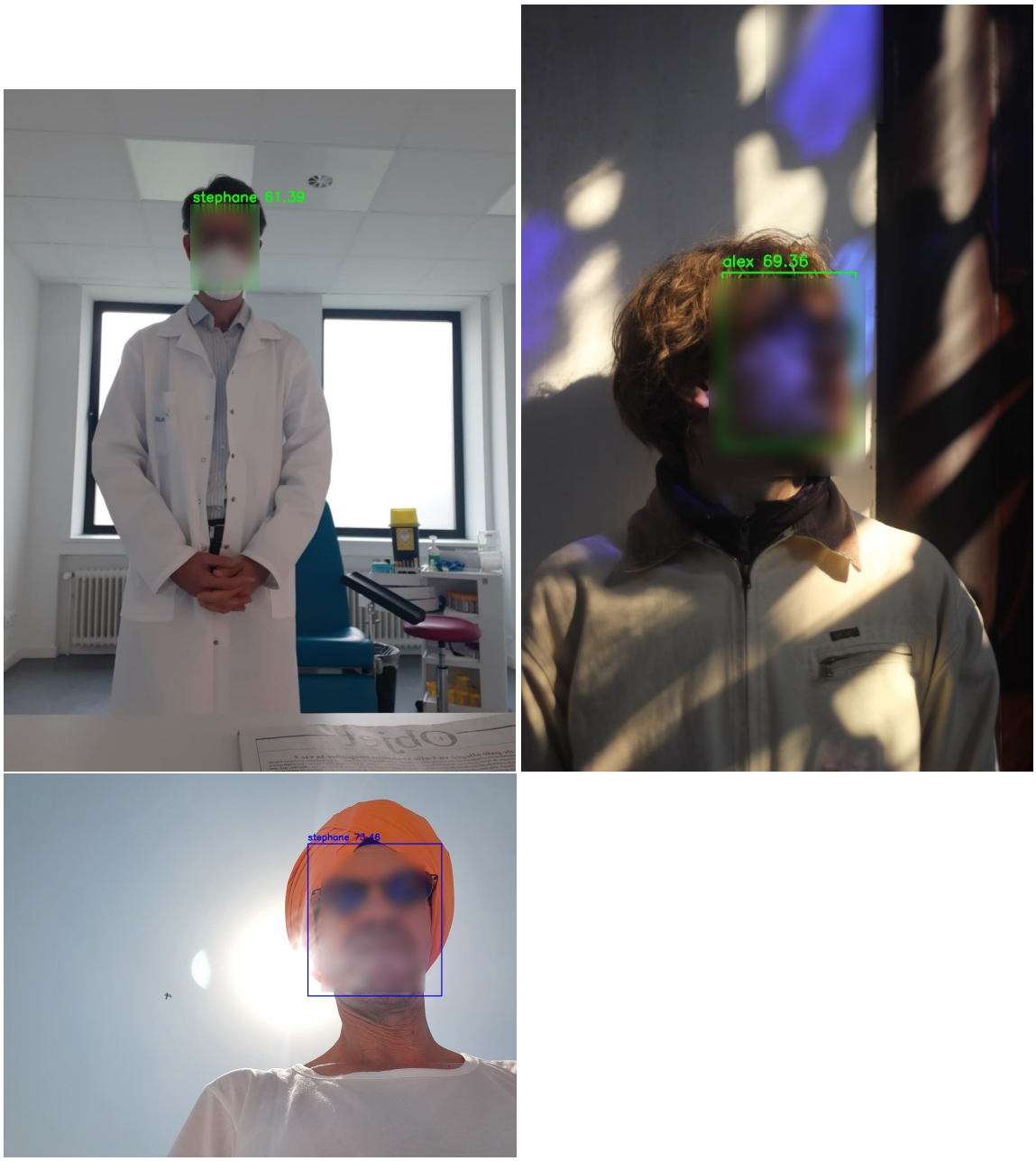
The most interesting test cases highlight the model's robustness. The first test image required longer processing, likely due to complexity. Subsequent cases confirmed that the system could accurately recognize individuals under a wide range of real-world conditions—including significant age differences, obscured features, and unconventional appearances. All faces in visual outputs were blurred to protect the privacy of the individuals involved.


alex 79.32


alex 72.78


alex 67.71


alex 74.98



Pre-processing Effective pre-processing is critical for both the accuracy and consistency of facial recognition. This applies to both the database images (reference data) and the images being analyzed.

We resize the images to a standard squared format (by default 800x800), which is also the display window size. Higher sizes will lead to more recognitions by yunet, but we found current performances to be sufficient. Resizing the pictures in the facial database to 150px and the ones to analyze to 800px caused more detections and improved recognition, going from, from the logs "alex: 28 times stephane: 16 times detected, and 54 unknowns" in 25 seconds to "Alexandre: 49 times Stephane: 16 times detected, and 73 unknowns" in 28 seconds - many of the subject one's faces were found within a video that is now properly resized, but more subjects were detected overall as well. The accuracy seems to have improved as well, but it would be unwise to draw too many conclusions from such a limited dataset as we would risk overfitting it. Tests with a larger controled dataset would help determine the preferred scanning parameters.

6.3.7 Logging System

After preliminary testing of the recognition components, a logging system was implemented to store execution data and support further testing. This system is also intended to be a core feature of the final deployment.

Recognition results are saved within a folder named `results_{YY-MM-DD}`, corresponding to the current execution date. If such a folder already exists, a subdirectory with an incrementing numerical suffix (e.g., `results_{YY-MM-DD}/1`, `/2`, etc.) is created to store the new results.

Within these folders, the system stores the analyzed images, annotated with rectangular overlays highlighting detected faces. Each rectangle is labeled with the predicted identity (if applicable) and the similarity score. Only matches exceeding a 60% similarity threshold are logged in this manner, other pictures are instead stored in the textttmiss/ folder if desired.

If a detected face matches an entry in the protected facial database with high confidence (above a 70% similarity threshold by default), the corresponding image is also copied into a dedicated subfolder associated with that individual. Additionally, an entry is appended to the individual's personal log file, detailing the detection event and relevant metadata. In a deployment scenario, the protectee should be notified and allowed to decide how to handle this information.

Each execution folder includes a `log.txt` file containing a summary of detections and processing metadata. This includes timestamps for the beginning and end of execution, as well as the total processing time. Optionally, these details can also be displayed in real-time in the terminal, controlled via a script variable.

In cases where multiple individuals are recognized in the same image, each relevant protectee receives a copy of the picture within their personal result folder. For privacy, only their own face is marked and identified in these copies. Although individuals may recognize each other from context, this design choice ensures data minimization and individual privacy—especially important in scenarios such as corporate environments, where this tool may be scaled and used across broader teams.

6.3.8 Audit System

An audit mechanism was integrated into the script to enhance traceability and accountability during the face recognition process.

During execution, images are displayed in real time as they are processed. Following this initial stage, an audit window is triggered, prompting the user to input their name. This input is recorded in an `audit_logs.txt` file, providing a traceable record of all interactions with the system.

Once a name is entered, the audit interface is displayed, as shown in Figure 6.1. The interface allows users to navigate through processed images using arrow keys or on-screen buttons.

Users can confirm the presence of individuals with an accuracy above 50% in images by clicking on a detected face and then selecting the "Confirm" button. Upon confirmation, a notification email is sent to the identified individual, mimicking the process followed for high-confidence matches (i.e., similarity above 70%). However, in this case, the notification explicitly notes that the identification was confirmed via manual audit. All actions performed through this interface are logged.

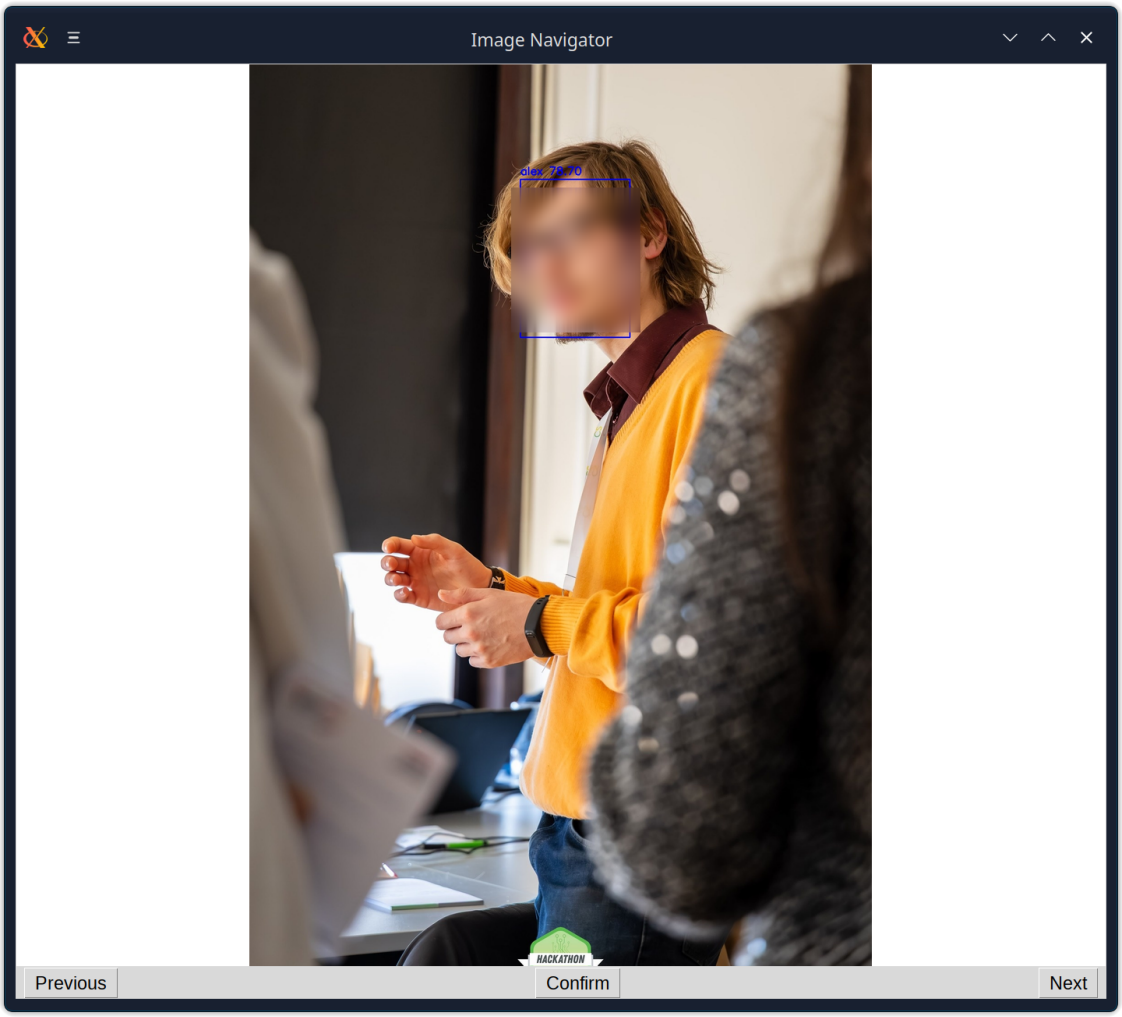


Figure 6.1: Audit GUI for manual validation of detected faces.

To ensure continuity, the system maintains an `audit.txt` file that tracks which images have already been audited. This allows the script to be rerun with auditing enabled and picture analysis disabled—useful for situations where human validation is performed separately.

The audit feature can be disabled entirely by passing the `-noAudit` argument when executing the script.

6.3.9 Threading and Performance Considerations

To optimize performance during image processing, we explored the use of concurrent execution, with threads or in this case through multiprocessing with InsightFace and yunet. This was motivated by the relatively long runtime associated with scanning large batches of images during the offline tests. We used Pool, from the multiprocessing library, and its pool.map function to launch the functions. This function takes a list of arguments to give to the function that each correspond to a process to be run indenpendantly.

The following results summarize the runtime (in minutes:seconds) when scanning the local dataset of 38 images four times using different numbers of parallel processes:

- **1 thread:** 1:38
- **2 threads:** 1:48
- **4 threads:** 1:51

However, this did not seem to yield performance improvements. On the contrary, results indicated a slight degradation in processing time, likely due to the overhead introduced by initializing multiple face detectors concurrently.

This limited empirical evaluation suggests that multiprocessing is not beneficial in the current implementation, at least on windows upon execution of the code. The most likely reason is due to the CPU-heavy nature of insightface leaving little resources for other threads. Other potential explanations include inefficient thread/process management or resource contention when initializing multiple instances of the recognition models.

Take-away The code was nevertheless adapted to be able to support multi-processing at this early stage, which will prove useful later on. This was done by moving the main code into a `if __name__ == '__main__':` section, and giving all variables to the various functions, instead of relying on them being available globaly. Windows causes more issues with this, as the processess do not have access to global constants or variables. Indeed, multi-processing is not the same on windows as it is on linux, as linux processes will also run any code found in the global scope of the script, as is detailed more here [12].

These changes in this version of the script will make it easier to add multi-processing later for the final product.

Core Functional Modules

The script provides several key features that can be toggled independently:

- **Face Detection and Recognition:** Users can select between YUNet, InsightFace and OpenCV models for both detection and recognition.

- **Media Input Modes:** The script supports image directories or video streams as input sources.
- **Audit and Logging:** Optional modules include an interactive audit interface and logging system to review or validate detections.
- **Offline vs Online Modes:** The script can operate either with pre-downloaded files or scrape online sources.
- **Web Scraping Depth:** Users can specify a list of URLs and how extensively each should be scanned.

Design Philosophy To accommodate a variety of operational contexts, the script was developed with strong modularity and configurability in mind. It is designed to support both testing and deployment scenarios through a flexible set of command-line arguments, allowing fine-grained control over performance, thresholds, model selection, and behavior. The codebase was built to be easily maintainable and extensible.

This configuration model ensures reproducibility, scalability, and adaptability across a wide variety of environments and use cases.

Command-Line Arguments Overview

This argument class, named args, is used to pass the arguments around to the functions that require them. This gives a lot of freedom without having to modify the script, and allows the code to function with multi-processing under windows, where global arguments from the main code instance are not accessible.

Key settings All of these arguments are provided to the main script as follows: `main.py -argument`, for non boolean variables the argument will be followed by an = and the desired value. For instance giving a custom url list can be done using: `main.py -urlList=https://site.com,https:`

We recommend using `-scanDepth=X` where X is the desired depth in order to control the desired depth (and thus time) of the search. Scanning with a depth of 0 and consequently one of 1, or even 2 or more will not reScan the previously scanned pages, unless the option `-reScan` is specified. `-help` can be used to get the above table when executing the script.

For testing purposes one can use `-offlineTests`, which will not do any online querying and thus allow testing only the recognition part of the code. `-incrementingFolders` can then be used to keep traces of all executions, and we can use `-showAll` in order to display all pictures, even the ones that do not have a potential protectee in them.

The arguments also allow changing the models provided and the folders to use, and audit and log file names, with the goal of facilitating automatic tests that would run independently or consequently without human interaction.

State of the art note

Yolo We had considered Yolo for recognition, however this would mean requiring installing torch for users, and taking at least a gigabyte of additional space whilst also making the script more complex to install.

Furthermore, yunet detection was deemed satisfactory both in terms of speed and precision for our use case, as during testing it found all but a few faces, and InsightFace takes more processing time.

Argument	Description	Default Value
Thresholds & Display		
-mat, -minimumAuditThreshold	Minimum similarity score to allow auditing	60
-st, -similarityThreshold	Threshold to consider face detected	60
-nt, -notifiedThreshold	Threshold to notify individuals automatically	70
-mfs, -minFaceSize	Minimum face size in pixels	32
-igf, -ignoreFrames	Frames to skip in video input	30
-w, -displaySize	The size of display window	800
-sd, -scanDepth	The depth of the scan, how many links into the website it goes	1
Boolean Flags		
-nd, -noDisplay	Disable display during execution	True
-storeFailed	Store failed detection images	False
-sa, -showAll	Show all results, not only matches	False
-sp, -skipPictures	Skip processing pictures	False
-sv, -skipVideos	Skip processing videos	False
-if, -incrementFolder	Increment result folder if needed	False
-na, -noAudit	Disable audit system	False
-ot, -offlineTests	Run on local files only	False
-sr, -terminalResults	Show detailed terminal output	False
-so, -reScan	Re-scan images even if already scanned	False
-sf, -saveFaces	Save cropped face images	False
Recognition Model Selection (exclusive)		
-openCVRec	Use OpenCV recognizer	False
(default)	Use InsightFace recognition	True
Paths and Filenames		
-faceDatabase	Path to folder with protectee face images	protectees/
-offlineFiles	Directory with offline media to test	offline_files/
-foundDir	Output folder for successful detections	found/
-missDir	Folder for missed/false detections	miss/
-detectedFaces	Folder for detected face crops	detected_faces/
-logFile	Main log file name	log.txt
-auditFile	Audit trace log filename	audit.txt
-scannedUrls	Record of scraped URLs	scanned_urls.txt
Detection Model Settings		
-yunetDetector	Path to YUNet model	models/yunet-
-detSize	Detector window size (w,h)	(320, 320)
-scoreTreshold	Minimum detection confidence	0.7
-nmsTreshold	NMS overlap suppression value	0.3
-topK	Max faces returned per image	5000
-opencvRecognizer	OpenCV recognition model path	models/face_rec-
-insightFaceModel	InsightFace model name or ID	buffalo_l
Web Scraping		
-urlList	Comma-separated list of URLs to scrape	See below.

Table 6.1: Overview of script arguments and default settings

Dlib We choose not to include Dlib in the final solution since it required many conditional branches within the code and more complications, as well as being bad in terms of diversity [25, 45].

OpenCV for Recognition

In addition to InsightFace, we experimented with OpenCV's default facial recognition model. The primary advantage of this model is its significantly higher speed—up to four times faster in tests. However, this comes at a cost: the OpenCV model tends to produce more false positives and misidentifications.

As a result, although the recognition step itself is faster, the increased volume of incorrect detections requires more human intervention during auditing, which can negate the time gains. For this reason, we recommend adjusting similarity thresholds upward when using OpenCV for recognition to mitigate the higher error rate. Users should consider this trade-off when choosing between speed and precision.

Video Face Recognition

Since videos are essentially sequences of images, we are able to process them using OpenCV by treating each frame as a standalone image. To optimize performance, the system performs recognition on every 10th frame by default—roughly once per second for most videos.

Video analysis introduces additional challenges compared to still images. Factors like motion blur, compression artifacts, and poor lighting can reduce recognition accuracy. For example, in one test, a subject was incorrectly identified as another due to low lighting and movement, particularly when using OpenCV's recognition engine.

While video offers a large pool of frames to analyze, this volume increases processing time and can exacerbate the impact of misdetections. In our tests, the OpenCV recognizer proved too imprecise for reliable video analysis, often resulting in excessive false positives. For improved accuracy, InsightFace remains the preferred option in this context, despite its slower performance.

Additionally, OpenCV tends to automatically rotate videos to match typical display orientation, which may affect alignment and should be accounted for when interpreting results.

6.3.10 Web Scraping

The final phase of development introduces web scraping to gather publicly available images for use in facial recognition testing. This addition brings the prototype closer to real-world applications, such as online identity monitoring or privacy auditing. To do this we have followed instructions for basic functionalities to build upon. [6]

Since image download speeds vary widely depending on website responsiveness, proxies, and rate limits, scraping times are excluded from performance benchmarks. Instead, we focus on measuring recognition speed and logging results to assess practical scalability, although we will mention scraping speed to give an idea of the time and resources required.

Scraping is conducted using a pre-approved list of websites, chosen both for relevance and to comply with ethical and legal considerations. The script automatically checks each domain's robots.txt file to respect scraping policies, since multiple websites now explicitly block scraping, and larger platforms typically require official API access. Respecting the robots.txt is not obligated but is considered good practice, and they should reflect the

Terms of Service which we do have to respect or risk legal consequences. We used this script used to verify legal use:

```python
def ScrapingAllowed(url, log, terminal_results, user_agent='*'):
    try:
        if(LikelyPrivate(url)):
            Logs(log, f"[STOPPED]Likely private page: {url}",
                terminal_results)
            return False
        parsed = urlparse(url)
        base_url = f"{parsed.scheme}://{parsed.netloc}"
        path = parsed.path or "/"  # Only the path, not the full
            URL

        robots_url = f"{base_url}/robots.txt"
        rp = RobotFileParser()
        rp.set_url(robots_url)
        rp.read()

        allowed = rp.can_fetch(user_agent, path)
        if not allowed:
            Logs(log, f"[BLOCKED] robots.txt disallows: {path}",
                terminal_results)
        else:
            Logs(log, f"[ALLOWED] robots.txt allows: {path}",
                terminal_results)
        return allowed
    except Exception as e:
        Logs(log, f"[WARNING] robots.txt could not be checked
            for {url} (reason: {e}) - defaulting to allowed",
            terminal_results)
        return True
```

Another part of the script warns when the main url is not accessible for scraping or invalid, and advises removing it from the list.

To avoid excessive data storage, the script saves only images with a similarity score above the auditing treshold, which is of 60% by default.

We tried scanning the addresses present in the `settings.py` file for our 2 protectees, finding results between 45 and 61%, although after auditing it was found that none matched their identity. The list contains various news websites as well as blogs, research sites, universities, cultural centers and more, generated by chatgpt. It included multiple belgian websites, but about half of them denied scraping.

- https://www.bcg.com
- https://www.cnn.com
- https://www.bbc.com
- https://www.nytimes.com
- https://www.tumblr.com
- https://500px.com
- https://www.wikipedia.org
- https://commons.wikimedia.org
- https://www.ieee.org
- https://www.hln.be
- https://www.rtbf.be
- https://www.vrt.be
- https://www.bruzz.be
- https://www.vtm.be
- https://www.7sur7.be
- https://www.student.be
- https://www.ugent.be
- https://www.uclouvain.be
- https://www.ulb.be
- https://www.vub.be
- https://www.muntpunt.be
- https://www.kvs.be
- https://www.kaaitheater.be
- https://www.debijloke.be
- https://www.culture.be
- https://www.flandersartsinstitute.be
- https://www.passaporta.be

- https://www.visitflanders.com
- https://www.arenberg.be
- https://www.kmska.be
- https://www.ccdeadelberg.be
- https://www.cultuurcentrummechelen.be
- https://www.mac-s.be
- https://www.decentrale.be
- https://www.fomu.be
- https://www.wiels.org

First tests Scanning the full list with a depth of 0 (only the main pages), multi-processing and no displaying of results took 616.03 at first, with shared detectors and recognizers, including 436.17 of scanning the various urls for pictures and videos, with the rest spent on facial analyzis. This is reasonable, especially considering the `time.sleep(1)` we are using to not overwhelm websites. Multiple exceptions and errors were raised during the execution of the code online, and all were addressed. Due to the nature of the script it was possible to continue the process from there, but we choose to rerun it from scratch. This search yielded 5 candidates for facial recognition above 60%, but during auditing it was found they were not in fact the protectees.

The depth of 1 changes a lot, since this is not a depth in terms of web pages as one might expect. Instead, all links present in the first home page will be accessed and checked for pictures too.

At first we only scanned the main pages for testing purposes, but after making a recursive search in the urls we have added a log of all of the scanning being performed, so we can show our output and the code that lead to a page being considered allowed to read or not. We show the permissions of access for every path we scan.

Multi-processing

In this context of web scraping, multi-processing makes sense, as even if it does not bring faster recognition, it will allow spreading the search and accelerating it to the limits of the current computer and speed of download of the various websites, instead of exploring them fully one by one.

We also added multi-processing to the web scanning phase. To do this, we used this quite code, which we will detail as it is not easy to read:

```
# The list of arguments provided to the script.
info = [
    [url, args.results_folder +
       urlparse(url).netloc.replace(".", "_") + "_" +
       "wlog.txt", args.scanDepth, args.scanned_urls,
       args.skipPictures, args.skipVideos, args.terminalResults,
       not args.noThreading]
    for url in args.urlList
]
if(not args.noThreading):
    with Pool(cv2.getNumberOfCPUs() - 1) as pool:
        i = 0
        for result in pool.map(RecursiveScrape, info):
            if(result):
                # The results to analyze, the arguments and the
                   third element is for file names.
                text_files_path = info[i][1][:-8]
                to_scan.append((result, args, text_files_path))
            i += 1
```

Parallel Web Scraping To perform web scraping in parallel across multiple URLs, we begin by constructing a list of arguments required by the scraping function to analyze each website. This structure is necessary because the `pool.map()` function accepts a single function and a list of argument sets, applying the function to each element in parallel. Therefore, all required parameters for each URL must be encapsulated within a single argument per entry in the list. While the `map()` function is used in the current implementation, future work could explore the use of `map_async()` [10], which may offer performance benefits at the cost of added complexity.

Ethical Considerations and Rate Limiting It is crucial to approach web scraping with caution, especially when employing efficient parallel techniques that may inadvertently send large numbers of requests in a short period. Excessive or aggressive crawling can lead to IP blocking or raise legal concerns. To mitigate this, a fixed rate-limiting mechanism is implemented using `time.sleep(1)` between requests. This delay is hard-coded rather than exposed via command-line arguments, intentionally discouraging misuse and promoting responsible scraping behavior. Additionally, the script includes validation checks to ensure that provided URLs are both syntactically correct and permit scraping according to standard web protocols.

Implementation Details The scraping process is parallelized using a multi-threaded approach, wherein each thread executes the `RecursiveScrape` function to extract links to publicly accessible images and videos. Each thread receives its own parameter set from the prepared list. To maintain isolation and clarity, each process writes to its own log file, which are subsequently merged once the scraping is complete. However, it is worth noting that this approach may encounter limitations on systems with constrained memory resources.

```
with Pool(cv2.getNumberOfCPUs() - 1) as pool:
        info = [
                [url, results_folder +
                   urlparse(url).netloc.replace(".", "_") + "_"
                   + LOG, scan_depth, scanned_urls,
                   skip_pictures, skip_videos, terminal_results,
                   not no_threading]
                for url in url_list
        ]
        for result in pool.map(RecursiveScrape, info):
                if(result):
                        to_scan.append(result)

        filenames = [text_file for text_file in
           os.listdir(results_folder) if text_file[:8] ==
           "wlog.txt"]

        scan_log = open(results_folder + "scan.txt")
        for fname in filenames:
                with open(fname) as infile:
                        scan_log.write(infile.read())
                        os.remove(fname)
```

Multiple tests were conducted scanning the whole url_list with a depth of 0, with

different numbers of threads.

- 1: 375, 368
- 2: 298, 301
- 4: 290

Considering the powerful computer used for the tests, we set the default amount of threads to 2 to avoid overwhelming smaller computers.

Improvements made There were freezing issues when multiple threads tried to access the display window at the same time, which was resolved by forcing the window to be off in these cases. We will note that it is always possible to view the saved pictures with the audit window or even a standard image viewer.

Another issue was the fact that windows handles threads in a different way, making them truly autonomous and unable to access global variables. To combat this the class argument is passed throughout the code (python will handle it as a reference and not copy it needlessly). The detection and recognition algorithms could also not be shared, so a new function was made to generate them for multiple website scans at a time.

The different text files also had to be handled independantly to avoid conflicts in the writing and because passing them through the argument calss was not a stable solution, due to how many threads might need to access it at the same time. The different files are then added by the program at the end of the existing log, audit or urls_scanned txt files, before the various temporary files are deleted. To simplify their identification their names end with predefined 8 characters: "plog.txt", "zcan.txt" and "audi.txt".

We also included a count of the amount of pictures scanned and the faces therein.

GPU use It is possible to use the GPU with InsightFace, and drastically improve performances. This was tested with an nvidia gpu and intel. To do this we need to follow this procedure [31]:

```
pip uninstall onnxruntime-gpu onnxruntime
pip install onnxruntime-gpu --extra-index-url https://aiinfra.pkgs.visualstudio.com/
    PublicPackages/_packaging/onnxruntime-cuda-12/pypi/simple/
```

To setup the right version of onnxruntime to use with CUDA-12, which is the nvidia tool we use for gpu usage. This tool has to be installed on their website, as well as cuDNN for the script to work.

We will detail the results later, but note this was only tested on Linux and using the GPU is not a requirement for running the script.

We ran tests using the GPU (but not the CPU at the same time, which could be improved later). The local dataset tests went from 24 to 7.5 seconds, and increase of about 300%! The depth 0 search went from 250 to 150 seconds, which is an increase of 72%! and the depth 1 search only took a total of XX - although due to the significant time required to run this command, this was not done on the same day, meaning we should not draw too many conclusions from this at this stage. The difference may be due to the full scanning not always scanning pictures, but this should be investigated further.

We added a count of how many images and faces were scanned to make it easier to analyze and compare performances in the future. This was only done at this stage because of the complications that come with multi-processing.

6.3.11 Final Tests

Significant improvements were made to the codebase, particularly in optimizing URL search procedures and addressing bugs that previously prevented comprehensive image scanning. As a result, the system now completes the scanning process in 14 seconds and the recognition stage in 312 seconds (at depth 0), solely by scraping images directly from the main webpage of each site. The script was executed using two concurrent threads, with CPU usage ranging between 65% and 100%, often remaining close to the upper bound. To improve resource management, the script was refactored to reduce the number of simultaneously instantiated detectors and recognizers. Instead of creating one instance per website, the new implementation distributes websites across a fixed number of processes in batches. For example, with four websites and two processes, each process handles two sites. By default, each process is configured to manage ten websites.

Depth 1 Search Executing the script at a search depth of 1 resulted in a substantial increase in runtime due to the exponential growth of links encountered in recursive web crawling. The process took 13,034 seconds (approximately 3 hours and 37 minutes), with the scanning phase alone accounting for 1,090 seconds (about 18 minutes). Initially, both threads were heavily utilized, but one process completed its queue earlier, leaving a single thread to handle the remaining workload. This imbalance suggests that increasing the number of threads may improve performance by distributing tasks more evenly. Notably, this approach remains more resource-efficient than instantiating a facial recognition model per thread.

Images are not stored locally, primarily due to ethical and storage considerations. Retaining images indefinitely may conflict with privacy objectives, especially if the original source removes them. Despite this, network usage remained modest, with a peak bandwidth of approximately 1 MB/s. As this tool is still in its prototypical stage, images are not automatically deleted. However, for ethical compliance, they should be manually removed once their content has been verified and the relevant data has been extracted for research purposes.

Depth 1 Search Results The initial execution of the depth 1 search yielded 64 images, of which 26 were duplicates. Some duplicates were found across different domains or distinct URLs within the same domain, while 13 duplicates appeared to be queried multiple times from the same source. No consistent duplication pattern was identified, suggesting incidental redundancy in image retrieval. Further investigation may be warranted if the tool is deployed in production environments. Manual auditing confirmed that none of the retrieved images matched the protected identities, with similarity scores ranging from 60% to 64%. These results align with visual inspection and indicate that the system's threshold should be adjusted to reduce false positives. Raising the similarity threshold to 65% would have excluded all of these non-matching results.

Depth 1 Final Version Search The second depth 1 execution used the latest version of the tool, which included image resizing for both database entries and online images. This run took 10,490 seconds, with 1,101 seconds spent on scanning. A total of 78 images were retrieved, including 26 duplicates. Most duplicates were found across different websites, and only two images were common between this run and the previous one, each appearing on four different domains. This suggests increased robustness in retrieval and filtering

across iterations. We should note however that this intense scraping caused some websites, like twitter, to give an "exceeded number of tries" block (simply for asking the robot.txt, we have to assume, from the pages that had twitter links in them). Thankfully, this is only temporary.

6.4 Final Script

This section briefly presents the final version of the script, outlining its modular architecture, available options, current limitations, and directions for future improvements.

6.4.1 Recovery

Given the extensive execution time required for a depth-1 search, the system was designed with built-in recovery functionality. This allows the script to be interrupted and resumed without loss of progress. During execution, metadata such as visited URLs and runtime state are recorded in intermediary text files. Each thread maintains its own file independently, ensuring separation of tasks. Upon restarting the script, the system checks for any previously stored data and resumes from the last known state. These files are then either merged at the conclusion of a full execution or reloaded at the beginning of a resumed session.

6.4.2 Usage Overview

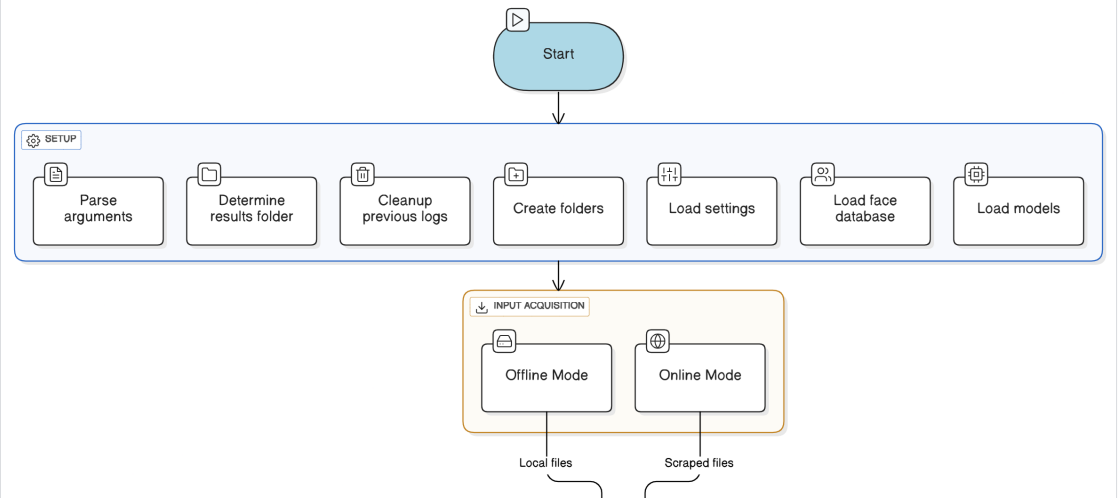
Typical usage of the tool begins with reviewing the `README.md` file, which outlines essential considerations for proper execution. Users are encouraged to familiarize themselves with the basics of facial recognition and web scraping to understand the broader context in which the tool operates.

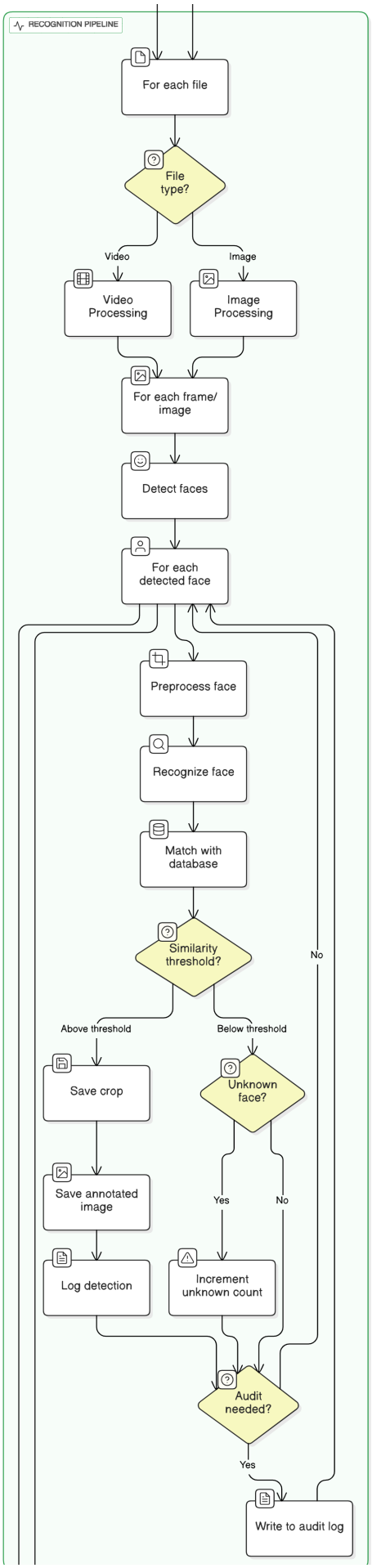
The script's standard workflow involves first executing `main.py`, which acquires facial data for all designated subjects. If this is the user's initial run and the face database has not yet been populated, the script will invoke `face.py` automatically. During execution, the system performs an online search using the default parameters, while real-time visual output is provided through an OpenCV window. Upon completion, and if relevant matches are found, the script will prompt the user to perform an audit.

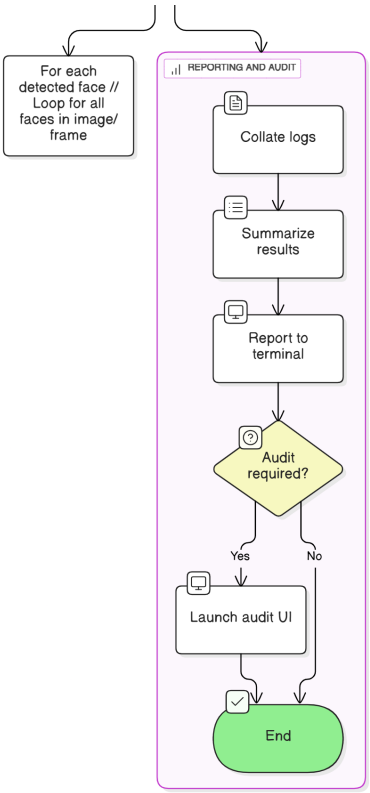
In enterprise or governmental contexts, the usage differs slightly. Organizations may first run `face.py` independently to establish a comprehensive database of protected individuals. Auditing tasks can then be assigned to designated personnel or automated according to predefined thresholds. Upon detection, employees may initiate appropriate procedures to request takedowns or modifications of their online images. Each image is saved with a filename corresponding to its exact online location, simplifying the process of identifying and contacting the hosting website.

6.4.3 Code Execution Graph

We have provided the main code to DiagramGPT which uses AI to generate a clear overview of a typical code execution, present on the bottom of this page and thereafter.




RECOGNITION PIPELINE
For each file
File type?
Video
Image
Video Processing
Image Processing
For each frame/ image
Detect faces
For each detected face
Preprocess face
Recognize face
Match with database
Similarity threshold?
No
Above threshold
Below threshold
Save crop
Unknown face?
Save annotated image
Yes
No
Log detection
Increment unknown count
Audit needed?
Yes
Write to audit log



6.4.4 Configuration

The script supports a wide variety of configurations to adapt to different testing, auditing, and deployment scenarios. Through the use of command-line arguments, users can fine-tune behavior such as threshold sensitivity, detection model, face database paths, and enable or disable modules like auditing, logging, and offline tests. This design enhances both modularity and extensibility, and can easily be expanded further in future iterations.

For a detailed list of available parameters and flags, see Table **??**, which summarizes the script's main options, including recognition models, logging settings, and display controls. The same information can be found from the script itself by running `main.py -help`.

6.4.5 Security Flaws and Limitations

As a prototype, the current version lacks several important security and robustness features, from a cybersecurity perspective:

- **Insufficient fault tolerance:** There has not been sufficient testing of edge cases, and error handling could be improved since some warnings can appear during the scanning.
- **No encryption:** Sensitive data, images and results are stored in plain text. A central database with appropriate encryption and access control should be implemented in a production setting.
- **Logging gaps:** While the script includes basic logging and auditing, a more robust and tamper-proof logging system is needed to support transparency and traceability.

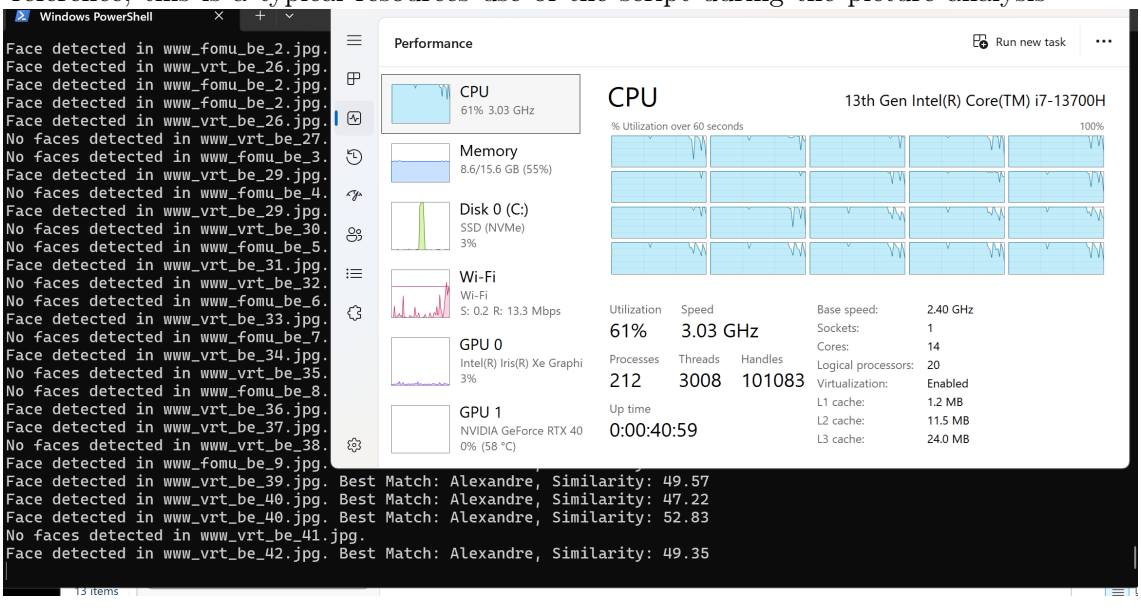
- **No memory protection:** There are currently no defenses against unauthorized memory readings or data leakage. However due to this happening on a local computer instead of the wide web, the annotated pictures cannot be intercepted.

6.4.6 Areas for Improvement

A number of enhancements are planned or suggested for future versions:

- **Model experimentation:** Support additional recognition algorithms to further evaluate speed and accuracy trade-offs, and run more tests.
- **Batch notifications:** Add functionality to notify identified individuals (optionally in batches) when their presence is detected, instead of just putting information in their log file with an image.
- **Multi-user support:** Expand the codebase to include proper login and link to the logs.
- **RAM limitations:** Manage file reading in such a way that computers with limited RAM space can safely use the script.
- **KeyboardInterrupt:** On windows the process is not stopped when using the keyboard interrupt keys (ctrl+c), forcing the user to close the terminal window. This is not an issue on linux, due to how thread handling is different for both.
- **Commenting and code clarity:** Comments, whilst not lacking, could be improved both in number and clarity.
- **Experimentation with threads:** We should try to combine GPU use with the CPU and multiple processes. The map_async function should also be explored.

For reference, this is a typical resources use of the script during the picture analysis phase:



Terminal display improvements and related issues Currently there is some information displayed on the terminal that is of little use to the average user, and warnings that signal potential issues with the code, which, although not significant, should be investigated. Notably, during the web scanning phase of the script we get the, `Some characters could not be decoded, and were replaced with REPLACEMENT CHARACTER.`, as well as another issue with using html reading to read some xml pages, that we choose to ignore for now. The image scanning phase also displays `libpng warning: iCCP: too many profiles` which causes the progress bar to be on different lines. It would also be good to investigate a way to display the results better, as currently it only shows it by website for each different threads.

Finally, when InsightFace runs it displays information which is useful for us to understand if it is using CUDA or a normal CPU provider and more, but is likely to be confusing to the average user.

6.5 Experimentation Conclusions

First of all, we will note that the script was done with legality and ethical use in mind, instead of prioritizing results. It is also robust and modular to allow for further testing and development, and comes with sufficient instructions to make the installation and usage clear for the average user, even if they are not particularly experienced with computers. With this in mind, we will be looking at the results.

In this project, we explored facial recognition techniques using a modular, progressive and script-based approach. We evaluated multiple models, such as InsightFace and OpenCV, under varied conditions including images, videos, and scraped web data.

Each model presents trade-offs:

- **OpenCV:** Offers faster execution times and lower computational cost, but suffers from reduced precision and higher false positive rates. This results in more time needed for human review.
- **InsightFace:** Provides better recognition accuracy and fewer false positives, even under challenging conditions such as occlusions, masks, and extreme lighting angles. However, it is significantly slower. Thankfully, we were able to accelerate it with GPU usage.

Adequate combination We deem that the combination of openCV and InsightFace remains the optimal one due to the sufficiently accurate detection and the importance of precise recognition to avoid false positives.

The final script now took an average of 250 seconds to scan the above-mentionned list of website with a depth of 0 (only the main page) and 2 threads, of which only 12 seconds was required for the scanning itself, and the rest was dedicated to recognition. The same task with one thread running with the GPU only took 150 seconds.

The use of multiprocessing made the code more efficient by spreading the requests to the different urls and running in paralel. The same logic applies to the recognition phase, for which results were better than during testing due to not all online images containing faces. However for this phase we found GPU usage to be better than using multiple processors running on the CPU.

Further testing This script can easily be tested more extensively by using the varied arguments available upon execution, such as changing the resizing, the model provided and the tresholds, as well as offline files used for testing and various urls scanned without having to change the code.

Web scraping Web scraping functionality was successfully implemented, though limited by ethical and legal constraints. It highlights the real-world viability of automated facial recognition for online identity monitoring, but also demonstrates the need for proper API access and policy compliance.

This model has the potential to be more accurate than online web search alternatives given we can make a more accuracte facial model using the multiple pictures we captured.

In short, the project shows the feasibility and versatility of a modular recognition framework to be iterated on, while also emphasizing the need for careful consideration of ethical, technical, and security-related challenges in real-world deployments.

Chapter 7
Conclusions

This thesis has explored the mechanisms of facial recognition systems, their increasing deployment in contemporary society, and the legal and ethical challenges they present. Particular attention was given to the European Union's evolving regulatory landscape, including the forthcoming AI Act, which will be applicable on the 2nd of August 2026.

Facial search tool In this context, the research presented here introduces a practical tool designed to enhance privacy protections for individuals, particularly employees within organizations. By maintaining a protected database of facial identities, the system can continuously scans publicly accessible images online to detect unauthorized appearances. While it cannot retroactively remove data from existing machine learning datasets, the tool enables affected individuals to take proactive steps, such as initiating takedown requests or adjusting future online behavior. This anticipates a legal environment in which individuals may have greater control over their biometric data.

The final version we propose is equipped with logs of usage, an audit for unsure results, as well as a suite of execution parameters. The code is modular, optimised, minimal and sufficiently commented. We have also successfully managed to increase both the accuracy and speed of the tool, by resizing images and making use of the GPU, after doing tests with multiple processes the former proved to be the better option. However, the fact this is a new project that required proper research and that much was done in terms of implementation means that there is now more room for extensive and more well defined testing, as well as investigating warnings and logs to find any potential issues.

Due to the terms of use of social media websites which prevent scraping, and limits in their API, we were not able to implement the full solution desired. Instead we choose to work with a curated list of urls to scan. Further research can request access to the twitter API due to the DSA, and companies can request access to basic public image scraping with meta, both on facebook and instagram, which could then be integrated into the existing code base.

Other adversarial methods Existing approaches to mitigating facial recognition risks - such as image obfuscation, adversarial modifications, or disguises - remain limited in scope and often impractical at scale. The solution proposed here complements these by emphasizing awareness, transparency, and early detection. Ultimately, addressing the broader risks of facial recognition requires a cultural shift toward digital literacy and data autonomy.

The system is distributed with comprehensive usage documentation and follows principles of modular software design to support ongoing development and ethical deployment. Future improvements may include the integration of adversarial image generation, stronger detection algorithms, and adaptation to evolving recognition techniques.

As society increasingly integrates facial recognition technologies into both commercial and governmental infrastructures, tools such as the one proposed in this thesis serve as important first steps in empowering individuals and institutions to safeguard their digital identities.

7.1 Privacy Concerns

The potential misuse of this technology is a significant concern. While the intent behind this tool is to enable personal security and automated identity verification, similar methods could be exploited by malicious actors. Unfortunately, such actors already have access to more developped commercial facial recognition platforms through paid subscriptions. Our solution does not add much additional risk, since there are better alternatives that do scrape social media, such as **facesearch**.

Concerns to be addressed before deployment While the current implementation demonstrates the technical feasibility of the proposed tool, several critical areas must be addressed to ensure privacy, security, and long-term reliability. First, the system lacks robust fault tolerance; edge cases have not been extensively tested, and error-handling mechanisms are limited. Enhancing exception management and incorporating validation at key stages would improve the script's reliability in real-world use. Additionally, all data—including facial images, search results, and temporary files—is stored in plaintext. In a production environment, this presents a considerable risk. A secure, centralized database with end-to-end encryption and fine-grained access control should be adopted to safeguard sensitive information.

Another concern lies in the logging and audit trail. Although the script maintains basic logs and allows for manual auditing, a more comprehensive, tamper-resistant logging infrastructure is essential to ensure transparency, reproducibility, and compliance with potential regulatory frameworks. Furthermore, the system currently does not implement any form of memory protection. As such, there are theoretical risks of data leakage via memory scraping techniques. Although the application is designed for local use and does not transmit images over the internet, mitigating in-memory data persistence through secure memory management practices should be considered, especially if deployed on shared or cloud-based systems.

Finally, usability and maintainability could benefit from the introduction of a formal installation process, and from packaging the system with configuration templates and environment isolation (e.g., via Docker). These additions would support easier deployment, ensure reproducibility, and help prevent unintentional misconfigurations. Addressing these limitations is crucial for transitioning this prototype into a secure, scalable solution appropriate for use by companies, governmental agencies, or privacy-conscious individuals.

At this stage, these concerns are theoretical, as the prototype has not yet been deployed. Nonetheless, any move toward a production-ready version must be accompanied by comprehensive privacy policies, access restrictions, and encryption standards to ensure ethical and secure usage.

7.2 Future Work

Several directions could enhance this project and expand its potential applications. First and foremost, the usability of the tool should be improved. Ideally, the system would support automated workflows where users can submit their facial images directly and then triggering recurring recognition processes without manual intervention.

As facial recognition algorithms continue to evolve, becoming both more efficient and more accurate, it will be important to periodically update the implementation to incorpo-

rate these advancements.

Further testing and benchmarking of alternative facial recognition models could help identify more robust or efficient algorithms for specific contexts. A comparative evaluation would clarify which models offer the best trade-offs in terms of accuracy, speed, and resource requirements.

Another important area for development involves the ethical and legal aspects of data collection. Many online platforms that are of interest to us currently prohibit automated scraping, so it would be good to request permission to use their data under a research-oriented framework. This step would also help establish partnerships and improve the tool's legitimacy and reliability.

Future iterations might also explore integrating Deepfake detection countermeasures.

Lastly, when the tool is ready for deployment, special attention must be paid to access control and protection mechanisms to prevent misuse or unauthorized access to its features.

Appendix A

main.py

```
"""
Author: Alexandre De Groodt
Script for facial recognition using the database provided by face.py using web scraping or local files.
The settings are in settings.py, and this uses the code of scraping.py to scrape the web, as well as ui.py to display the analysis/audit afterwards.
The code has many options, by default it will show a window that lets the user keep track of the process as it is happening.

This main file contains all the logic as well as the code required for facial detection and recognition.
"""

import numpy as np
import cv2, os
import sys, contextlib
from insightface.app import FaceAnalysis
from datetime import datetime
import tqdm
from multiprocessing import Pool
import dlib
import imutils
import tempfile
import base64
import math

from urllib.parse import urljoin, urlparse

from scripts.ui import *
from face import *
from scripts.settings import *
from scripts.scraping import *

# Constants.
DISPLAY_WINDOW = "Scanning Process"
RED, GREEN, BLUE = (0, 0, 255), (0, 255, 0), (255, 0, 0)

# The temporary files are named in a specific way to easily identify them.
LOG_TEMP = "plog.txt"
SCAN_TEMP = "zcan.txt"
AUDIT_TEMP = "audi.txt"

# ---------------- SETUP ---------------

```

```
# Setup foldering.
def MakeFolders(path, args, make_log = False):
    global log, audit
    args.results_folder = path
    i = 0
    if(args.incrementFolder):
        while os.path.exists(path + str(i) + "/"):
            i += 1
        results_folder = path + str(i) + "/"

    os.makedirs(args.results_folder + args.detectedFaces,
       exist_ok=True)
    os.makedirs(args.results_folder + args.foundDir,
       exist_ok=True)
    if(args.storeFailed):
        os.makedirs(args.results_folder + args.missDir,
           exist_ok=True)
    args.scanned_urls = []

    if(make_log or not os.path.exists(args.results_folder +
       args.logFile)):
        log = open(args.results_folder + args.logFile, "a+")
        audit = open(args.results_folder + args.auditFile, "a+")
        # Keep trace of scanned urls in the main folder, so that
           it stays relevant beyond this execution.
        # First we read the list of scanned urls, and then the
           code will append to it as it scans imgs.
        if(os.path.exists(args.scannedUrls) and not args.reScan):
            args.scanned_urls = [line.strip() for line in
               open(args.scannedUrls, "r").readlines()]

# Setup the openCV displaying window.
def MakeWindow(args):
    if not args.noDisplay:
        print("Scanning will start soon. Press q to hide the
           display.")
        cv2.namedWindow(DISPLAY_WINDOW, cv2.WINDOW_NORMAL)
        cv2.resizeWindow(DISPLAY_WINDOW, args.displaySize,
           args.displaySize)

# Save a log and print the result if requested.
def Logs(log, text, terminal_results):
    log.write(text + "\n")
    if(terminal_results):
        print(text)


# ---------------- SAVING ---------------

# Save the face crop and then annotate it.
def SaveCrop(face_crop, name, img, best_similarity, img_file, x,
   y, w, h, log, audit, args):
```

```
83      # Remove the extension.
84      img_name = os.path.splitext(img_file)[0]
85      # Draw red rectangle and name in red if unknown and green if
            known.
86      if(name is None or best_similarity < args.auditThreshold):
87          color = RED
88      elif(best_similarity < args.notifiedThreshold):
89          color = GREEN
90      else:
91          # Face detected with enough surety, write it to the
                person's folder.
92          args.personList[name] += 1
93          color = BLUE
94          face_filename =
                os.path.join(f"{args.faceDatabase}/{name}/",
                f"{img_name}_face.jpg")
95          cv2.imwrite(face_filename, face_crop)
96      if(args.saveFaces):
97          face_filename = os.path.join(args.results_folder +
                args.detectedFaces, f"{name}_{img_name}.jpg")
98          cv2.imwrite(face_filename, face_crop)
99
100     if(best_similarity < args.notifiedThreshold and
            best_similarity > args.auditThreshold):
101         audit.write(f"{img_file}:{name}:{best_similarity:.2f}:"+
102             "{x}:{y}:{w}:{h}:0\n")
103     DrawRectangle(img, f"{name} {best_similarity:.2f}", color,
            x, y, w, h)
104
105     text = f"Face detected in {img_file}. Best Match: {name},
            Similarity: {best_similarity:.2f}"
106     Logs(log, text, args.terminalResults)
107
108 # Save the img with annotated information.
109 def SaveImage(matches, img, img_path, img_file, result_found,
        log, args):
110     if(result_found):
111         save_path = os.path.join(args.results_folder +
                args.foundDir, img_file)
112     elif(args.storeFailed):
113         save_path = os.path.join(args.results_folder +
                args.missDir, img_file)
114
115     # If we keep this picture for the audit it means a result
            was found.
116     if(not args.noDisplay and (args.showAll or result_found)):
117         key = cv2.waitKey(1)
118         # Needed for it to have time to display and offer the
                possibility to stop the window.
119         if(key == ord('q') or not
                cv2.getWindowProperty(DISPLAY_WINDOW,
                cv2.WND_PROP_VISIBLE)):
```

```
            args.noDisplay = True
            cv2.destroyAllWindows()
        elif(key == ord('t')):
            pass
        else:
            cv2.imshow(DISPLAY_WINDOW, img)

    if(result_found):
        cv2.imwrite(save_path, img)
    # Read it from scratch and add only the anotated information
       for the concerned person.
    for name in matches:
        if(args.offlineTests):
            personal_img = cv2.imread(img_path)
        if personal_img is None:
            return
        x, y, w, h, similarity = matches[name][0],
           matches[name][1], matches[name][2], matches[name][3],
           matches[name][4]
        DrawRectangle(img, f"{name} {similarity:.2f}", (0, 255,
           0), x, y, w, h)
        save_path = os.path.join(f"{args.faceDatabase}/{name}/",
           img_file)
        cv2.imwrite(save_path, personal_img)
        args.personal_logs[name] = args.personal_logs[name] +
           f"\t Face detected in {img_file} with a
           {similarity:.2f} accuracy.\n"



# ---------------- PREP ---------------

# Used for reading videos and extracting face information.
def VideoRecPrep(vid_name, vid_path, log, args, detector,
   recognizer):
    video = cv2.VideoCapture(vid_path)
    if not video.isOpened():
        print("Error: Could not open video.")
        return
    vid_name = os.path.splitext(vid_name)[0]
    # Save the video results in a sub folder.
    #MakeFolders(base_folder + "/" + vid_name + "/")
    ret, frame = video.read()
    if not ret:
        print("Error: Could not read initial frame from video.")
        video.release()
        return
    # img_number * IGNORE_FRAMES = current frame in the video.
    img_number = 0
    while video.isOpened() and ret:
        frame = Resize(frame, args.displaySize)
        frame = cv2.rotate(frame, cv2.ROTATE_90_COUNTERCLOCKWISE)
```

```
164             # No need to read all frames, process one and then skip
                    ahead.
165             FaceRecognition(vid_name + str(img_number) + ".jpg",
                    vid_path, frame, log, audit, args, detector,
                    recognizer)
166             video.set(cv2.CAP_PROP_POS_FRAMES, img_number *
                    args.ignoreFrames)
167             ret, frame = video.read()
168             img_number += 1
169         video.release()
170
171     # Calls the main function after some transformations, used when
            the image has not been read already.
172     def ImgRecPrep(img_name, img_path, log, audit, args, detector,
            recognizer):
173         img = cv2.imread(img_path)
174         if img is None:
175             Logs(log, f"Error loading {img_path}, skipping.",
                    args.terminalResults)
176             return 0
177
178         return FaceRecognition(img_name, img_path, Resize(img,
                args.displaySize), log, audit, args, detector, recognizer)
179
180     def Detect(img, detector):
181         # Ensure it's a 3-channel BGR img
182         if img.ndim == 2 or img.shape[2] == 1:
183             raise ValueError("Input img must be a 3-channel BGR img")
184         faces = []
185
186         h, w = img.shape[:2]
187         detector.setInputSize((w, h))
188         _, faces = detector.detect(img)
189         return faces
190
191
192     # ---------------- RECOGNITION ----------------
193
194     # Main face detection and recognition function.
195     def FaceRecognition(img_file, img_path, img, log, audit, args,
            detector, recognizer):
196         faces = Detect(img, detector)
197         matches = {}
198         face_count = 0
199         keep_audit = False
200         if(faces is None):
201             Logs(log, f"No faces detected in {img_file}.",
                    args.terminalResults)
202         else:
203             for i, face in enumerate(faces):
204                 face_count += 1
205                 x, y, w, h = face[:4].astype(int)
```

```
206             face_crop = img[y:y+h, x:x+h]
207             if(face_crop.size == 0 or w < args.minFaceSize or h
                    < args.minFaceSize):
208                 continue
209
210             if(args.openCVRec):
211                 alignedFace = recognizer.alignCrop(img, face)
212                 embedding = recognizer.feature(alignedFace)
213             else:
214                 face_data = recognizer.get(face_crop)
215                 if len(face_data) == 0:
216                     continue
217                 embedding = face_data[0].embedding
218             best_match = None
219             best_similarity = -1
220             similarity = 0
221             for name, ref_embedding in args.stored_faces.items():
222                 if(args.openCVRec):
223                     # Cosine similarity normalized from 0 to 100.
224                     similarity = (args.recognizer.match(
225                     embedding.astype(np.float32),
                            ref_embedding.astype(np.float32),
                            cv2.FaceRecognizerSF_FR_COSINE) + 1) * 50
226                     #l2_score = recognizer.match(embedding,
                            ref_embedding,
                            cv2.FaceRecognizerSF_FR_NORM_L2)
227                     #l2_percent = max(0, 100 * (1 - l2_score /
                            2.0)) > 60
228                 else:
229                     similarity = ((np.dot(embedding,
                            ref_embedding) /
                            (np.linalg.norm(embedding) *
                            np.linalg.norm(ref_embedding))) + 1) * 50
230
231                 if similarity > best_similarity:
232                     best_similarity = similarity
233                     best_match = name
234                     # If there are multiple people in the
                            picture they each need to be notified and
                            sent a copy.
235                     if similarity > args.auditThreshold:
236                         keep_audit = True
237                         matches[best_match] = (x, y, w, h,
                                similarity)
238             if similarity < args.auditThreshold:
239                 args.unknown_count += 1
240             SaveCrop(face_crop, best_match, img,
                    best_similarity, img_file, x, y, w, h, log,
                    audit, args)
241     audit.flush()
242     args.picture_scanned += 1
243     SaveImage(matches, img, img_path, img_file, keep_audit, log,
```

```
        args)
    return face_count


# ---------------- SCAN --------------

# Scan all images and videos provided with the given url with
    the detectors and recognizers for the main function.
def ScanUrl(to_scan, detector, recognizer):
    _to_scan, args, folder = to_scan
    img_to_scan, vid_to_scan = _to_scan
    log = open(folder + LOG_TEMP, "a+")
    scanned = open(folder + SCAN_TEMP, "a+")
    audit = open(folder + AUDIT_TEMP, "a+")
    img_face_count = [0, 0]

    i = 0
    for img_info in img_to_scan:
        url, page = img_info[0], img_info[1]
        try:
            # Handle base64-encoded images (data URIs) separately
            if url.startswith("data:image"):
                header, encoded = url.split(",", 1)
                image_data = base64.b64decode(encoded)
            else:
                # Only use requests for regular URLs
                r = requests.get(url, stream=True)
                if r.status_code != 200:
                    Logs(log, f"Failed to download picture:
                        {url} (status code {r.status_code})",
                        args.terminalResults)
                    continue
                image_data = r.content

            img_array = np.asarray(bytearray(image_data),
                dtype=np.uint8)
            img = cv2.imdecode(img_array, cv2.IMREAD_COLOR)
            if img is not None:
                scanned.write(url + "\n")
                filename = f"{page}_{i}.jpg"
                img_face_count[1] += FaceRecognition(filename,
                    args.results_folder, Resize(img,
                    args.displaySize), log, audit, args,
                    detector, recognizer)
                img_face_count[0] += 1
                i += 1
        except Exception as exception:
            Logs(log, f"Exception during image handling:
                {exception}", args.terminalResults)

    i = 0
    for vid_info in vid_to_scan:
```

```
            url, page = vid_info[0], vid_info[1]
            r = requests.get(url, stream=True)
            if r.status_code == 200:
                # Save a temp of the video.
                with tempfile.NamedTemporaryFile(delete=False,
                    suffix=".mp4") as tmp:
                    tmp.write(r.content)
                    tmp_path = tmp.name
                VideoRecPrep(f"{page[1]}_{i}.mp4", tmp_path, log,
                    args, detector, recognizer)
                os.remove(tmp_path)  # Clean up temporary file
            else:
                Logs(log, f"Failed to download video: {url} from
                    {page} (status code {r.status_code})",
                    args.terminalResults)
    return img_face_count

def MakeModels(args):
    detector = cv2.FaceDetectorYN.create(args.yunetDetector, "",
        args.detSize, score_threshold=args.scoreTreshold,
        nms_threshold=args.nmsTreshold, top_k=args.topK)

    if(args.openCVRec):
        recognizer =
            cv2.FaceRecognizerSF.create(args.opencvRecognizer, "")
    else:
        recognizer = FaceAnalysis(name=args.insightFaceModel,
            providers=[args.insightFaceProvider])
        recognizer.prepare(ctx_id=0, det_size=args.detSize)
        # To go after the stuff it prints.
        print("\n")

    return detector, recognizer

# Wrapper function to call ScanUrl on each item in the sublist
def ScanUrlBatch(scan_info):
    batch, args = scan_info
    detector, recognizer = MakeModels(args)
    return_var = [0, 0]
    for item in tqdm.tqdm(batch):
        return_var += ScanUrl(item, detector, recognizer)
    return return_var

# Split a list into N nearly equal parts
def Chunkify(lst, n):
    avg = math.ceil(len(lst) / n)
    return [lst[i:i + avg] for i in range(0, len(lst), avg)]

# Take all the logs that fit the pattern and unify them into one.
def UnifyLogs(source_folder, pattern, result, audit_log=False):
    # The size check is not very rigurous, but it works.
    filenames = [source_folder + text_file for text_file in
```

```
            os.listdir(source_folder) if len(text_file) > 8 and
            text_file[-8:] == pattern]
331         full_log = open(result, "a+")
332         for fname in filenames:
333             with open(fname) as infile:
334                 full_log.write(infile.read())
335             os.remove(fname)
336         if(not audit_log):
337             full_log.write(f"Scanning finished at: {datetime.now()}")
338             full_log.write("End of log ----------------------------")
339         full_log.close()
340
341     # Joins the varying files into one and returns the log file for
        further use.
342     def Cleanup(args):
343         UnifyLogs(args.results_folder, LOG_TEMP, args.results_folder
            + args.logFile)
344         log = open(args.results_folder + args.logFile, "a+")
345         UnifyLogs(args.results_folder, SCAN_TEMP, args.scannedUrls)
346         UnifyLogs(args.results_folder, AUDIT_TEMP,
            args.results_folder + args.auditFile, True)
347         return log
348
349     if __name__ == '__main__':
350         # See in the settings.py file for more.
351         args = ParseArguments()
352
353         # Name the folder with the current date.
354         base_folder = f"results_{datetime.date(datetime.now())}/"
355         # Useful in case the last execution was stopped mid-process.
356         if(os.path.exists(base_folder)):
357             args.results_folder = base_folder
358             log = Cleanup(args)
359             # Make folder will open it again.
360             log.close()
361         MakeFolders(base_folder, args, True)
362
363         # Variables used for storage.
364         args.stored_faces = {}
365         args.personList = {}
366         # Captain Jean Luc Picard.
367         args.personal_logs = {}
368         # The amount of unknown faces detected, only relevant for
            offline picture testing.
369         args.unknown_count = 0
370         args.picture_scanned = 0
371         # Used to count pictures scanned and faces detected.
372         img_face_count = [0, 0]
373
374         # Verify the folders do exist. If one was created, we can
            assume so was the rest.
375         if(os.path.exists(args.results_folder) is None):
```

```
        print("Folders not created. Exiting script.")
    else:
        log.write("Start time: " + str(datetime.now()) + "\n")
        # Store all the important current settings in the log
            file.
        log.write(PrintArguments(args))

        startminute =
            (datetime.now()-datetime(1970,1,1)).total_seconds()

        detector, recognizer = MakeModels(args)

        LoadFaces(log, args, detector, recognizer)
        # If we have no pictures, try capturing some with the
            webcam using the other script.
        if(not len(args.stored_faces)):
            WebcamCaptures()
            CallConvert()
            LoadFaces(log, args, detector, recognizer)

        if(args.offlineTests):
            # Make a list containing all input imgs and videos.
            scan_files = [f for f in
                os.listdir(args.offlineFiles) if
                f.lower().endswith((".jpg", ".png", ".jpeg",
                ".mp4"))]

        # Keep a copy for video saving.
        temp_results_folder = args.results_folder
        if(args.offlineTests):
            MakeWindow(args)
            if(len(scan_files)):
                for scan_file in tqdm.tqdm(scan_files):
                    if scan_file.lower().endswith(".mp4") and
                        not args.skipVideos:
                        VideoRecPrep(scan_file,
                            os.path.join(args.offlineFiles,
                            scan_file), log, args, detector,
                            recognizer)
                        # Currently disabled: store vids in a
                            separate folder and reset it back to
                            the main one here
                        #MakeFolders(base_folder)
                    elif not args.skipPictures:
                        ImgRecPrep(scan_file,
                            os.path.join(args.offlineFiles,
                            scan_file), log, audit, args,
                            detector, recognizer)
        # Online web scraping.
        else:
            if(args.reScan):
                args.scanned_urls = []
```

```
413             to_scan = []
414             if(not args.skipPictures or not args.skipVideos):
415                 # The list of arguments provided to the script.
416                 info = [
417                     [url, args.results_folder +
                            urlparse(url).netloc.replace(".", "_") +
                            "_" + "wlog.txt", args.scanDepth,
                            args.scanned_urls, args.skipPictures,
                            args.skipVideos, args.terminalResults,
                            not args.noThreading]
418                     for url in args.urlList
419                 ]
420                 if(not args.noThreading):
421                     with Pool(cv2.getNumberOfCPUs() - 1) as pool:
422                         i = 0
423                         for result in pool.map(RecursiveScrape,
                                info):
424                             if(result):
425                                 # The results to analyze, the
                                    arguments and the third
                                    element is for file names.
426                                 text_files_path = info[i][1][:-8]
427                                 to_scan.append((result, args,
                                    text_files_path))
428                             i += 1
429                 else:
430                     i = 0
431                     for line in info:
432                         result = RecursiveScrape(line)
433                         if(result):
434                             # The results to be used by the
                                ScanUrl function.
435                             text_files_path = info[i][1][:-8]
436                             to_scan.append((result, args,
                                text_files_path))
437                         i += 1
438
439                 a = f"Scanning time:
                        {((datetime.now()-datetime(1970,1,1))
440                      .total_seconds() - startminute):.2f}\n"
441                 print(a)
442                 log.write(a)
443                 UnifyLogs(args.results_folder, "wlog.txt",
                        args.results_folder + "scan.txt")
444
445             # Recognition phase.
446             if(len(to_scan)):
447                 MakeWindow(args)
448                 if(args.noThreading or args.threadNbr == 1 or
                        len(to_scan) == 1):
449                     detector, recognizer = MakeModels(args)
450                     for site in tqdm.tqdm(to_scan):
```

```
                        ScanUrl(site, detector, recognizer)
                else:
                    # Spread the list into sub_batches so we
                       don't have more detectors and recognizer
                       running at the same time than we have to.
                    chunks = [(chunk, args) for chunk in
                       Chunkify(to_scan, args.threadNbr)]
                    with Pool(args.threadNbr) as pool:
                        for result in pool.map(ScanUrlBatch,
                            chunks):
                            img_face_count[0] += result[0]
                            img_face_count[1] += result[1]
                # Logs, assemble!
                log.close()
                audit.close()
                log = Cleanup(args)

        # Report results, if any.
        if(len(args.personList) or args.unknown_count or
            img_face_count[0] or img_face_count[1]):
            text = ""
            for name in args.personList:
                if(args.personList[name]):
                    text += f" {name}: {args.personList[name]}
                       times"
                    person_log =
                       open(f"{args.faceDatabase}{name}{args.logFile}",
                       "a+")
                    person_log.write(f"Face detected
                       {args.personList[name]} times at:
                       {datetime.now()}\n")
                    person_log.write(args.personal_logs[name] +
                       "\n")
                    person_log.close()
            if(len(text)):
                to_write = f"{args.personList} matches found in
                   the pictures, {text} detected, and
                   {args.unknown_count} unknowns in
                   {args.picture_scanned} pictures."
                Logs(log, to_write, args.terminalResults)
            overview = f"{img_face_count[0]} pictures scanned
                with {img_face_count[1]} facial matches found."
            print(overview)
            Logs(log, overview, False)
            Logs(log, f"End time: {datetime.now()}\n",
                args.terminalResults)
            a = f"Execution time:
                {((datetime.now()-datetime(1970,1,1)).total_seconds()
                - startminute):.2f}\n"
            # Always write the execution time and summary
                results to the screen.
            #print(f"Numbers of, unknown faces:
```

```
                {args.unknown_count}, scanned pictures:
                {args.picture_scanned}\n")
484             print(a)
485             log.write(a)
486
487         log.close()
488
489         audit = open(args.results_folder + args.auditFile, "r+")
490         if(not args.noAudit and audit.read(1)):
491             cv2.destroyAllWindows()
492             print(f"The image navigator will open to display the
                    images with similarity between
                    {args.auditThreshold} and
                    {args.notifiedThreshold}.\n"
493                 "Use the left and right arrows to move, and
                        click on a person's face to confirm their
                        identity by pressing the 'confirm' button
                        below.\n"
494                 "Note that you cannot do this for faces that
                        have too low or too high similarities.\n"
495                 "Feel free to do this audit in multiple parts,
                        each picture will only be shown once in
                        that case.\n")
496             operator = "alex"
497
498             # Create the tkinter window
499             window = tk.Tk()
500             app = ImageNavigator(window, args.results_folder,
                    args.faceDatabase, operator, args.displaySize)
501             window.mainloop()
502         elif(not args.noAudit):
503             print("No pictures to audit. You can change the
                    tresholds for auditing in the code using
                    --auditThreshold=50 - or with another number.")
```

Appendix B

scripts/settings.py

```
"""
Author: Alexandre De Groodt
Sets the settings for main.py, displays them and provides an
   explanation to users upon running main.py --help.
"""

import argparse
import json

def PrintArguments(args):
    # ====== PRINT SUMMARY ======
    to_print = "\n[INFO] Running with the following
       configuration:\n"

    to_print += "- Numeric thresholds:\n" + json.dumps({
        "auditThreshold": args.auditThreshold,
        "notifiedThreshold": args.notifiedThreshold,
        "minFaceSize": args.minFaceSize,
        "ignoreFrames": args.ignoreFrames,
        "scanDepth": args.scanDepth,
        "threadNbr": args.threadNbr
    }, indent=4) + "\n"

    to_print += "- Booleans:\n" + json.dumps({
        "noDisplay": args.noDisplay,
        "storeFailed": args.storeFailed,
        "showAll": args.showAll,
        "skipPictures": args.skipPictures,
        "skipVideos": args.skipVideos,
        "incrementFolder": args.incrementFolder,
        "noAudit": args.noAudit,
        "offlineTests": args.offlineTests,
        "terminalResults": args.terminalResults,
        "reScan": args.reScan,
        "saveFaces": args.saveFaces
    }, indent=4) + "\n"

    to_print += "- Directories:\n" + json.dumps({
        "faceDatabase": args.faceDatabase,
        "offlineFiles": args.offlineFiles,
        "foundDir": args.foundDir,
        "missDir": args.missDir,
        "detectedFaces": args.detectedFaces
    }, indent=4) + "\n"

    to_print += "- Files:\n" + json.dumps({
```

```
        "logFile": args.logFile,
        "auditFile": args.auditFile,
        "scannedUrls": args.scannedUrls
    }, indent=4) + "\n"

    to_print += "- Detector settings:\n" + json.dumps({
        "yunetDetector": args.yunetDetector,
        "detSize": args.detSize,
        "scoreTreshold": args.scoreTreshold,
        "nmsTreshold": args.nmsTreshold,
        "topK": args.topK
    }, indent=4) + "\n"

    recognizer = "InsightFace"
    if args.openCVRec:
        recognizer = "OpenCV"

    to_print += f"- Recognizer: {recognizer}\n"

    if(recognizer == "InsightFace"):
        to_print += "- Recognizer settings:\n" + json.dumps({
            "insightFaceModel": args.insightFaceModel,
            "insightFaceProvider": args.insightFaceProvider
        }, indent=4) + "\n"

    to_print += "- URL list:\n" + json.dumps(args.urlList,
        indent=4) + "\n"

    to_print += "Type main.py --help for more info.\n"

    return to_print


def ParseArguments():
    parser = argparse.ArgumentParser(description="Facial
        recognition and audit CLI")

    # Integer thresholds.
    parser.add_argument('-at', '--auditThreshold', type=int,
        default=60, help="Minimum audit match score (default:
        60)")
    parser.add_argument('-nt', '--notifiedThreshold', type=int,
        default=70, help="Notification threshold (default: 70)")
    parser.add_argument('-mfs', '--minFaceSize', type=int,
        default=32, help="Minimum face size (default: 32)")
    parser.add_argument('-igf', '--ignoreFrames', type=int,
        default=30, help="Frames to skip in video (default: 30)")
    parser.add_argument('-w', '--displaySize', type=int,
        default=800, help="Size of the squared display.")
    parser.add_argument('-sd', '--scanDepth', type=int,
        default=0, help="Depth of the website scans (0 for base
        pages only) (default: 1)")
```

```
87      parser.add_argument('-tn', '--threadNbr', type=int,
            default=2, help="WARNING: number of threads used to run
            the facial recognition algorithm. (default: 2)")
88
89      # Booleans.
90      parser.add_argument('-stf', '--storeFailed',
            action='store_true', default=False, help="Store images
            that do not pass the treshold (default: False)")
91      parser.add_argument('-sa', '--showAll', action='store_true',
            default=False, help="Show pictures without results
            (default: False)")
92      parser.add_argument('-sp', '--skipPictures',
            action='store_true', default=False, help="Skip pictures
            (default: False)")
93      parser.add_argument('-sv', '--skipVideos',
            action='store_true', default=False, help="Skip video
            (default: False)")
94      parser.add_argument('-if', '--incrementFolder',
            action='store_true', default=False, help="Increment
            output folder (default: False)")
95      parser.add_argument('-na', '--noAudit', action='store_true',
            default=False, help="Perform audit (default: False)")
96      parser.add_argument('-ot', '--offlineTests',
            action='store_true', default=False, help="Use offline
            files (default: False)")
97      parser.add_argument('-sr', '--terminalResults',
            action='store_true', default=False, help="Show detailed
            terminal results (default: False)")
98      parser.add_argument('-so', '--reScan', action='store_true',
            default=False, help="Scan images again (default: False)")
99      parser.add_argument('-sf', '--saveFaces',
            action='store_true', default=False, help="Save detected
            faces (default: False)")
100
101     parser.add_argument('--openCVRec', action='store_true',
            default=False)
102     # Displaying and threading (mutually exclusive)
103     group2 = parser.add_mutually_exclusive_group()
104     group2.add_argument('-td', '--noThreading',
            action='store_true', default=False, help="Avoid
            multi-threading for web scraping, necessary to show the
            window (default: False)")
105     group2.add_argument('-nd', '--noDisplay',
            action='store_true', default=True, help="Hides the video
            window (default: True)")
106
107     # Paths and filenames (strings).
108     parser.add_argument('--faceDatabase', type=str,
            default="protectees/")
109     parser.add_argument('--offlineFiles', type=str,
            default="offline_files/")
110     parser.add_argument('--foundDir', type=str, default="found/")
```

```
111     parser.add_argument('--missDir', type=str, default="miss/")
112     parser.add_argument('--detectedFaces', type=str,
            default="detected_faces/")
113     parser.add_argument('--logFile', type=str, default="log.txt")
114     parser.add_argument('--auditFile', type=str,
            default="audit.txt")
115     parser.add_argument('--scannedUrls', type=str,
            default="scanned_urls.txt")
116
117     # Detection model settings.
118     parser.add_argument('--yunetDetector', type=str,
            default="models/yunet.onnx")
119     parser.add_argument('--detSize', type=tuple, default=(320,
            320))
120     parser.add_argument('--scoreTreshold', type=float,
            default=0.7)
121     parser.add_argument('--nmsTreshold', type=float, default=0.3)
122     parser.add_argument('--topK', type=int, default=5000)
123     parser.add_argument('--opencvRecognizer', type=str,
            default="models/face_recognition_sface_2021dec.onnx")
124     parser.add_argument('--insightFaceModel', type=str,
            default="buffalo_l")
125     parser.add_argument('--insightFaceProvider', type=str,
            default="CPUExecutionProvider")
126
127     # URL list
128     parser.add_argument('--urlList', type=lambda s: [url.strip()
            for url in s.split(',')], default=[
129         "https://www.bcg.com",
130         #"https://www.instagram.com",
                "https://www.facebook.com", "https://twitter.com",
                "https://www.tiktok.com", "https://www.youtube.com",
                "https://www.reddit.com", "https://www.linkedin.com"
131         "https://www.cnn.com",
132         "https://www.bbc.com",
133         "https://www.nytimes.com",
134         "https://www.tumblr.com",
135         "https://500px.com",
136         #"https://www.ebay.com", takes forever
137         "https://www.wikipedia.org",
138         "https://commons.wikimedia.org",
139         "https://www.ieee.org",
140         # Belgian urls:
141         "https://www.hln.be",
142         "https://www.rtbf.be",
143         "https://www.vrt.be",
144         "https://www.bruzz.be",
145         "https://www.vtm.be",
146         "https://www.7sur7.be",
147         "https://www.student.be",
148         "https://www.ugent.be",
149         "https://www.uclouvain.be",
```

```
150         "https://www.ulb.be",
151         "https://www.vub.be",
152         # Cultural centers:
153         "https://www.muntpunt.be",              # Cultural hub &
                library in Brussels
154         "https://www.kvs.be",                   # Royal Flemish
                Theatre
155         "https://www.kaaitheater.be",           # Contemporary
                theatre
156         "https://www.debijloke.be",             # Muziekcentrum
                De Bijloke Gent
157         "https://www.culture.be",               # Portal for
                cultural activities (government)
158         "https://www.flandersartsinstitute.be",# Research &
                support for Flemish arts
159         "https://www.passaporta.be",            # International
                house of literature
160         "https://www.visitflanders.com",        # Official
                tourism platform (lots of public event imagery)
161         "https://www.ccdeadelberg.be",          # Local cultural
                center in Halle
162         "https://www.decentrale.be",            # Cultural
                diversity center in Ghent
163         "https://www.arenberg.be",              # Culture house
                in Antwerp
164         "https://www.cultuurcentrummechelen.be",# Culture center
                Mechelen
165         "https://www.fomu.be",                  # Museum of
                photography, Antwerp
166         "https://www.kmska.be",                 # Royal Museum of
                Fine Arts Antwerp
167         "https://www.mac-s.be",                 # Museum of
                Contemporary Art, Grand-Hornu
168         "https://www.wiels.org",                # Contemporary
                art center in Brussels
169     ], help="Comma-separated URLs to scrape")
170
171     args = parser.parse_args()
172     # Have the display on for offline testing since it doesn't
            use threads.
173     if(args.offlineTests):
174         args.noDisplay = False
175         args.noThreading = True
176
177     PrintArguments(args)
178
179     return args
```

Appendix C

face.py

```
"""
Author: Alexandre De Groodt
Face detection, capture and extraction into an InsightFace compatible format from the webcam.
This consists in two parts: first the taking of the pictures and then conversion into npy format.
"""

import cv2
import numpy as np
# To handle directories and files.
import os
# To convert the pictures of the face to the right format.
from insightface.app import FaceAnalysis
import tqdm

capture = True
convert = True
resize = True
# Size for the stored face images.
SIZE = 150
# Interval between captures of the face.
CAPTURE_INTERVAL = 10
PICTURES_TO_CAPTURE = 50

# Resize whilst preserving aspect ratio.
def Resize(image, target_size):
    # Get the original dimensions
    h, w = image.shape[:2]

    # Calculate the ratio of the new size to the old size
    ratio = target_size / float(max(h, w))

    # Calculate the new dimensions
    new_dimensions = (int(w * ratio), int(h * ratio))

    # Resize the image
    resized_img = cv2.resize(image, new_dimensions, interpolation=cv2.INTER_AREA)

    # Create a new blank white (or black) image
    blank_img = 255 * np.ones(shape=[target_size, target_size, 3], dtype=np.uint8)

    # Calculate padding
    y_offset = (target_size - new_dimensions[1]) // 2
```

```
    x_offset = (target_size - new_dimensions[0]) // 2

    # Place the resized image onto the blank image
    blank_img[y_offset:y_offset + new_dimensions[1],
       x_offset:x_offset + new_dimensions[0]] = resized_img

    return blank_img


def CaptureFace(faces_data):
    i = 0
    while len(faces_data) < PICTURES_TO_CAPTURE:
        # Capture the video and ensure it works.
        ret, frame = video.read()
        if not ret:
            print("Capture issues.")
            break

        # Used to detect the faces.
        _, faces = face_detector.detect(frame)

        # Crop the faces.
        if faces is not None:
            for face in faces:
                x, y, w, h = face[:4].astype(int)

                # Process the image (crop, resize, rgb form)
                face = frame[y:y+h, x:x+w]
                if(resize):
                    face = Resize(face, SIZE)

                if i % CAPTURE_INTERVAL == 0:
                    faces_data.append(face)
                    filename = os.path.join(img_dir,
                       f"face_{len(faces_data)}.jpg")
                    cv2.imwrite(filename, face)  # Save image as
                       a file

                cv2.rectangle(frame, (x, y), (x + w, y + h), (0,
                   0, 255), 2)
                i += 1
                # Draw the amount of faces captured and a
                   rectangle around the face.
                cv2.putText(frame, str(len(faces_data)), (50,
                   50), cv2.FONT_HERSHEY_COMPLEX, 1, (50, 50,
                   255), 1)
                cv2.rectangle(frame, (x, y), (x+w, y+h), (50,
                   50, 255), 1)

        # Display the video.
        cv2.imshow("Frame", frame)
        # Leave by pressing q.
```

```
87          if cv2.waitKey(1) == ord('q'):
88              break
89      video.release()
90      cv2.destroyAllWindows()
91
92  def WebcamCaptures():
93      global video, face_detector, img_dir, npy_dir
94      # Initialize video capture.
95      video = cv2.VideoCapture(0)
96      if(video.isOpened()):
97          # Use a pre-made classifier to detect the faces,
                assuming normal frontal faces for this.
98          face_detector =
                cv2.FaceDetectorYN.create("models/yunet.onnx", "",
                (640, 480), score_threshold=0.8, nms_threshold=0.3,
                top_k=5000)
99          faces_data = []
100         save_dir = "protectees/"
101         print("Enter your name: ")
102         name = str(input())
103         # Create directories if they don't exist.
104         input_folder = save_dir + name + "/"
105         img_dir = input_folder + "img/"
106         npy_dir = input_folder + "npy/"
107         os.makedirs(img_dir, exist_ok=True)
108         os.makedirs(npy_dir, exist_ok=True)
109         open(input_folder + "log.txt", "w")
110
111         CaptureFace(faces_data)
112     else:
113         print("Issue with opening the video.")
114
115
116 def Convert():
117     for image_file in tqdm.tqdm(image_files):
118         if not image_file.lower().endswith((".jpg", ".png",
                ".jpeg")):
119             continue  # Skip non-image files
120
121         img_path = os.path.join(img_dir, image_file)
122         image = cv2.imread(img_path)
123
124         if image is None:
125             print(f"Error loading {image_file}, skipping.")
126             continue
127
128         # Detect faces
129         face_data = app.get(image)
130         if len(face_data) > 0:
131
132             # Save the embedding as an .npy file
133             person_name = os.path.splitext(image_file)[0]  # Use
```

```
                filename as the identifier
            npy_path = os.path.join(npy_dir,
                f"{person_name}.npy")
            # Save the first detected face.
            np.save(npy_path, face_data[0].embedding)
        else:
            not_found.append(image_file)

    if(len(not_found)):
        print(f"No faces detected in {not_found}, skipped.")

def CallConvert():
    global app, not_found, image_files
    app = FaceAnalysis(name="buffalo_l",
        providers=["CPUExecutionProvider"])
    app.prepare(ctx_id=0, det_size=(320, 320))  # Keep this at
        320 for consistency
    not_found = []
    image_files = [f for f in os.listdir(img_dir) if
        f.lower().endswith((".jpg", ".png", ".jpeg"))]
    Convert()

# Load reference embeddings of the known faces. Used by the main
    code.
def LoadFaces(log, args, detector, recognizer):
    for person_name in os.listdir(args.faceDatabase):
        if(person_name[-4:] == ".txt"):
            continue
        args.personList[person_name] = 0
        args.personal_logs[person_name] = ""
        embeddings_list = []
        if(args.openCVRec):
            person_path = os.path.join(args.faceDatabase,
                person_name + "/img")
            for picture in os.listdir(person_path):
                img_path = os.path.join(person_path, picture)
                img = cv2.imread(img_path)
                if img is None:
                    Logs(log, f"Error loading img: {img_path}",
                        args.terminalResults)
                    continue

                # Get embedding from recognizer
                embedding = recognizer.feature(img)
                embeddings_list.append(np.array(embedding)
                    .astype(np.float32))

            # Store the average embedding for this person
            if(embeddings_list):
                args.stored_faces[person_name] =
                    np.mean(embeddings_list, axis=0)
            else:
```

```
177                 Logs(log, f"No valid embeddings for
                        {person_name}", args.terminalResults)
178         else:
179             person_path = os.path.join(args.faceDatabase,
                    person_name + "/npy/")
180             if(os.path.isdir(person_path)):
181                 embeddings = [
182                     np.load(os.path.join(person_path, file))
183                     for file in os.listdir(person_path)
184                     if file.endswith(".npy")
185                 ]
186                 if(embeddings):
187                     args.stored_faces[person_name] =
                            np.mean(embeddings, axis=0)
188
189 if __name__ == '__main__':
190     if(capture):
191         WebcamCaptures()
192     if(convert):
193         CallConvert()
```

Appendix D

README.md

```
NOT INTENDED FOR COMMERCIAL USE
The pictures kept should be manually deleted after proper
   verification, and are only kept for research and debugging
   purposes.

### Recommendations

We recommend reading the master thesis and the web scraping for
   researchers guide http://arxiv.org/abs/2410.23432 to get an
   understanding of web scraping and facial recognition, as well
   as the purpose of the script and things to look out for. The
   guide will include a shortened version of this as a copy of
   the web \hyperref[scraping]**scraping** section of the state
   of the art.

### Build Instructions:

This project requires python, tkinter (installed by default on
   window), cmake and pip.

**Linux**:
Setup the python environment first:

- `sudo apt install python3-venv python-is-python3`
- `python -m venv venv`
- `source venv/bin/activate`

The last command needs to be used whenever we wish to run the
   script.

Use the **requirements.txt** file to install the requirements as
   follows: `pip install -r requirements.txt`
You may have to install tkinter, for instance on ubuntu run:
`sudo apt install python3-tk`

**Windows**:
Install python (3.10 for example) from the app manager.
Install cmake, following this guide or another (I had to restart
   my computer for it to work):
https://www.youtube.com/watch?v=8_X5Iq9niDE
Visual studio c++ must be installed, or the tools version.
https://stackoverflow.com/questions/63648184/error-installing-
    packages-using-pip-you-must-use-visual-studio-to-build-a-pyth
In case you run into this error you will have to go into the
   regedit (window+r then type regedit) to change the option for
   long file names, which requires administrator access.
```

```
https://stackoverflow.com/questions/72352528/how-to-fix-winerror-
    206-the-filename-or-extension-is-too-long-error
In case it is required to reinstall the packages to fix an
   issue, for instance with opencv-python: `pip uninstall
   opencv-python && pip install opencv-python`.

Then you can install the requirements,

### Usage guide

The script will be scanning the web and detecting the presence
   of the persons within our database.
If you just wish to verify your presence online, you can simply
   launch the main script, by running:
`python main.py` - followed by the arguments you want.
We recommend starting with --depthScan=0 to only scan the main
   page of websites, and then iterating, the system will only
   scan each picture once.
It is possible to give a list of urls to scan by adding the
   argument --urlList=url1,url2 by providing real urls
   naturally. These will only be scanned if the robots.txt files
   allow scraping.
Further options can be found by running `python main.py --help`
   for more information.
In case of doubt, consider first running the script by providing
   an offline file directory to scan (named offline_files by
   default) using `python main.py --offlineTests --noThread
   --showAll --terminalResults` to show the display as the
   process happens and display the logs live in the terminal.

By default only the basic search of the website's main page will
   be done, use `python main.py --scanDepth=1` to go further (we
   advise against numbers higher than 1, this is worse than
   exponantial)


For other uses, such as a company protecting its employees,
   consent of the persons present within the database is
   required in order to respect the GDPR and avoid many security
   risks, it should also be appropriately protected and
   encrypted, - rather than being a simple folder as shown in
   this prototype. Otherwise this may have the an opposite
   effect than intended with leaked faces being used to
   faciliate facial searches for these individuals, or produce
   deepfakes of the protected persons. This product is not ready
   for safe deployment at this stage, unless it is running in a
   protected environment with an operator we can trust not to
   abuse it.
First, gather the face data with your webcam, using the `python
   face.py` command. (This would be run automatically the first
   time you launch the main script without face data.)
Once it has collected enough facial data, by default 50, the
```

```
   program will close.
Then execute the main script, ‘python main.py‘ with the desired
   arguments. The web scraping will leave a log in the results
   folder named scan.txt that contains traces of authorized
   access to the different web pages, we recommend to avoid
   deleting it as well as the log.txt file.
Once the scanning is completed, an audit window will be
   displayed for the images that are uncertain. The treshold for
   this, as well as the model used for recognition, can be
   changed with parameters, we encourage running ‘python main.py
   --help‘ to understand the options in more details. During
   operation, the script will generate log.txt files, both in
   the results folder and in the protectees folder, when a given
   individual is identified either manually or through a high
   recognition level (70% by default).
When new faces are added to the database we can use ‘python
   main.py --reScan‘ to scan already accessed images again - or
   simply delete the scanned_urls.txt file. If you wish to run
   tests and handle them separately use the --incrementFolder
   option. You can also change the audit treshold with
   --auditTreshold=50 for instance with 50% - the default is
   60%, and save failed recognitions or just their faces in
   another folder using --storeFailed and --saveFaces.

### GPU usage
It is possible to use GPU acceleration, although this is for
   advanced users. Follow the instructions from this comment
‘pip uninstall onnxruntime-gpu onnxruntime‘
‘pip install onnxruntime-gpu --extra-index-url
   https://aiinfra.pkgs.visualstudio.com/PublicPackages/
    _packaging/onnxruntime-cuda-12/pypi/simple/‘
https://github.com/deepinsight/insightface/issues
    /2394#issuecomment-1929310317
Then go to the relevant NVIDIA pages and see the install
   instructions (for linux use the command lines suggested).
https://developer.nvidia.com/cuda-downloads
For proper functionning installing this is heavily recommended.
https://developer.nvidia.com/cudnn-downloads

You should be able to use the script with the GPU, note that
   this only works without threads, so follow this command:
‘python main.py --threadNbr=1
   --insightFaceProvider=CUDAExecutionProvider‘

Note that you should always use ‘--reScan‘ for consistent test
   results.

### Best Practices for Ethical Scraping

These are practises we should consider, to avoid legal and
   ethical problems when doing web scraping in this research
   context:
```

```
79
80  - **Respect Website Policies:** Always review and adhere to a
        website's terms of service and robots.txt file to determine
        permissible scraping activities.
81  - **Safeguard Data:** Protect the collected data against
        unauthorized access or breaches. Indeed our prototype will
        produce a file that will contain the results of the search,
        and should be safely guarded once it is deployed.
82  - **GDPR:** For web scraping researchers are left with more
        freedom than most, so long as no personal information has
        been collected, they do not have to send privacy notice under
        GDPR. They should be able and willing to demonstrate the
        proportionality. Researchers are also able to request large
        content provider platforms to provide them with access under
        certain conditions, with the DSA in Europe.
83  - **Privacy:** For now we do not keep the images, still we
        should never identify persons not from our database or not
        having given consent for us to do so. However, we should also
        add that the data collected should be of justifiable size and
        purpose. Under these considerations, the GDPR states that we
        do not have to inform and ask for consent when looking for
        information, due to the enormous effort that would be
        required. Indeed, this is not considered human subject
        research.
84  - **Picture context:** We should consider the context in which
        users will be placing pictures online, and not try to access
        any private pictures or "hot" websites. With all of these
        concerns, we should keep in mind we may miss some pictures,
        but there is so much data to find that we should have enough
        information already.
85  - **Website scale:** We should also be mindful not to place too
        much on a drain on websites themselves, and instead place
        most of the load on big websites if possible (for instance
        avoid scraping a small video platform). This is also a
        concern for us, which will be explored in the next section.
86  - **Variability:** Even though we are not doing machine learning
        we still have to concern ourselves with taking data from
        varied website sources in order to have a representative web
        presence.
87
88  ### Considerations for deploying this tool
89
90  As discussed in the preceding sections, the software
        accompanying this thesis will include a usage guide. This
        guide will be integrated into the README.md file and will
        provide essential instructions and ethical considerations for
        operating the script.
91
92  It is strongly recommended that users first read the associated
        master's thesis and the "Web Scraping for Researchers" guide
        by Brown (2024) to gain a foundational understanding of both
        web scraping and facial recognition technologies, as well as
```

```
    the context, objectives, and limitations of the tool. The
    important points of the state-of-the-art web scraping
    discussion (see Section \ref{scraping}) will also be
    reproduced in the guide for quick reference.
93
94  The purpose of the script is to scan publicly available websites
    to detect the presence of individuals listed in a predefined
    facial database. However, it is important to consider how to
    manage the outcomes of such detection. While it is
    technically feasible to compile all results into a single
    comprehensive report, doing so over a wide scope (e.g.,
    crawling the open web) may yield an excessive number of
    findings, leading to information overload. Furthermore, if
    the review process is handled by a single individual or
    centralized body, this may introduce bias or conflicts of
    interest.
95
96  To mitigate these risks, we recommend that the script be
    executed in smaller, controlled batches, and that the
    resulting reports be distributed directly to the relevant
    individuals. This decentralized approach empowers the data
    subjects to assess the findings and determine appropriate
    actions for themselves. The individuals should also be
    trained in how to use the tool, but also be made more aware
    of the privacy risks that come with our online lives, and be
    able to ask questions on these critical subjects to really
    grasp the consequences. It may seem like a stretch at first
    to imagine someone tracking our location from pictures, but
    when considering the amount of photos uploaded per year it
    sounds more plausible.
97
98  Crucially, explicit consent from all individuals included in the
    facial recognition database is required prior to deployment.
    To remain compliant with the General Data Protection
    Regulation (GDPR) and to uphold fundamental principles of
    cybersecurity, the database must be secured and encrypted.
    The use of an unprotected directory structure, as presented
    in this proof-of-concept implementation, is insufficient and
    poses a significant privacy risk. Without adequate
    safeguards, the tool could be misused to facilitate facial
    recognition surveillance against the very individuals it is
    meant to protect, or even contribute to the generation of
    deepfakes and other forms of synthetic identity abuse.
99
100 In summary, ethical deployment of this system requires informed
    consent, robust data protection, and responsible operational
    practices.
```

Appendix E

scraping.py

```
"""
Author: Alexandre De Groodt
Used for scraping the web for pictures and videos whilst making
    sure to respect the robot.txt rules.
"""

from bs4 import BeautifulSoup
from bs4 import XMLParsedAsHTMLWarning
import requests, shutil
from urllib.robotparser import RobotFileParser
from urllib.parse import urljoin, urlparse
from urllib.error import URLError, HTTPError
import time
import re
import warnings
# We choose to ignore this warning.
warnings.filterwarnings("ignore",
    category=XMLParsedAsHTMLWarning)


# Save a log and print the result if requested.
def Logs(log, text, terminal_results):
    log.write(text + "\n")
    if(terminal_results):
        print(text)

# Check if a page is probably private to avoid scraping it.
def LikelyPrivate(url, user_agent='*'):
    session = requests.Session()
    response = session.get(url, allow_redirects=True, timeout=10)

    # --- 1. Check for redirect to login/auth pages ---
    for r in response.history:
        if any(part in r.url.lower() for part in ['login',
            'signin', 'auth', 'account']):
            return True  # redirected to auth page

    final_url = response.url.lower()
    if any(part in final_url for part in ['login', 'signin',
        'auth', 'account']):
        return True

    # --- 2. Check HTTP status codes ---
    if response.status_code in [401, 403]:
        return True

```

```
    # --- 3. Check for private-looking URL patterns ---
    private_patterns = [
        r'/me/', r'/my/', r'/account', r'/settings', r'/dashboard',
        r'/profile/settings', r'/private', r'/admin', r'/user/.{20,}'  # long token
    ]
    for pattern in private_patterns:
        if re.search(pattern, final_url):
            return True

    # --- 4. Parse HTML content for robots meta tag & login forms ---
    soup = BeautifulSoup(response.content, 'html.parser')

    # Check meta robots
    meta = soup.find("meta", {"name": "robots"})
    if meta and "noindex" in meta.get("content", "").lower():
        return True

    # Check for login form (password input)
    if soup.find("input", {"type": "password"}):
        return True

    return False  # passed all checks -> likely public

def ScrapingAllowed(url, log, terminal_results, user_agent='*'):
    try:
        if(LikelyPrivate(url)):
            Logs(log, f"[STOPPED] This is likely a private page: {url}", terminal_results)
            return False
        parsed = urlparse(url)
        base_url = f"{parsed.scheme}://{parsed.netloc}"
        path = parsed.path or "/"  # Only the path, not the full URL

        robots_url = f"{base_url}/robots.txt"
        rp = RobotFileParser()
        rp.set_url(robots_url)
        rp.read()

        allowed = rp.can_fetch(user_agent, path)
        if not allowed:
            Logs(log, f"[BLOCKED] robots.txt disallows: {path} for {url}.", terminal_results)
        else:
            Logs(log, f"[ALLOWED] robots.txt allows: {path} for {url}.", terminal_results)
        return allowed
    except Exception as e:
        Logs(log, f"[WARNING] robots.txt could not be checked
```

```
            for {url} (reason: {e}) - defaulting to allowed",
               terminal_results)
        return True


def UrlToName(url):
    parse = urlparse(url)
    name = parse.netloc + parse.path
    return name.replace(".", "_").replace("/", "_")

# Readies all images and videos of the given url for processing,
   if allowed.
def ScrapeImgVid(url, log, scanned_urls, skip_img, skip_vid,
   terminal_results):
    img_to_scrape, vid_to_scrape, images, videos = [], [], [], []
    file_name = UrlToName(url)
    try:
        response = requests.get(url)
        if response.status_code != 200:
            Logs(log, f"[ERROR] Failed to retrieve {url}, status
               code: {response.status_code}", terminal_results)
            return
    except Exception as exception:
        Logs(log, f"Exception during html querying:
           {exception}", terminal_results)
        return
    soup = BeautifulSoup(response.content, 'html.parser')
    if(not skip_img):
        images = soup.find_all('img')
        if not images:
            Logs(log, f"[INFO] No images found at {url}",
               terminal_results)
    if(not skip_vid):
        videos = soup.find_all('vid')
        if not videos:
            pass#Logs(log, f"[INFO] No videos found at {url}",
               terminal_results) - this is quite common

    for image in images:
        skip = False
        img_url = image.get('src') or image.get('data-src') or
           image.get('data-lazy-src')

        if not img_url:
            continue

        full_url = urljoin(url + "/", img_url)
        if(scanned_urls is not None):
            for url_scanned in scanned_urls:
                if(url_scanned == full_url):
                    skip = True
                    break
```

```
            if(not skip):
                img_to_scrape.append((full_url, file_name))

        for video in videos:
            vid_url = video_tag.find("a")['href']
            if not vid_url:
                continue

            full_url = urljoin(url + "/", vid_url)
            if(scanned_urls is not None):
                for url_scanned in scanned_urls:
                    if(url_scanned == full_url):
                        continue
            vid_to_scrape.append((full_url, file_name))

        return (img_to_scrape, vid_to_scrape)


def NormalizeUrl(url):
    return url.split('#')[0].rstrip('/')

def IsInternalLink(link, base_domain):
    parsed = urlparse(link)
    return parsed.netloc == '' or parsed.netloc == base_domain

def RecursiveScrape(info):
    seed_url, LOG, max_depth, scanned_urls, skip_img, skip_vid,
       terminal_results, polite = info
    log = open(LOG, "a+")
    visited = set()
    to_visit = [(seed_url, 0)]
    if(seed_url[:4] != "http"):
        print(f"[ERROR] Invalid website link without http,
           {seed_url}", terminal_results)
        to_visit = []
    elif(not ScrapingAllowed(seed_url, log, terminal_results)):
        print(f"[BLOCKED] Scraping not allowed for website
           {seed_url}, advising to remove it from the list.")
        to_visit = []
    all_img_data, all_vid_data = [], []
    while to_visit:
        current_url, depth = to_visit.pop(0)
        normalized = NormalizeUrl(current_url)

        # Check that it is indeed an url.
        if normalized in visited:
            continue
        visited.add(normalized)

        Logs(log, f"[VISIT] {current_url} (depth {depth})",
           terminal_results)

```

```
179         try:
180             result = ScrapeImgVid(current_url, log,
                    scanned_urls, skip_img, skip_vid,
                    terminal_results)
181             if result:
182                 img_data, vid_data = result
183                 all_img_data.extend(img_data)
184                 all_vid_data.extend(vid_data)
185
186             # Fetch links to recurse into
187             response = requests.get(current_url)
188             soup = BeautifulSoup(response.content, 'html.parser')
189             base_domain = urlparse(seed_url).netloc
190
191             for link_tag in soup.find_all('a', href=True):
192                 href = link_tag.get('href')
193                 joined = urljoin(current_url, href)
194                 joined = joined.strip('\t ')
195                 if(depth + 1 > max_depth or joined[:4] !=
                        "http"):
196                     continue
197                 if(not ScrapingAllowed(joined, log,
                        terminal_results)):
198                     Logs(log, f"[BLOCKED] Scraping not allowed
                            for {joined}", terminal_results)
199                     continue
200                 normalized_link = NormalizeUrl(joined)
201                 if(IsInternalLink(joined, base_domain) and
                        normalized_link not in visited):
202                     to_visit.append((joined, depth + 1))
203
204         except Exception as e:
205             Logs(log, f"[ERROR] While processing {current_url}:
                    {e}", terminal_results)
206
207         if(polite):
208             time.sleep(1)
209     log.flush()
210     log.close()
211     return all_img_data, all_vid_data
```

Appendix F

ui.py

```
"""
Author: Alexandre De Groodt
Code for the UI part of the project, handles the display of images.
By default it only shows those with a likelihood between 50 and 70%, for which we need to confirm the identiy.
"""

import cv2
import numpy as np
import os
import tkinter as tk
from PIL import Image, ImageTk
from tkinter import filedialog
from datetime import datetime
import re, io

# Used to draw a rectangle of the given color around a face.
# Add a black rectangle behind the text when in the editor to make it more visible.
def DrawRectangle(image, name, color, x, y, w, h, edit = 0):
    if(edit):
        cv2.rectangle(image, (x, y - 55), (x + 150, y - 30), (0, 0, 0), cv2.FILLED)
    cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
    cv2.putText(image, name, (x, y - 10 - edit * 20), cv2.FONT_HERSHEY_SIMPLEX, 0.8, color, 2)

# Strip the extension and number of the video to find the common name they all have.
# For instance, frame0.jpg, frame1.jpg would give frame.
def VideoStrip(video_name):
    folder_name = os.path.splitext(video_name)[0]
    folder_name = re.sub(r'\d+$', '', folder_name)
    return folder_name

# Main window.
class ImageNavigator:
    def __init__(self, master, folder, reference_folder, operator, size):
        self.master = master
        self.folder = folder
        self.reference_folder = reference_folder
        # The audit logs are stored within the foler holding personal pictures as well.
        self.audit_logs = open(self.reference_folder +
```

```
            "audit_logs.txt", "a+")
        self.audit = open(self.folder + "audit.txt", "r")
        self.operator = operator
        self.audit_logs.write(f"Log by {self.operator} start
            time: {datetime.now()}\n")
        self.master.title("Image Navigator")
        self.size = size
        self.master.geometry(f"{size}x{size}")

        self.images = [line.strip().split(':') for line in
            self.audit.readlines() if not
            int(line.strip().split(':')[-1])]
        self.current_image_index = 0

        self.canvas = tk.Canvas(self.master, bg='white')
        self.canvas.pack(fill=tk.BOTH, expand=True)

        # Navigation, and confirmation button.
        self.prev_button = tk.Button(self.master,
            text="Previous", command=self.ShowPreviousImage)
        self.prev_button.pack(side=tk.LEFT, padx=10)
        self.next_button = tk.Button(self.master, text="Next",
            command=self.ShowNextImage)
        self.next_button.pack(side=tk.RIGHT, padx=10)
        self.confirm_button = tk.Button(self.master,
            text="Confirm", command=self.ConfirmationDialog)
        self.confirm_button.pack(side=tk.BOTTOM, padx=10)

        # Bind left mouse key and keyboard left and right arrows.
        self.master.bind("<Button-1>", self.OnClick)
        self.master.bind('<Left>', self.ShowPreviousImage)
        self.master.bind('<Right>', self.ShowNextImage)
        if(len(self.images)):
            self.ShowImage()
        else:
            print("No images to scan.")
            self.master.destroy()
            self.__del__()

    def ShowImage(self):
        self.face_selected = False
        self.video = False
        self.current_image_index = self.current_image_index %
            len(self.images)
        self.image_name =
            self.images[self.current_image_index][0]
        image_path = os.path.join(self.folder + "found/",
            self.image_name)

        # If we cannot find here, it is either a video or in the
            miss folder.
        if(not os.path.exists(image_path)):
```

```
            image_path = os.path.join(self.folder + "miss/",
                self.image_name)
            if(not os.path.exists(image_path)):
                self.video = True
                folder_name = VideoStrip(self.image_name)
                image_path = os.path.join(self.folder +
                    folder_name + "/found/", self.image_name)
        load = Image.open(image_path)

        render = ImageTk.PhotoImage(load)

        self.canvas.delete("all")
        self.x_coord = (self.size - load.width) // 2
        self.y_coord = (self.size - load.height) // 2
        self.canvas.create_image(self.x_coord, self.y_coord,
            anchor=tk.NW, image=render)
        self.canvas.image = render

        # Keep it to redraw it quickly in onClick().
        self.last_loaded_image = load

        self.array = np.array(load)
        self.array = cv2.cvtColor(self.array, cv2.COLOR_BGR2RGB)

        self.selected_face = False

        self.master.title(f"Image Navigator: {self.image_name}")
        self.images[self.current_image_index][-1] = 1

    # We need the event for the keyboard input, but not for the
        clicks.
    def ShowPreviousImage(self, event=None):
        while self.images[self.current_image_index][0] ==
            self.image_name:
            self.current_image_index = (self.current_image_index
                - 1) % len(self.images)
        # Make sure we're always on the first audit log line for
            this image.
        new_name = self.images[self.current_image_index][0]
        while self.images[self.current_image_index][0] ==
            new_name:
            self.current_image_index = (self.current_image_index
                - 1) % len(self.images)
        else:
            self.current_image_index += 1
        self.ShowImage()

    def ShowNextImage(self, event=None):
        while self.images[self.current_image_index][0] ==
            self.image_name:
            self.current_image_index = (self.current_image_index
                + 1) % len(self.images)
```

```
121         # If this is a video skip all frames by going to the
                right.
122         if(self.video):
123             stripped_name = VideoStrip(self.image_name)
124             while
                    VideoStrip(self.images[self.current_image_index][0])
                    == stripped_name:
125                 self.current_image_index =
                        (self.current_image_index + 1) %
                        len(self.images)
126         self.ShowImage()
127
128     def ConfirmationDialog(self):
129         if(self.face_selected == True):
130             image_info = self.images[self.current_image_index +
                    self.selected_face]
131             name = image_info[1]
132             if(tk.messagebox.askyesno("Confirmation", f"Send a
                    mail to {name} to warn them of their presence in
                    this picture?")):
133                 cv2.imwrite(os.path.join(f"{self.reference_folder}{name}/",
                        f"{self.image_name}_confirmed_face.jpg"),
                        self.array)
134                 mail =
                        open(f"{self.reference_folder}{name}/mail.txt",
                        "a+")
135                 message = f"The presence of {name} was confirmed
                        in {self.image_name} by {self.operator}."
136                 mail.write(message)
137                 mail.close()
138                 self.audit_logs.write(message + f" At
                        {datetime.now()}.")
139                 print(message)
140                 self.ShowNextImage()
141
142     def OnClick(self, event):
143         click_x = event.x - self.x_coord
144         click_y = event.y - self.y_coord
145
146         if click_x < 0 or click_y < 0 or click_x >=
                self.last_loaded_image.width or click_y >=
                self.last_loaded_image.height or self.face_selected:
147             return  # Click outside image boundsa
148
149         self.selected_face = 0
150         while (self.current_image_index + self.selected_face <
                len(self.images)) and
                self.images[self.current_image_index +
                self.selected_face][0] == self.image_name:
151             index = self.current_image_index + self.selected_face
152             x, y, w, h = int(self.images[index][-5]),
                    int(self.images[index][-4]),
```

```
                int(self.images[index][-3]),
                int(self.images[index][-2])
            if x <= click_x <= x + w and y <= click_y <= y + h:
                # Draw a new rectangle on a copy of the image
                img = np.array(self.last_loaded_image.copy())

                DrawRectangle(img, "confirm", (0, 0, 255), x, y,
                    w, h, True)

                # Convert back to display
                updated_image =
                    ImageTk.PhotoImage(Image.fromarray(img))
                self.canvas.delete("all")
                self.canvas.create_image(self.x_coord,
                    self.y_coord, anchor=tk.NW,
                    image=updated_image)
                self.canvas.image = updated_image
                self.face_selected = True
                break
            self.selected_face += 1

    def __del__(self):
        if(self.images):
            # Update the audit file by only re-adding the files
                we did not look at yet.
            with io.open(self.folder + "audit.txt", "w") as
                audit:
                for image in self.images:
                    if int(image[-1]) == 0:
                        to_write = ""
                        for i in image:
                            to_write += str(i) + ":"
                        to_write = to_write[:-1] + "\n"
                        audit.write(to_write)
            audit.close()

# Example usage.
if __name__ == "__main__":
    root = tk.Tk()
    img_folder = "path_to_your_images_folder"
    app = ImageNavigator(root, img_folder)
    root.mainloop()
```

Appendix G

scripts/app.py

```
"""
Author: Alexandre De Groodt

This script scans for pictures on one's facebook profile for the faces of the protected person list.
"""

import requests
import cv2
import numpy as np
import os

ACCESS_TOKEN = ""
GRAPH_API_URL = "https://graph.facebook.com/v22.0/me/posts?fields="+ "full_picture,name,place&access_token=" + ACCESS_TOKEN

# Load OpenCV's pre-trained face detector
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

# Load known faces
KNOWN_FACES_DIR = "../protectees"
known_face_descriptors = []
known_face_names = []
orb = cv2.ORB_create(nfeatures=1000)
sift = cv2.SIFT_create()

for file_name in os.listdir(KNOWN_FACES_DIR):
    if file_name.endswith((".jpg", ".png")):
        image_path = os.path.join(KNOWN_FACES_DIR, file_name)
        known_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        if known_image is None:
            print(f"Error loading image: {image_path}")
            continue
        known_image = cv2.equalizeHist(known_image)
        keypoints, descriptor = orb.detectAndCompute(known_image, None)

        # If ORB fails, use SIFT
        if descriptor is None or len(keypoints) == 0:
            keypoints, descriptor = sift.detectAndCompute(known_image, None)

        if descriptor is not None:
```

```
            # Convert all descriptors to float32 for compatibility
            descriptor = descriptor.astype(np.float32)
            known_face_descriptors.append(descriptor)
            known_face_names.append(os.path.splitext(file_name)[0])

# Function to check if a photo was taken in Belgium
def is_photo_in_belgium(photo):
    if "place" in photo and "location" in photo["place"]:
        location = photo["place"]["location"]
        return location.get("country") == "Belgium"
    return False

# Fetch photos from Facebook API
response = requests.get(GRAPH_API_URL)
data = response.json()
print(data)

for photo in data.get("data", []):
    image_url = photo.get("images", [{}])[0].get("source") or photo.get("full_picture")
    if not image_url:
        continue

    img_response = requests.get(image_url)
    img_array = np.asarray(bytearray(img_response.content), dtype=np.uint8)
    image = cv2.imdecode(img_array, cv2.IMREAD_COLOR)
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Detect faces
    faces = face_cascade.detectMultiScale(gray_image, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    for (x, y, w, h) in faces:
        face_roi = gray_image[y:y + h, x:x + w]  # Extract face region

        keypoints, descriptor = orb.detectAndCompute(face_roi, None)
        if descriptor is None or len(keypoints) == 0:
            keypoints, descriptor = sift.detectAndCompute(face_roi, None)

        if descriptor is not None:
            # Convert to float32
            descriptor = descriptor.astype(np.float32)

            bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)  # NORM_L2 for SIFT
            best_match_count = 0
            name = "Unknown"
```

```
85
86              for i, known_descriptor in
                    enumerate(known_face_descriptors):
87                  # Ensure known descriptor is also float32
88                  known_descriptor =
                        known_descriptor.astype(np.float32)
89
90                  # **Ensure same number of columns**
91                  if descriptor.shape[1] !=
                        known_descriptor.shape[1]:
92                      continue  # Skip if dimensions don't match
93
94                  matches = bf.match(descriptor, known_descriptor)
95                  if len(matches) > best_match_count:
96                      best_match_count = len(matches)
97                      name = known_face_names[i]
98
99          # Draw rectangle and label
100         cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255,
                0), 2)
101         cv2.putText(image, name, (x, y - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
102
103     cv2.imshow('Face Detection - Belgium Only', image)
104     cv2.waitKey(0)
105     cv2.destroyAllWindows()
```

Appendix H

requirements.txt

```
albucore==0.0.24
albumentations==2.0.7
annotated-types==0.7.0
beautifulsoup4==4.13.4
bs4==0.0.2
certifi==2025.4.26
charset-normalizer==3.4.2
cmake==4.0.2
colorama==0.4.6
coloredlogs==15.0.1
contourpy==1.3.2
cycler==0.12.1
Cython==3.1.1
dlib==19.24.9
easydict==1.13
flatbuffers==25.2.10
fonttools==4.58.0
humanfriendly==10.0
idna==3.10
imageio==2.37.0
imutils==0.5.4
insightface==0.7.3
joblib==1.5.1
kiwisolver==1.4.8
lazy_loader==0.4
matplotlib==3.10.3
mpmath==1.3.0
networkx==3.4.2
numpy==2.2.6
onnx==1.18.0
onnxruntime==1.22.0
opencv-python==4.11.0.86
opencv-python-headless==4.11.0.86
packaging==25.0
pillow==11.2.1
prettytable==3.16.0
protobuf==6.31.0
pydantic==2.11.5
pydantic_core==2.33.2
pyparsing==3.2.3
pyreadline3==3.5.4
python-dateutil==2.9.0.post0
PyYAML==6.0.2
requests==2.32.3
scikit-image==0.25.2
scikit-learn==1.6.1
```

```
scipy==1.15.3
simsimd==6.2.1
six==1.17.0
soupsieve==2.7
stringzilla==3.12.5
sympy==1.14.0
threadpoolctl==3.6.0
tifffile==2025.5.10
tqdm==4.67.1
typing-inspection==0.4.1
typing_extensions==4.13.2
urllib3==2.4.0
wcwidth==0.2.13
```

Appendix I

.gitignore

```
venv/*
results*/
protectees/*
offline_files/*
__pycache__/
scanned_urls.txt
.utils.py.*
log.txt
```

Bibliography

[1] A. CREFF, G.C.: Publication date: 2025-05-24. , Last accessed: 2025-05-28. , https://www.tf1.fr/fr-be/tf1/jt-20h/videos/reconnaissance-faciale-jusquou-peut-on-aller-73127861.html, section: Journal de 20 heures [Cited on pages 3 and 20.]

[2] Abdelbar, A., et al.: Publication date: 2024-10-22. Last accessed: 2025-03-26. [Cited on pages 6, 14, 16, and 21.]

[3] Anton: , Last accessed: 2025-05-30. , https://www.hackster.io/gr1m/raspberry-pi-facial-recognition-16e34e [Cited on pages VI and 8.]

[4] Big Brother Watch: Publication date: 2023-05-23. , Last accessed: 2025-05-13. , https://www.youtube.com/watch?v=bX-Yxy1ESAQ [Cited on page 3.]

[5] bigvisionai: , Last accessed: 2025-05-26. , https://colab.research.google.com/github/bigvisionai/upgrad_alumni_workshop_day2/blob/master/face_recognition/OpenCV_DNN_Face_Detection_Recognition.ipynb [Cited on page 36.]

[6] Bivona, A.: Publication date: 2020-11-21. , Last accessed: 2025-05-26. , https://towardsdatascience.com/a-tutorial-on-scraping-images-from-the-web-using-beautifulsoup-206a7633e948/ [Cited on page 47.]

[7] Bolhasani, H., Marandinejad, M.: Deep neural networks accelerators with focus on tensor processors Publication date: 2024-03-01. , Last accessed: 2025-05-30. [Cited on pages VI and 7.]

[8] Bonifacic, I.: Publication date: 2020-02-05. , Last accessed: 2025-04-27. , https://www.proquest.com/docview/2351699400, place: New York, United States. Publisher: Apollo Global Management [Cited on page 4.]

[9] Brown, M.A., Gruen, A., Maldoff, G., Messing, S., Sanderson, Z., Zimmer, M.: Publication date: 2024-12-19. , Last accessed: 2025-05-13. , http://arxiv.org/abs/2410.23432 [Cited on pages 18, 19, and 33.]

[10] Brownlee, J.: Publication date: 2022-07-10. , Last accessed: 2025-06-01. , https://superfastpython.com/multiprocessing-pool-map_async/ [Cited on page 50.]

[11] Byler, D.: Producing 'enemy intelligence': Information infrastructure and the smart city in northwest china Publication date: 2022. , place: Austin Publisher: University of Texas Press [Cited on pages 3 and 20.]

[12] Carattino, A.: , Last accessed: 2025-06-01. , https://pythonforthelab.com/blog/differences-between-multiprocessing-windows-and-linux [Cited on page 44.]

[13] Castro, Daniel, M.M.: Publication date: 2020-01-27. , Last accessed: 2025-05-11. , https://itif.org/publications/2020/01/27/critics-were-wrong-nist-data-shows-best-facial-recognition-algorithms/ [Cited on pages 16 and 17.]

[14] Clayton, T.: Publication date: 2025-02-11. , Last accessed: 2025-05-12. , https://rigorousthemes.com/blog/best-pimeyes-alternatives/ [Cited on page 8.]

[15] Clothier, E., Michalski, D., Malec, C., Nowina-Krowicki, M.: Are contemporary facial recognition algorithms making human facial comparison performance worse? Publication date: 2024. , place: Ireland Publisher: Elsevier B.V [Cited on page 22.]

[16] CNN Business: Publication date: 2020-02-10. , Last accessed: 2025-04-28. , https://www.youtube.com/watch?v=pGJNXG2vmZw [Cited on pages 8, 17, and 20.]

[17] Commission, E.: Publication date: 2025-05-23. , Last accessed: 2025-05-28. , https://digital-strategy.ec.europa.eu/en/policies/regulatory-framework-ai [Cited on pages 3, 9, and 18.]

[18] Daniels, T.: Publication date: 2023-07-13. , Last accessed: 2025-06-02. , https://www.lapseoftheshutter.com/photography-statistics/, section: Articles [Cited on page 1.]

[19] Edmond, G., White, D., Towler, A., Roque, M.S., Kemp, R.: Facial recognition and image comparison evidence: Identification by investigators, familiars, experts, super-recognisers and algorithms Publication date: 2021. , Last accessed: 2025-05-12. , num Pages: 99-160 Place: Melbourne, Australia Publisher: Melbourne University Law Review Association Inc. [Cited on page 22.]

[20] of Europe, C.: , Last accessed: 2025-05-28. , https://www.coe.int/en/web/artificial-intelligence [Cited on page 3.]

[21] Evtimov, I., Sturmfels, P., Kohno, T.: FoggySight: A scheme for facial lookup privacy Publication date: 2021-07-01. , Last accessed: 2025-04-27. [Cited on page 25.]

[22] Financieras, C.N.: Publication date: 2022-06-13. Last accessed: 2025-04-27. , publisher: ContentEngine LLC, a Florida limited liability company [Cited on pages 1 and 8.]

[23] Firc, A., Malinka, K., Hanáček, P.: Deepfakes as a threat to a speaker and facial recognition: An overview of tools and attack vectors Publication date: 2023-04. , Last accessed: 2025-03-23. [Cited on page 22.]

[24] Fussey, P., Davies, B., Innes, M.: 'assisted' facial recognition and the reinvention of suspicion and discretion in digital policing Publication date: 2021-02-26. , Last accessed: 2025-03-23. [Cited on page 22.]

[25] Fábián, I., Gulyás, G.G.: A comparative study on the privacy risks of face recognition libraries Publication date: 2021. , place: Szeged Publisher: Laszlo Nyul [Cited on pages 15, 17, and 47.]

[26] gracelm: Publication date: 2021-11-02. , Last accessed: 2025-05-28. , https://about.fb.com/news/2021/11/update-on-use-of-face-recognition/ [Cited on page 24.]

[27] Grother, P., Ngan, M., Hanaoka, K.: Publication date: 2019-12. , Last accessed: 2025-05-30. , https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8280.pdf [Cited on page 17.]

[28] Hak5: Publication date: 2020-06-15. , Last accessed: 2025-05-13. , https://www.youtube.com/watch?v=tbdcL5Ux-9Y [Cited on page 24.]

[29] Helper, D.: , Last accessed: 2025-05-13. , https://www.dmca.com/FAQ/What-is-a-DMCA-Takedown [Cited on page 9.]

[30] Hill, K.: Publication date: 2021-11-23. Last accessed: 2025-06-01. [Cited on page 7.]

[31] JIA, P.: , Last accessed: 2025-06-01. , https://github.com/deepinsight/insightface/issues/2394 [Cited on page 51.]

[32] Kumar, H.: , Last accessed: 2025-05-30. , https://kharshit.github.io/blog/2018/09/28/generative-models-and-generative-adversarial-networks [Cited on pages VI and 15.]

[33] Learning, G.: , Last accessed: 2025-05-26. , https://www.mygreatlearning.com/academy/learn-for-free/courses/face-detection-with-opencv-in-python [Cited on page 34.]

[34] Li, M.: Research and analysis of facial recognition based on FaceNet, DeepFace, and OpenFace Publication date: 2025. , Last accessed: 2025-04-01. , publisher: EDP Sciences [Cited on pages 6, 14, and 15.]

[35] Logan, B.E.: Publication date: 2025-04-10. , Last accessed: 2025-05-13. , https://en.wikipedia.org/w/index.php?title=HiQ_Labs_v._LinkedIn&oldid=1284918689, page Version ID: 1284918689 [Cited on page 18.]

[36] MalcomVetter, T.: Publication date: 2024-02-20. , Last accessed: 2025-05-30. , https://malcomvetter.medium.com/deep-deep-fakes-d4507c735f44 [Cited on pages VI and 21.]

[37] McSorley, T.: The case for a ban on facial recognition surveillance in canada Publication date: 2021. , place: Kingston Publisher: Surveillance Studies Network [Cited on pages 1, 3, 4, and 23.]

[38] Media, O.: , Last accessed: 2025-05-30. , https://embeddedcomputing.com/technology/ai-machine-learning/ai-logic-devices-worload-acceleration/iwave-i-mx8m-mini-board-with-nxp-eiq-ml-software-enables-low-cost-facial-recognition-s [Cited on pages VI and 6.]

[39] Mitchum, R.: Publication date: 2020-08-03. , Last accessed: 2025-05-13. , https://news.uchicago.edu/story/new-tool-protect-yourself-against-facial-recognition-software [Cited on page 10.]

[40] N, A., Anusudha, K.: Real time face recognition system based on YOLO and InsightFace Publication date: 2024. , place: New York Publisher: Springer US [Cited on pages 7 and 16.]

[41] Nachmani, O., Saun, T., Huynh, M., Forrest, C.R., McRae, M.: "facekit"—toward an automated facial analysis app using a machine learning–derived facial recognition algorithm Publication date: 2023-11. , Last accessed: 2025-03-23. [Cited on pages 16 and 21.]

[42] Nagella, V.S.: Publication date: 2019-12-26. , Last accessed: 2025-05-25. , https://medium.com/@sasi24/cosine-similarity-vs-euclidean-distance-e5d9a9375fc8 [Cited on page 38.]

[43] Nonis, F., Dagnes, N., Marcolin, F., Vezzetti, E.: 3d approaches and challenges in facial expression recognition algorithms—a literature review Publication date: 2019-09-18. , Last accessed: 2025-03-23. [Cited on pages 16 and 21.]

[44] Rezende, I.N.: Facial recognition in police hands: Assessing the 'clearview case' from a european perspective Publication date: 2020-09-01. , Last accessed: 2025-04-29. , publisher: SAGE Publications Ltd STM [Cited on pages 1, 9, 18, and 23.]

[45] Rosebrock, A.: Publication date: 2021-04-19. , Last accessed: 2025-05-26. , https://pyimagesearch.com/2021/04/19/face-detection-with-dlib-hog-and-cnn/ [Cited on page 47.]

[46] Roussi, A.: Resisting the rise of facial recognition Publication date: 2020-11-19. , Last accessed: 2025-03-23. [Cited on page 3.]

[47] Saleem, A.: , Last accessed: 2025-05-30. , https://datasciencedojo.com/blog/eu-ai-act/ [Cited on pages VI and 2.]

[48] Schuetz, P.N.: Publication date: 2021. Last accessed: 2025-03-23. [Cited on pages 16 and 17.]

[49] Shan, S., Wenger, E., Zhang, J., Li, H., Zheng, H., Zhao, B.Y.: Publication date: 2020-06-23. Last accessed: 2025-05-13. , place: Ithaca, United States Publisher: Cornell University Library, arXiv.org Section: Computer Science; Statistics University: Cornell University Library arXiv.org [Cited on pages 1, 24, and 25.]

[50] Smyth, J.C.: Publication date: 2021-05-05. , Last accessed: 2025-05-13. , https://apnews.com/article/race-and-ethnicity-health-coronavirus-pandemic-business-technology-e4266250f7e2d691d4d [Cited on page 3.]

[51] Swan, B.: Publication date: 2020-02-26. , Last accessed: 2025-04-27. , https://www.proquest.com/docview/2364730449/citation/41CE0967427149ABPQ/1, place: New York, United States Publisher: The Newsweek/Daily Beast Company LLC [Cited on page 23.]

[52] Tolosana, R., Vera-Rodriguez, R., Fierrez, J., Morales, A., Ortega-Garcia, J.: Deepfakes and beyond: A survey of face manipulation and fake detection Publication date: 2020-12. , Last accessed: 2025-03-23. [Cited on page 21.]

[53] Vadlapati, J., Senthil Velan, S., Varghese, E.: Facial recognition using the OpenCV libraries of python for the pictures of human faces wearing face masks during the COVID-19 pandemic. In: 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT). Publication date: 2021-07. . Last accessed: 2025-05-13. [Cited on pages 10 and 26.]

[54] Wankhede, C.: Publication date: 2022-08-16. , Last accessed: 2025-04-28. , https://www.androidauthority.com/face-unlock-smartphones-3043993 [Cited on page 6.]

[55] WIRED: Publication date: 2022-07-06. , Last accessed: 2025-05-13. , https://www.youtube.com/watch?v=9Xg-7FfLIVw [Cited on page 3.]

[56] Wu, W., Peng, H., Yu, S.: YuNet: A tiny millisecond-level face detector Publication date: 2023-10-01. , Last accessed: 2025-04-01. [Cited on page 15.]