Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Losing Individuality: The Need for Facial Obfuscation

Vitali Borsak

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science

Supervisor: Dr. Inmaculada Arnedillo-Sanchez

April 2024

Losing Individuality: The Need for Facial Obfuscation

Vitali Borsak, Master of Science in Computer Science

University of Dublin, Trinity College, 2024

Supervisor: Dr. Inmaculada Arnedillo-Sanchez

With the rapid advancements of the computer vision field, particularly the area of face detection and recognition, great concern for privacy has arose. With the introduction of face recognition functionality into surveillance systems which are being widely adopted, one wonders what happens to people's individuality. With constant monitoring and profiling, people will become more worried about expressing themselves so that their personal record adheres to societal norms. This impact is even greater for kids whose uniqueness could be damaged as they would want to ensure that they explore similar things to their peers as to not seem like an outsider which can greatly hinder their development.

The motivation behind this paper comes exactly from that, to assist my professor with her research, regarding motor development in children and how it can hinder their cognitive development. I was tasked to create a tool that would process data of children performing various tasks that would obfuscate these children's faces to hide their identities and protect their privacy and individuality. The obfuscation however, needed to be limited to a point where the location of facial features could still be detectable for future research.

In this paper I delve into the various face detection and obfuscation techniques along with available frameworks that implement these computer vision methods to develop a tool that can be easily utilized to protect children's privacy and potentially have future more advanced implementations.

Acknowledgments

I would like to acknowledge and give my warmest thanks to my supervisor Dr. Inmaculada Arnedillo-Sanchez who made this work possible and also for her time, guidance, and patience.

I would also like to give special thanks to my family for their continuous support and understanding when undertaking my research and writing project.

Vitali Borsak

University of Dublin, Trinity College
April 2024

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Facial detection and recognition technology is increasingly becoming an integral aspect of our daily lives, which can be observed in various domains ranging from cafes to education and advertisement. Bolger (2018) covered how facial recognition technology was employed in a cafe to record regular customers' typical orders and begin preparing them in advance when they enter. Moreover, 'smart billboards' are another innovation that utilizes facial recognition technology. Smart billboards detect the emotional states of passersby, to then display advertisements on said billboards tweaked to fit the viewer's current mood. The advancement of facial recognition technology has bolstered marketing strategies along with underlining the technology's versatility in various environments, allowing for more complex computations such as analyzing viewer's moods to produce a robust marketing strategy. Alongside the marketing sector benefiting from facial recognition, the educational sector has also adopted this technology to streamline the attendance tracking process, along with assessing students' emotional engagement during lectures (Andrejevic and Selwyn (2019)). These innovative ideas are just some of the endless potential applications of facial detection, that are only scratching the surface of the potential and innovation that this technology introduces.

The recent advancement in Facial Recognition Systems (FRS) can be attributed to the enhancements in the quality of video and image capturing hardware, along with continued improvements in computer vision capabilities such as face detection and recognition. The technological breakthrough caused a spike in security measures as surveillance equipment has become capable of producing high-quality video at a lower cost. According to Lin and Purnell (2019), it is projected that the global count of surveillance cameras will surpass 1 billion, with over half of these located in China and more than 85 million within the United States, underscoring the rapid growth of surveillance infrastructure. Alongside

surveillance cameras, FRS technology is being incorporated with body cameras in law enforcement (Harwell (2018)). Specifically, police departments in London, South Wales, Detroit, and Orlando are the first to pilot these integrated systems, signalling a potential broader adoption of FRS in law enforcement practices (Kaste (2018); Burgess (2018); Harmon (2019)). While the widespread adaptation of surveillance systems fosters a secure environment, it simultaneously raises the implications for privacy and individuality. Finding the correct balance between public safety and personal privacy rights is extremely tough and should be approached with utmost care.

Modern FRS predominantly incorporate Artificial Intelligence and Machine Learning algorithms, relying on their superior accuracy and pattern recognition capabilities. While certain functionalities such as identifying and cataloguing criminals are undeniably extremely important, however, it pose a significant risk to the privacy of ordinary citizens. The functionality of FRS can be extended to record individual's everyday activities, which drastically impacts their right to privacy. Such systems, as Kaplan (2023) explains, effectively completely remove the concept of anonymity for individuals in public spaces. The implications of FRS on individual data are far and wide, it captures the individual's preferences, frequently visited locations, modes of transport taken, social interactions, and other behavioural patterns. This essentially removes the person's right to privacy by profiling every individual. Great ethical considerations are needed when deploying a system that implements facial recognition.

The highest concern surrounding the rapid adaptation of FRS surrounds the implications that it may have on the natural development trajectory of children. The nature of this technology ensures that the children are profiled through their hobbies and interests, meaning that the evolving identities of the young individuals are carefully tracked from a very early age, creating a detailed digital footprint that will accompany them throughout their lives. Such surveillance poses a risk to their natural development process, as the awareness that their actions will be recorded and permanently stored on their digital records, might force them to suppress exploring new interests and expressing themselves freely, thereby hindering their growth as a unique individual. Based on this, we can deduce that it is important to prioritize the integration of obfuscation into FRS, to specifically safeguard privacy for all individuals, especially children, to not hinder their natural development. This ensures that privacy concerns are addressed while creating a landscape for future FRS development which will respect the fundamental rights to privacy and freedom to express and pursue new interests.

One major challenge for modern FRS involves natural obfuscation of critical facial features such as sunglasses, scarves, and face masks. This issue became particularly prominent following the COVID-19 pandemic in early 2020 (Marco Ciotti (2020)), which led to a widespread public adaptation of face masks to lower infection rates, which in turn, lowers COVID-19-related mortality (Worby and Chang (2020)). Despite the pandemic's conclusion and the removal of mandated mask-wearing, many people have chosen to continue wearing face masks as a precaution. This major global event necessitated the consideration for training data with both masked and unmasked faces for machine-learning-based FRS.

1.1 Project Area

Fundamentally, facial recognition technology operates by analyzing images that are extracted as frames from either live or pre-recorded video streams. The initial step in this process involves the detection and subsequent extraction of the face or faces from the framed area. Once identified, these facial images undergo a series of preprocessing steps designed to normalize variations in lighting conditions, thereby enhancing the reliability of feature extraction. Following the meticulous extraction of facial features, the technology proceeds to the pivotal stage of identity recognition. This is accomplished through the employment of sophisticated classification methodologies, which analyze the extracted features to ascertain the identity of the individual(s) in question. Such a comprehensive approach, as outlined by (Shang-Hung (2000)), underscores the intricate process underlying facial recognition technology, from initial image capture to the final identification of individuals based on their unique facial features.

Over the recent years, deep artificial neural networks have distinguished themselves by securing numerous victories in competitions (Schmidhuber (2014)), demonstrating superior performance over all other subclasses of 2D face recognition techniques, namely, Holistic Methods, which analyze the entire face as a whole. Techniques like Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) are commonly used to reduce dimensionality and highlight features that distinguish faces. Geometric Feature-Based Methods focus on the geometrical relationships of facial features, such as the distances and angles between eyes, nose, and mouth, to recognize faces. Along with Local-Texture Based Methods which, including Local Binary Patterns (LBP) and Gabor filters, concentrate on the texture patterns of small facial regions, capturing detailed facial features like wrinkles and skin texture (Insaf et al. (2020)).

As highlighted earlier, although FRS employing machine learning and deep artificial neural networks achieve remarkable accuracy, they concurrently raise significant privacy concerns. These issues are further compounded by the risk of data breaches, which pose a substantial threat to individuals' personal information and privacy. A viable strategy to mitigate these concerns involves the integration of obfuscation techniques. This approach aims to conceal individuals' identities, rendering them anonymous, while preserving the distinctiveness of facial features. Such a method ensures that the data remains valuable for the training and testing of FRS machine learning models without compromising personal privacy. This balance between utility and privacy represents a critical step forward in the ethical deployment of facial recognition technologies.

1.2 Project Motivation

The motivation for this project stems from the desire to support research focused on analyzing children's physical development through observations of their engagement in a variety of physical activities, including jumping, running, galloping, and skipping. These recordings are crucial for identifying any delays in physical development, which is intrinsically connected to cognitive development in children (Hillman et al. (2019)). One concern for these recordings, is in regard to privacy, given the ease with which the identities of the children in the footage can be detected. This vulnerability poses a significant risk to their privacy, especially in the event of a data breach or if the footage is utilized in any future research. Therefore, it is essential to process this sensitive data to obscure the children's faces, making sure their identities are hidden. It is still, however, crucial to retain the recognizability of facial features in the obfuscated footage to ensure its utility for future research. The balance between privacy protection and the preservation of useful data underscores the project's complex requirements.

A significant challenge faced by this project is the period when the data was collected, which was during and shortly after the COVID-19 pandemic. This means the dataset comprises of individuals both wearing face masks, in compliance with the pandemic mandates, and those without, particularly children who were deemed less susceptible to infection and transmission (Ludvigsson (2020)). This variance in the dataset presents a unique obstacle in detecting faces, therefore, great consideration is needed in the design of the system to accommodate for the variance in the dataset.

1.3 Project Aims

The main objective of this project is to generate an application that can analyze a set of images/ videos, focusing on detecting and obfuscating the faces present in the data to a degree that preserves facial features so that they can still be recognizable for potential future research. The specific goals of this dissertation are as follows:

1. Explore face detection techniques and algorithms.
2. Explore obfuscation techniques.
3. Explore computer vision libraries.
4. Explore GUI tools for a better user experience and configurability.
5. Evaluate results of facial detection.
6. Evaluate results of facial obfuscation, primarily retention of recognizability of facial features.
7. Verify performance on video footage.

1.4 Dissertation Structure

Chapter 1 serves as the introductory section of the paper, exploring the reasoning behind the development of the facial obfuscation application. Furthermore, this chapter offers a preliminary overview of face detection and face recognition systems, laying the groundwork for a more comprehensive exploration of these technologies in later chapters.

Chapter 2 delves into the existing body of research relevant to this dissertation, offering a thorough analysis of studies with objectives that align closely with this paper.

Chapter 3 looks into the various design decisions that were explored, ranging from the choice of programming language to use, computer vision libraries to implement, and the Graphical User Interface (GUI) to build the application on.

Chapter 4 covers the final implementation of the facial obfuscation application. Explaining the reasoning for choosing the GUI library to build the application on and the computer vision frameworks to use for the image processing; face detection and obfuscation.

Chapter 5 evaluates the performance of our facial obfuscation application through various experiments, covering what evaluations were used and why along with dissection of the results.

Chapter 6 is the final chapter, which overviews the paper along with the resulting facial obfuscation application. This chapter also covers the speculated future work for this research.

Chapter 2

Literature Review

The facial obfuscation application is split into two main components: first, face detection, which identifies the presence of faces within images and second, face obfuscation, which focuses on obscuring faces while preserving the identifiability of facial features with minimal loss. This section will delve into existing literature and research that align closely with the main objectives and components of this paper.

2.1 History of Facial Detection & Recognition

Facial recognition technology has evolved significantly over several decades. The foundational work by Bledsoe (1964) marked a pivotal moment, introducing a semi-automatic method for facial recognition. This early system required operators to manually input facial dimensions, such as the size of the nose and mouth, laying the groundwork for the sophisticated automated systems that are in use today.

Once artificial intelligence began to come into various fields, the field of facial recognition was also among those around the year of 1988. Turk and Pentland (1991) of the Massachusetts Institute of Technology (MIT) in 1991 made a groundbreaking development by being the pioneers of the first effective FRS by utilizing Principal Component Analysis (PCA). This was a method that significantly enhanced the system's ability to analyze and recognize facial features which will be covered in more detail later in the chapter. This milestone represented a major leap in the field of facial detection and recognition and set the stage for future innovations in the field.

There were various other key moments in history when the evolution of facial detection and recognition technologies saw significant acceleration. One such moment was in 1998 when

the Defense Advanced Research Projects Agency (DARPA) launched the Face Recognition Technology Program (FERET), which gave a significant boost to research in the field by providing an open-source database with 2,400 images of 850 individuals (Phillips et al. (1998)). A further acceleration in the field happened in 2005 when the Face Recognition Grand Challenge (FRGC) aimed at encouraging the development and advancement of facial recognition technologies (Phillips et al. (2005)).

In 2011 deep learning was another major advancement in many fields, facial detection and recognition included. Deep learning, a machine learning methodology which utilizes artificial neural networks (Guo and Zhang (2019)), marked a major point in history as this innovation caused a significant leap in performance, which scaled by the number of training images used, thus causing massive improvements in the accuracy and reliability of facial detection and recognition systems.

2.2 Facial Detection Techniques

While facial detection and facial recognition research went hand-in-hand for a combined goal of progressing FRS, given the focus of this paper on the facial detection component of FRS, I will not delve into the facial recognition component from this point on, so the remainder of the paper will concentrate mainly on exploring facial detection technologies.

Facial detection is the process of distinguishing human faces in a digital image or video without identifying them, unlike facial recognition. The main types of facial detection include feature-based detection, which extracts and matches facial features from images and image-based detection, which focuses on matching between training and testing images to detect faces in an image (Kumar et al. (2019)).

2.2.1 Feature-Based Facial Detection Methods

Active Shape Model

The Active Shape Model (ASM) is a computer vision technique aimed at detecting and modelling objects by analyzing their shape and appearance. Within the facial detection context, ASM identifies facial landmarks such as the outlines of the eyes, nose, and mouth by adjusting a predefined shape model to these points. ASM enables precise detection and analysis of facial features through iterative refinement of the model's parameters to better fit the observed facial structure (Cootes et al. (2000)). ASM proves to be especially beneficial for real-time facial feature tracking.

Snakes were originated by Kass, Witkin, and Terzopoulos (Kass et al. (1988)) and are pivotal for image segmentation and feature extraction. Snakes function as energy-minimizing splines that adapt to image contours by balancing internal forces (maintaining smoothness) against external forces drawn to image features like lines and edges. For facial detection with ASM, Snakes refine the model's fit to facial feature boundaries by iteratively adjusting to minimize energy, thus enhancing the precision of facial feature detection and tracking by conforming to their natural contours.

$$Snake = Internal + External \tag{2.1}$$

The Point Distribution Model (PDM) is integral to the ASM framework (Farfade et al. (2015)).PDM maps and object's shape, for example, a face, through specific landmarks. PDM can understand the diversity of shape changes by examining shape variations from a dataset of labelled images. This process allows PDM to construct a dynamic model that accurately reflects the potential alterations in an object's shape based on empirical data.

PDM restricts model deformation to realistic shapes derived from the training data. This precision ensures that the model's adaptations to new images yield authentic shapes. The PDM equips the ASM with the flexibility to accurately model different object instances by adhering to real-world observed deformations through incorporating mean shape alongside variations identified vis PCA (Turk and Pentland (1991)).

Deformable Templates represent an advancement over traditional Snakes (Yuille et al. (1992)). These models are inherently adaptable, designed to contour precisely to objects within images, such as facial features, by modifying a set of parameters. This method initiates with a baseline template shape, which is then mathematically adjusted to individual instances. The iterative refinement of template parameters aims to minimize the discrepancy between the model and the target object while ensuring accurate adaptation to variations in shape. Deformable Templates enhance ASMs' capacity for precise object location and tracking across diverse conditions, making them particularly effective for facial detection and recognition tasks.

2.2.2 Colour Models

Colour-based models are quite different to feature-based facial detection methods as the colour-based models utilize the unique colour properties of human skin to pinpoint faces

in an image or video. Human skin tones generally occupy a specific range of colour values in a certain colour space, meaning that these models aren't limited by diverse ethnicities and individual differences. This approach allows for effective segmentation of skin regions from backgrounds and other non-skin elements, thus enhancing the efficiency of the facial detection process (Phung et al. (2005)).

Colour-based detection methods are most effective in controlled environments but may struggle based on various factors such as the camera quality, the presence of objects that resemble skin tones in the image or frame, and some models struggle with light variations. To improve their accuracy, these methods are often combined with other detection techniques. I will cover the four basic colour models, namely RGB, HSV, YCbCr, and CIELAB.

RGB colour model comprises three primary colours—Red (R), Green (G), and Blue (B)—as well as three secondary colours: Magenta, Yellow, and Cyan (Fig 2.1). While this model serves as the foundation for all others, it has a limitation: it cannot isolate the colour (Chroma) and intensity of a pixel, making it challenging to detect skin-coloured regions (Chandrappa et al. (2011)). Therefore it is sensitive to light variations.

How RGB face-detection works is a normalized histogram is used to detect pixels of skin colour within an image and may be further normalized for changes in intensity on dividing luminance (Ismail and Sabri (2009)). Due to these limitations, RGB is mostly used for image processing and storing digital images rather than facial detection (Fig 2.2), thanks to chrominance and luminance being mixed in with RGB.

HSV model consists of three attributes, Hue (H), Saturation (S), and Intensity Value (V). Hue represents the colour, consisting of red, blue, and yellow, with values ranging from 0 to 360. The Saturation ranges from 0% to 100% and is responsible for the purity of the colour. Value, on the other hand, refers to the brightness of the colour. From the HSV model, Hue and Saturation are the primary focus for facial detection in regards to skin colour.

$$0 <= H <= 0.25 \tag{2.2}$$

$$0.15 <= S <= 0.9 \tag{2.3}$$

Looking at facial detection using the HSV model (Fig 2.3), we see a drastic improvement over the results using the RGB model meaning that when colour is involved, HSV seems

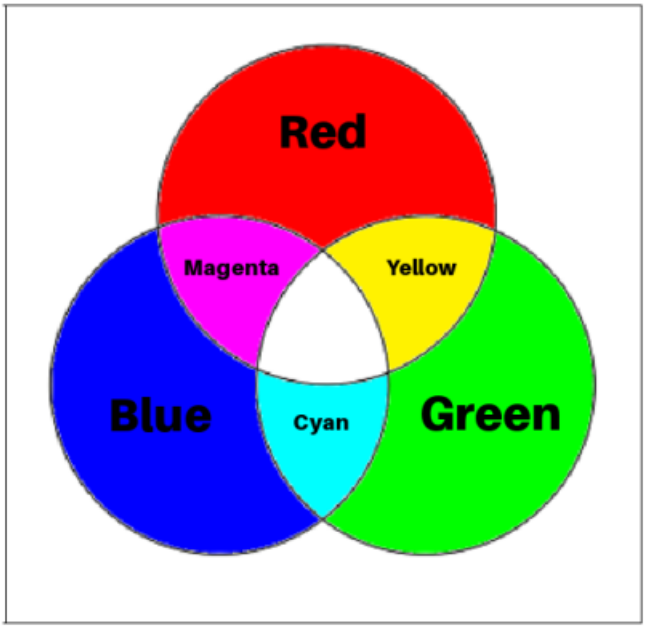
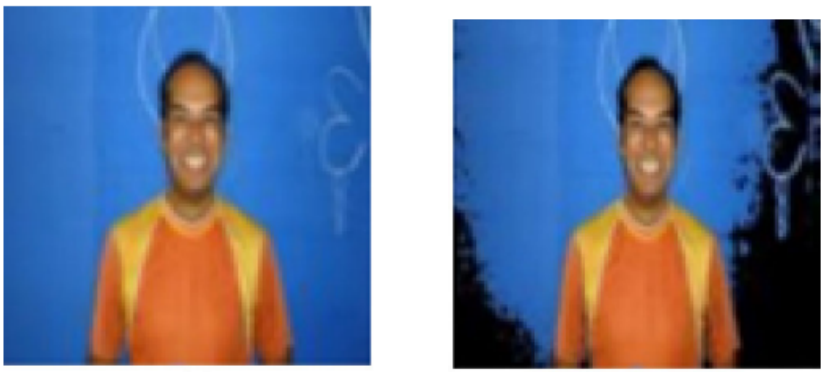


Figure 2.1: RGB Colour Model

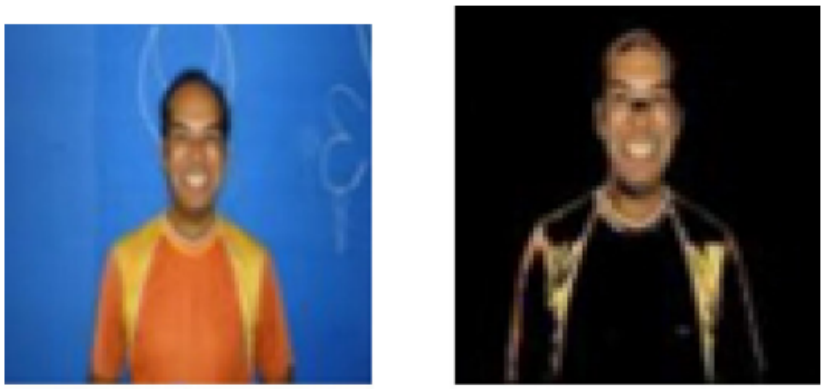
to be the preferred method of facial detection over RGB. However, when the Saturation is low, Hue may not be reliable (Rewar and Lenka (2013)).

YCbCr colour model is defined by two channels, Luminance (Y) and Chrominance (Cb and Cr). This model segments the image into the Luminance component and Chrominance components (Phung et al. (2005)). Since the Luminance component is almost entirely independent from the Chrominance components there is a non-linear relationship between the luminance and chrominance of the skin colour in regions where there is a low or high luminance. The range of the luminance component lies between 16-235 with the lower



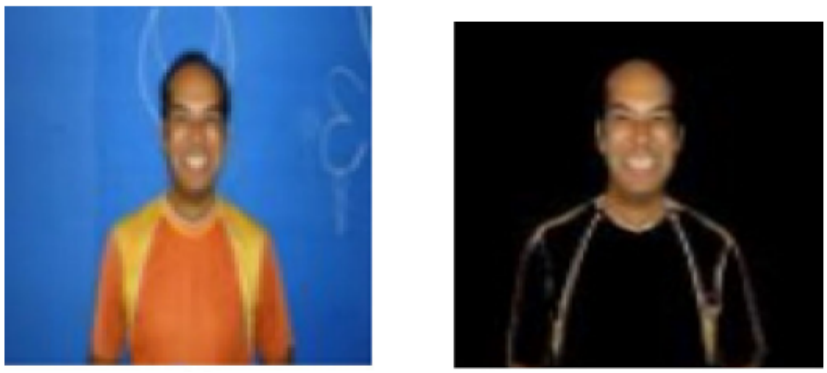
(a) Original Image (b) RGB Image

Figure 2.2: Colour-based facial recognition model (RGB) (Rewar and Lenka (2013))



(a) Original Image (b) HSV Image

Figure 2.3: Colour-based facial recognition model (HSV) (Rewar and Lenka (2013))



(a) Original Image (b) YCbCr Image

Figure 2.4: Colour-based facial recognition model (YCbCr) (Rewar and Lenka (2013))

limit being black and the upper limit representing white, while the Chrominance is set to the range of 16-240. Various plots for Y, Cb, and Cr for face and non-face pixels were generated to find the range of Y, Cb, and Cr values for face pixels.

This causes the YCbCr model to have incredible results in facial detection in comparison to previously mentioned models which can be seen in Fig 2.4. This is thanks to the fact that the distribution of skin regions is consistent across different races in the Chrominance channel (Padhee (2012)) and the fact that the influence of luminosity can be removed during the processing of an image.

CIELAB colour model was developed in 1976, when the CIE (International Commission on Illumination) recommended the CIEL*a*b* colour scale. It provides a standard, approximately uniform colour scale which could be used by everyone so that the colour values can be easily compared. It is related to the RGB colour space through a highly

nonlinear transformation. Fig 2.5 shows a visual representation of the channel, from which we can see that L represents the lightness, with $+a$ and $-a$ indicating the level of red and green respectively. The value of $+b$ represents yellow and $-b$ represents blue. L ranges from 0, which represents the colour black, and 100, which represents the perfect reflecting diffuser (white colour), while on the other hand, the a and b values do not have a specific numeric value.

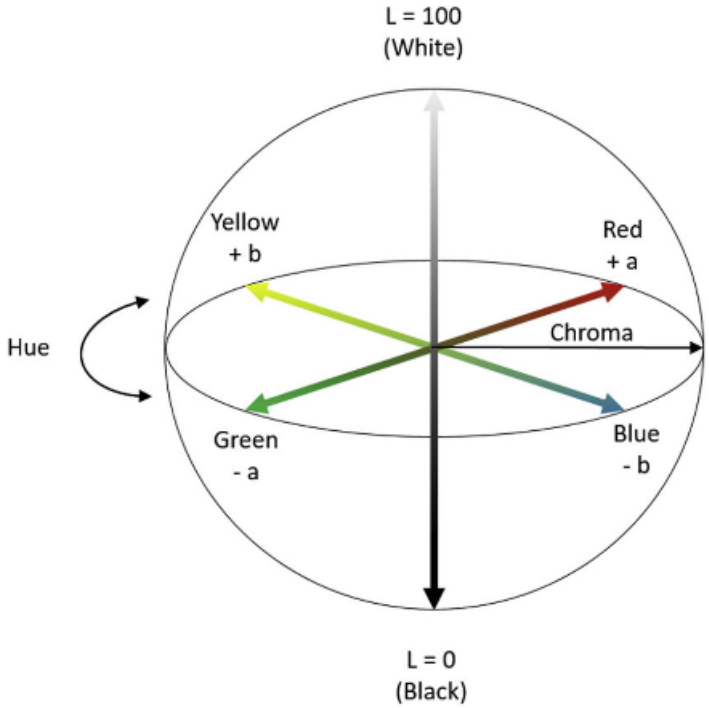


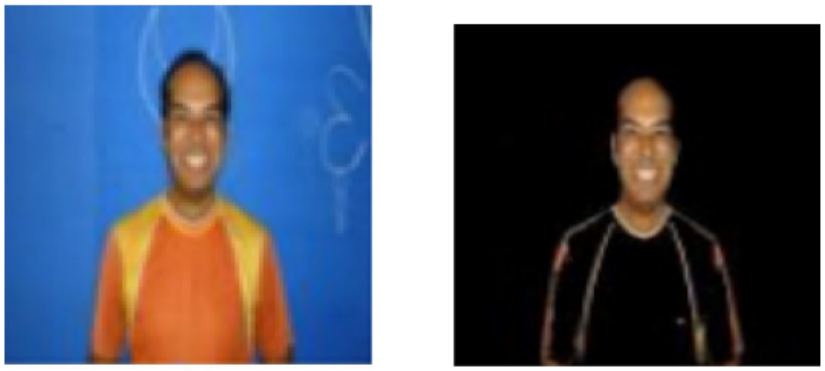
Figure 2.5: CIELAB Colour Model (Ly et al. (2020))

Fig 2.6 shows that CIELAB has a similar performance to YCbCr, where it outperforms both RGB and HSV colour models.

Comparing & Analysing the colour models utilized within feature-based facial detection methods, the YCbCr model yields the best detection rate and a lower percentage of false positives, thereby outperforming alternative color-based models (Rewar and Lenka (2013)).

2.2.3 Image-Based Facial Detection Methods

Image-based facial detection methods refer to a set of techniques used to locate faces within an image or video. These methods are usually more advanced than the previously discussed approaches to facial detection since most methods that utilize image-based facial detection methods tend to involve machine learning and neural networks to a certain



(a) Original Image (b) CIELAB Image

Figure 2.6: Colour-based facial recognition model (CIELAB) (Rewar and Lenka (2013))

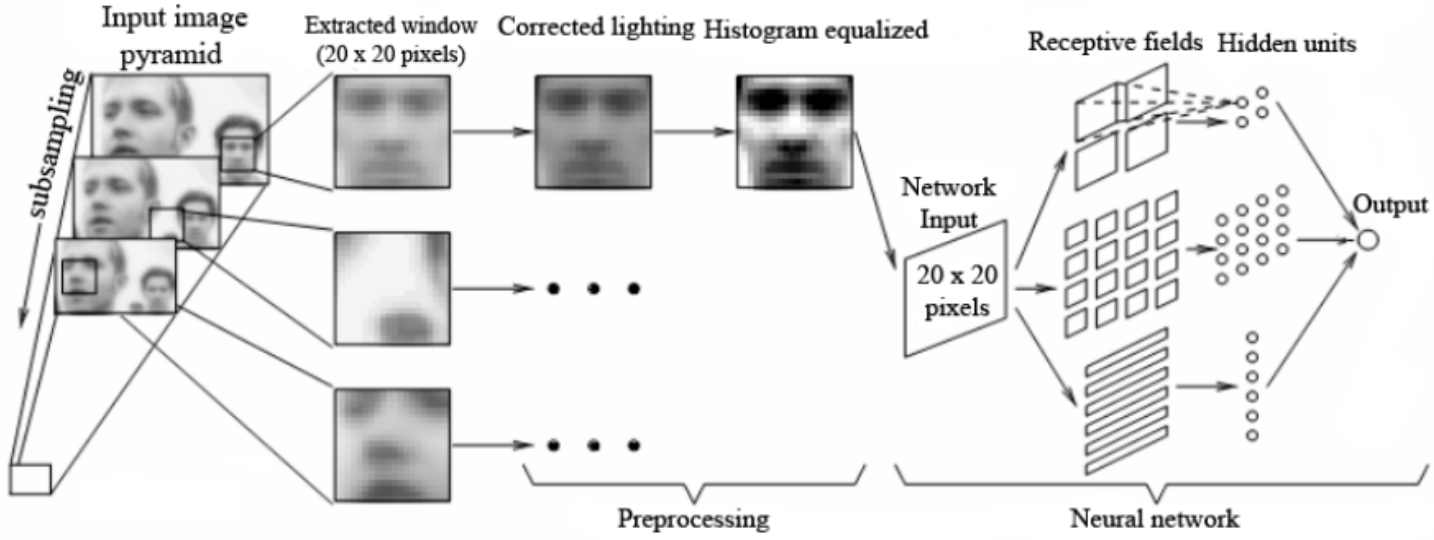


Figure 2.7: Basic Face Detection Algorithm (Rowley et al. (1998))

degree. This typically results in these methods yielding higher accuracy and lower susceptibility to variations such as face orientation, expression, and lighting conditions.

Neural networks are vital when addressing various computational challenges, ranging from computer vision tasks to natural language processing. Their effectiveness and versatility come from their design, which mirrors the brain's neural networks. These networks are comprised of multiple layers connected by weighted edges which are adjusted during the training process based on the discrepancy between the predicted and the actual outcomes. This allows the network to learn from the data, enhancing its accuracy and making it ideal for solving complex problems across various domains (Dongare et al. (2012)). An overview of how a basic face detection algorithm works can be seen in Fig 2.7.

Support Vector Machine (SVM) is a statistical face detection method using a two-class classifier system. It leverages support vectors, which are pivotal samples closest to the decision boundary, to create an optimal hyperplane. This hyperplane effectively separates face from non-face images by transforming the image data into a higher dimensional space, where it becomes easier to classify. The process includes training the SVM with images labeled as faces or non-faces, optimizing for accuracy while minimizing classification errors through a nonlinear optimization approach (Shavers et al. (2006)).

Principal Component Analysis (PCA) is another example of a statistical face detection method. As Faruqe and Hasan (2009) explained, PCA is a procedure where facial images are converted into a set of orthogonal vectors or 'eigenfaces'. These vectors or eigenfaces represent the most significant variations in the face data, which effectively reduces the dimensionality of the dataset while retaining the features that are most relevant for detecting faces. This allows for efficient facial detection by projecting new images into this reduced vector space and comparing them against the existing threshold to determine if a face or faces are present. This method ignores irrelevant variations and focuses on the most important facial features to enhance its detection accuracy.

Haar Cascade Classifier , developed by Viola and Jones, utilizes a machine-learning approach for its face detection algorithm. The Haar Cascade Classifier employs Haar-like features to contrast features between regions of an image to capture the presence of face structures. These features are used in parallel with a cascade of classifiers, where each stage continuously filters out non-facial regions, such as no nose, no eyes, etc., which significantly reduces computation for each following classifier. With the constant enhanced computation, this method allows for quick detection of faces by focusing computational resources only on the promising face regions which allows for better real-time processing of images. The classifier is trained using a large set of positive and negative images, where the positive set contains faces, and the negative does not. The result is a classifier that can identify faces with high accuracy and a low number of false positives, making the Haar Cascade Classifier a widely used technique in face detection applications (Cuimei et al. (2017)).

Histograms of Oriented Gradients (HOG) technique relies on evaluating well-normalized local histograms of image gradient orientations in a dense grid, capturing edge or gradient structure indicative of local shape. This method is highly robust as it works well on varying factors such as orientation and lighting (Navneet (2005)).

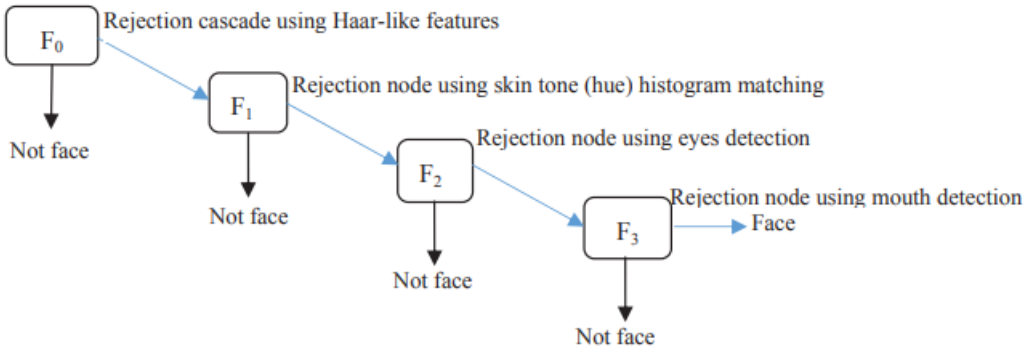


Figure 2.8: Haar Cascade (Cuimei et al. (2017))

Region-based Convolutional Neural Network (R-CNN) along with their successors such as Faster R-CNN, that have made a drastic leap in the field of facial detection and general object detection in images and videos. R-CNNs have various steps to achieve the amazing outcome. The first is feature extraction, this step involves feature extraction where the image/ frame of a video is passed through a pre-trained CNN to extract features. The output of this step is a feature map that represents the important aspects of the image while reducing its dimensionality.

Step two varies on which implementation of R-CNN is used.

- **Basic R-CNN** uses selective search to propose possible regions that contain our object of interest (in our case a face). Once these regions are found, they are individually sent to a CNN for further evaluation.
- **Faster R-CNN** introduces the concept of Region Proposal Network (RPN), which speeds up the process by predicting object bounds objectness score directly from the feature maps. This removes the requirement for selective search and produces high-quality region proposals.

  An extra step is involved in Faster R-CNN that is absent from the basic R-CNN implementation, this is ROI Pooling. This step involves resizing the proposed regions to a fixed size, this is required since the subsequent layers of a fully connected network require inputs of the same size.

The penultimate step involves the classification of the proposed regions, this classifies the regions into various categories such as contains face, doesn't contain face, background,

etc. The regions that are classified as containing an object such as a face are then adjusted to better fit the object.

The final step involves a non-maxima impression. Since multiple bounding boxes may be present around the same face, we use the non-maxima impression to discard all of the bounding boxes except the one with the highest confidence score. After all these steps, the output is a set of bounding boxes along with a confidence score which indicates the likelihood of the bounding box containing an object (in our case a face) (Jiang and Learned-Miller (2017); Sun et al. (2018)). A visualization of the Faster R-CNN can be observed in (Fig 2.9)

Following the completion of these steps, the result is an assortment of bounding boxes, each accompanied by a confidence score. This score quantifies the probability that a given bounding box encases an object, specifically a face in this context.

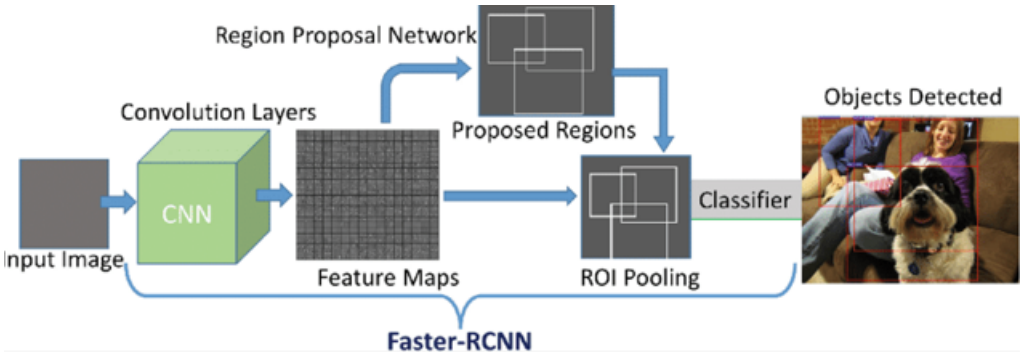


Figure 2.9: Faster R-CNN Pipeline (Saikia et al. (2017))

Deep Learning is a significant leap in the field of machine learning and artificial intelligence. Deep learning excels in interpreting and analyzing complex data structures by utilizing layered neural networks to extract and elevate features from unprocessed data. Each layer in the deep learning architecture refines the data into more and more abstract representations, which allows the system to recognize complex patterns and relationships (LeCun et al. (2015)).

Deep learning outperforms traditional computer vision techniques in a multitude of tasks such as image classification, object and face detection, and segmentation, thanks to its ability to automatically learn from data which eliminates manual feature selection. While deep learning demonstrates amazing accuracy, it has one major downside, which is that it is very computationally expensive, making it much slower than most traditional computer

vision approaches (O'Mahony et al. (2020)).

Comparing & Analysing the image-based approaches for facial detection, referencing the results observed from Dang and Sharma (2017), we see that when comparing SVM, Viola-Jones (Haar Cascade Classifier), and basic Neural Network-Based Face Detection, Haar Cascade Classifier seems to outperform the others for facial detection methods.

While the Haar Cascade Classifier shows to outperform simple CNN for facial detection tasks, in my assumption, more complex frameworks such as R-CNN and advanced deep learning models surpass the Haar Cascade Classifier due to their complex architectures.

2.3 Obfuscation Techniques

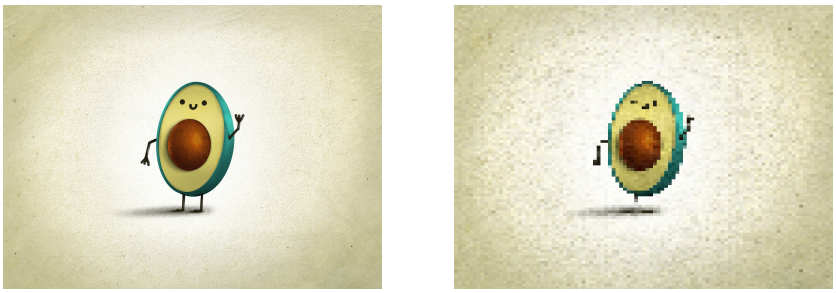
Obfuscation in computer vision is the process of modifying an image or video to hide or obscure the entire image/ video or only a specific region. In our case, it is the process of obscuring an individual's face to hide their identity for privacy purposes. Many methods exist to achieve this goal, each with its strengths and weaknesses which will be explored below.

2.3.1 Masking

Masking is the process of applying a uniform colour or pattern over a designated area within an image or a video, fully obscuring that area. This approach is straightforward and extremely effective, its only downside is that it is extremely visually intrusive and eliminates any possibility of face detection as it hides the entire face along with any facial features.

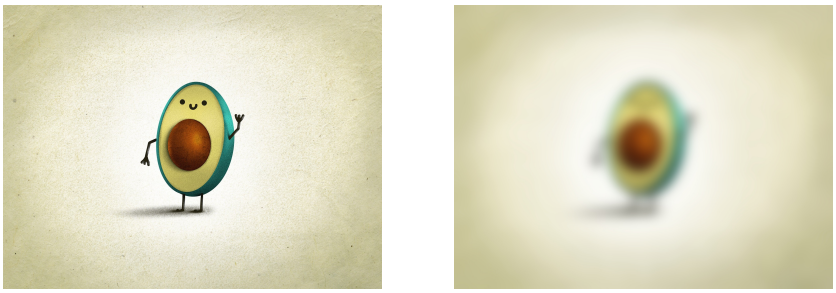
2.3.2 Pixelation

This method significantly reduces the quality of an image or image area, which can be applied once the coordinates of the face are found in an image or video frame. If the pixels in the image are merged, the facial features become indiscernible. While this is a fast and simple method, it has its drawbacks in the fact that the pixilated area can be reversed to reveal the original image with advanced de-pixelation algorithms. An example of pixelation can be seen in Fig 2.10



(a) Original Image (b) Pixilated Image

Figure 2.10: Obfuscation Methods (Pixilating)



(a) Original Image (b) Blurred Image

Figure 2.11: Obfuscation Methods (Blurring)

2.3.3 Blurring

Blurring an image or an image section involves algorithms such as Gaussian Blur and convolving the image with a low-pass image kernel which smooths out the image making the face unrecognizable while retaining the facial features. This method is specifically good for non-invasive facial detection as it retains the anonymity of the individual while allowing the system to detect the person's face. This method is more effective than pixelation in preventing facial recognition, but can still be partially reversed with advanced image processing techniques (Hummel et al. (1987)). An example of blurring can be observed in Fig 2.11

2.3.4 Face Replacement

his sophisticated method involves swapping each individual's face in the image or video frame with a random template face, with more advanced machine-learning approaches such as Generative Adversarial Networks (GANs) (Goodfellow et al. (2020)) and autoencoders. These advanced machine learning approaches generate a face of a non-existing person and apply it to the detected faces in the image, making it seem like no alterations were done to the image while hiding the individual's identity. The only drawback of these

methods is that they are more computationally intense (Croft et al. (2022)).

2.4 Related Work

The need for privacy in video has been a long-standing issue, one that has come to light alongside the development of facial detection and recognition. This means that many people have created applications to obfuscate individual's faces for privacy concerns.

2.4.1 tnouvel's face-obfuscation Application

The face-obfuscation application that was developed by the GitHub user tnouvel was designed with a similar goal in mind, to protect people's privacy. This application utilizes OpenCV's DNN module which allows the user to load different models for deep learning. This application also features a GUI to enable the user to upload either a video or image, along with specifying some settings such as obfuscation type and level (Tnouvel (2022)). However, this application fails to test detection on people wearing face masks along with the overall performance of their application.

2.4.2 davebaraka's face-obfuscator Application

This application is a Chrome browser extension, targeted towards dynamically and locally blocking pre-trained faces from images loaded in Chrome (Davebaraka (2020)). This application is powered by face-api, a JavaScript recognition API (Justadudewhohacks (2018)). While this application has the necessary aspects that we require for the facial obfuscator application, it has a different target task meaning it's missing many of the features that we require for the facial obfuscation application.

2.4.3 Face blur

Face blur is an AI-powered solution for privacy protection, boasting a fast, accurate, and adaptable application for privacy in data processing. This application processes data for various industries that are required to distribute their footage to third parties and protects the identity of the individuals present in the data (Streamingo.ai (2023)). This application however does not provide us with a way to fine-tune the obfuscation level and type.

2.5 Summary

When developing a system that involves computer vision approaches, it is clear that we have to take into consideration that there has been extensive research done that derives many methods for different tasks such as facial detection and obfuscation. The important consideration that needs to be done when designing a new system is the target of the proposed system and which of these methods will suit to tackle the system's requirements the best as all of these methods have both advantages and disadvantages associated with them. While some facial detection detection algorithms can apply accuracy, they also will be more complex and time to implement and train, while some facial obfuscation techniques can be implemented fast and easily, they lack protection as they can be reversed to identify the original image.

Chapter 3

Design

Before implementing the face obfuscation application, a multitude of design decisions and planning needs to be done. Primarily, what programming language would be best suited as this will determine the computer vision and GUI libraries that we can utilize? In this chapter, I will cover the options available to us, comparing various programming languages, computer vision libraries, and GUIs.

3.1 Requirements

The requirement for this face obfuscation application is a highly adaptable application that will be able to detect faces in videography in any condition, such as any angle of the camera, any amount of faces present in the frames, with and without face-masks, any lighting, and while in motion. After the facial detection, the application will need to obfuscate the located face at every frame, while keeping the facial features recognizable and the face can be detected again for future research.

Another requirement is to provide an easy-to-use GUI that will allow the user to select a video to apply face obfuscation. This should be easily runnable and needs to support various video formats.

3.1.1 Challenges

While designing the face obfuscation application, various challenges arose due to the targeted data that was gathered in uncontrolled conditions. This means that the face detection algorithm or library used will need to be able to tackle lower camera quality, multiple faces present in the frames, and the camera being at varying locations with varying heights, angles, and backgrounds. The application will also need to be able to

detect faces that are wearing face masks, this is because the data was gathered after the COVID-19 pandemic, which means some of the kids wore masks for safety while others didn't.

3.2 Design Choices

3.2.1 Programming Language

The choice of programming language is important as this will determine the set of computer vision libraries that will be chosen to employ facial detection, along with the obfuscation of the found faces, which can be done by the same library or a different one. Similarly, the language will determine what GUI libraries can be used which is important as this application might be used by non-technical people, so an easy-to-use GUI which can support many features.

I decided to look at OpenCV, which is the most well-known computer vision library, and which languages it supports. If a programming language is supported by OpenCV, it is most likely quite a suitable language for computer vision-related tasks, and will therefore have other libraries that we can explore. The three main languages that are supported are Python (van Rossum and Drake (2006)), C++ (Stroustrup (2013)), and Java (Flanagan (2005)).

Python is the most versatile language that most software developers are well-versed in. Due to the ease and versatility of this language, many libraries have been written for Python (Viduka et al. (2021)), including various computer vision libraries that employ a majority of face detection methods, and obfuscation techniques that I covered in the Literature Review Chapter and much more. Along with this, developers use Python for graphical applications due to Python being able to do many tasks with ease like editing files and various network-based tasks. Python however has one drawback, it's quite difficult to design large-scale projects, this is due to the language requiring an interpreter instead of a compiler, whereby an interpreter compiles code line by line which becomes an issue for larger projects (Sharma (2022)).

C++ is an extension from C (Kernighan and Ritchie (2002)), a systems language, which takes all the low-level programming capabilities and advantages and combines them with high-level programming by introducing object-oriented programming. The characteristics that set C++ apart from other languages can be attributed to its design philosophy, which

emphasizes programmer control over hardware efficiency while also providing high-level abstractions (Jordan (1990)). C++ is the preferred language choice when developing large projects that take advantage of object-oriented programming but require high performance which C++ can provide by giving the developer full control over the hardware's memory.

Java is another object-oriented, high-level programming language. Java is popular amongst developers for many reasons; primarily its robustness and security along with having many useful features such as automatic garbage collection. One of the main attractions of Java is Java Virtual Machine (JVM), which is available on many types of hardware and operating systems, this allows any Java application to run on any device and operating system without modification, facilitating software distribution. Unfortunately, Java is without its downsides, mainly being larger overheads in memory, primarily due to the abstractions introduced by the JVM, and performance due to features like automatic garbage collection and object-orientated programming (Payne (2024)). Another downside of Java is building GUI applications tends to pose a problem in the performance of the application.

Comparing & Analysing the three programming languages, we see that Python is the best all-around choice, as it offers a multitude of libraries for both computer vision and GUI tasks while also being runnable on many different types of operating systems and hardware. While it does have a downside when creating large-scale projects, this won't pose an issue for us as our project won't be too large.

The reason for not choosing C++ is because of the issue with dependencies after compilation, as we want a tool that is versatile, flexible, and can work on many different machines. Similarly, Java was not chosen as it has its issues with memory and performance issues, which could potentially make our GUI application slow and frustrating to use.

3.2.2 Computer Vision Library

Now that we have chosen our desired programming language, we need to explore the available libraries that will aid us in the computer vision tasks, which is the main aspect of the face obfuscation application, such as facial detection and obfuscation. From looking at the face detection techniques covered in the Literature Review Chapter, it is clear that machine-learning-based algorithms are superior, so I decided to implement face detection through libraries that have pre-trained models, as they will be more accurate than models

that I would train myself, which means choosing the correct library is crucial to the outcome of our facial obfuscation application. We would also like to test the performance of some libraries that utilize basic methods of facial detection, to compare with the more advanced and speculatively better approaches. After a thorough review of various sources, I comprised a list of the most well-known and recommended libraries for facial detection and computer vision (CM (2023); AI (2023)).

Open Source Computer Vision Library (OpenCV) is the most well-known computer vision library that provides over 2500 optimized algorithms, with both classic and modern, state-of-the-art algorithms (OpenCV (2020)). This library is quite flexible and easy to use, being able to analyze both image and video and is usually used in contrast with other computer vision libraries due to its multitude of algorithms.

This library can do facial detection as it implements various facial detection algorithms such as Haar Cascade Classifier and Eigenfaces (Khan et al. (2019)) that were covered in the Literature Review Chapter. Similarly, OpenCV also employs blurring of either the entire image/ video or a provided region.

Dlib is a comprehensive library focused on offering a wide variety of machine-learning functionality, including facial detection (Dlib (2022)). It includes an advanced linear algebra toolkit with BLAS support, alongside algorithms for Bayesian networks, kernel-based methods, and more (King (2009)). The library offers a user-friendly design and is adaptable to a wide range of projects which is ideal for research or commercial applications projects.

Dlib only offers facial recognition for our facial obfuscation application, without any support for obfuscation techniques, which only covers half of the computer vision requirements of our application, meaning we will need to use this library in contrast with another that implements obfuscation techniques. For facial detection, Dlib allows its users to choose between two options; Histogram of Oriented Gradients (HOG) + Linear SVM and Max-Margin (MMOD) CNN. HOG + Linear SVM is accurate and computationally efficient, while Max-Margin (MMOD) CNN is extremely accurate and very robust (Rosebrock (2023)).

face_recognition is a facial detection and recognition library built upon Dlib's deep-learning model and boasts a score of 99.38% accuracy in the Labeled Faces in the Wild (of Massachusetts (2018)) benchmark. The library implements many additional features,

one such feature is quite interesting to our facial obfuscation application, facial feature extraction (Geitgey (2020)).

The benefit of this library is that it is built specifically for facial detection and recognition and has been designed to be easy and straightforward to use. It also has a high facial detection and recognition rate because it uses a deep-learning model, meaning that it will be highly accurate.

Multi-task Cascaded Convolutional Networks (MTCNN) library is a deep-learning framework designed for efficient and accurate face detection. Its structure comprises of a cascade of CNNs to handle tasks such as face detection, bounding box regression, and facial landmark localization (Zhang et al. (2016)).

The library boasts real-time performance, and robustness against variations in pose, illumination, and occlusion. However, its one major downside is that it is computationally intense along with being reliant on substantial training data which could pose a challenge in resource-constrained environments.

RetinaFace is a deep-learning face-detection library for Python. It boasts amazing performance even in massive crowds (Serengil (2021)).

RetinaFace utilizes a multi-task learning approach, that simultaneously predicts face scores, bounding boxes, facial landmarks, and a dense 3D face mesh. By leveraging extra-supervised signals from facial landmark annotations and self-supervised signals for 3D face shape prediction, this method significantly improves face detection. While this method outperforms other face recognition methods in both accuracy and speed, it still struggles in resource-constrained environments (Deng et al. (2019)).

Comparing & Analysing the libraries together, comparing the methods they each use for face detection, and looking back at those methods that were covered in the Literature Review Chapter, we see that the Dlib, MTCNN, face_recognition, and RetinaFace libraries would be best suited as they use CNNs and Deep Learning.

3.2.3 GUI Library

Choosing a Graphical User Interface (GUI) library was also quite important for the facial obfuscation application since we would like a front-end that could easily implement our

back-end facial obfuscation by allowing the user to upload video/ image. While we would like easy integration, we still want the GUI to allow for advanced features to give the user many settings to fine-tune the obfuscation to their liking. Following various online resources of recommended GUI libraries, I decided to look into a few of them to see the features they offer along with how easy they are to use (Kummarikuntla (2024); Nederkoorn (2021)).

Tkinter, Tkinter(Lundh (1999)) is the standard GUI framework for Python. It has many benefits such as ease of use and many features that are ideal for desktop application development (GfG (2023)). However, because the library was developed quite some time ago, it has quite an outdated format and feel to it.

CustomTkinter (TomSchimansky (2021)) tries to tackle the downsides of the original Tkinter library by introducing a more modern framework while retaining the ease of use and all the functionality that the original Tkinter has to offer Custom Tkinter also ensures that the applications that are built on it have the same look and feel on different platforms such as Windows, Mac, and Linux (Khan (2022)).

Kivy is a cross-platform Python app development application that introduces a fast and easy-to-use framework for development applications for Windows, Linux, macOS, iOS, and Android with a single codebase (Kivy (2010)).

PyQt or PySide are two components that implement Qt, a Python GUI framework. The main difference between the two is licensing rights, otherwise they both implement Qt and are interchangeable. Qt offers a high-level API that allows the user to access many useful aspects of a modern desktop application (Bank (2021)).

WxPython is another cross-platform Python GUI framework that allows for development for Windows, Mac, and Unix systems (wxWidgets (2012).

PyGTK (Henstridge (2003)) is a framework mainly used for developing GUI applications for Python that boasts performance as its main benefit over other applications. It does, however, have one downside, and that is the size of the code, where every component needs to be individually constructed, positioned, and then shown (DataScientist (2024)).

Comparing & Analysing the GUI libraries together we see that the older frameworks (Tkinter, PyGTK, PyQT, and wxPython) all have their own benefits and downsides,

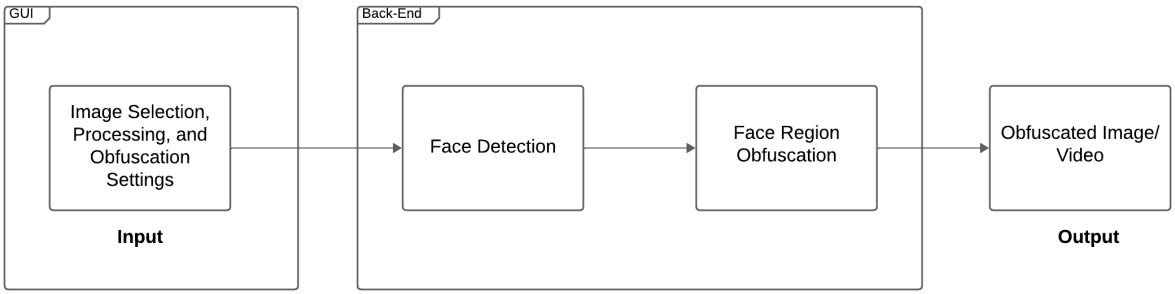


Figure 3.1: Facial Obfuscation Application Design

with PyGTK being more complex and Tkinter having a more limited set of widgets (Polo (2017)). Frameworks that present a more modern look and feel (CustomTkinter and Kivy) both offer a plethora of functionality and cross-platform integration, all while being easy to use.

3.3 Overview of the Design

The final proposed design, as visualized in Fig 3.1, consists of various steps, the first being the processing of the video/ images along with various obfuscation settings. The next and most important step is face detection using a face-detection computer vision framework. The final step is to obfuscate the detected faces with the specified settings.

3.4 Summary

Great consideration and evaluation are needed to design the facial obfuscation application as many tools can achieve the desired result. However, we would like to develop the tool with the best performance, accuracy, speed, and future integration in mind.

Choosing a suitable programming language will determine the subset of libraries and development frameworks that are available to us. Following this, the libraries we choose will impact the accuracy and speed of the obfuscation of our application. Finally, the GUI that we implement will determine the amount of functionality and settings that we can introduce to the user to allow them to be partially in charge of the output.

Chapter 4

Implementation

4.1 Overview of the Solution

Once the components of the facial obfuscation application were decided, which we see in Fig 3.1, it was time to implement the facial obfuscation application which I decided to build entirely using the Python programming language. I realised that if I wanted to make an application that would stand the test of time, I needed to create an adaptable application that could be easily upgraded and its components easily changed. Looking back at the Literature Review Chapter it is clear that the field of computer vision is constantly evolving, especially in the area of face detection. Along with this, if we recall all the libraries that were covered in the Design Chapter, we can see that libraries are constantly developed to build on existing ones or completely new ones which implement better methods or models. I decided to structure my code in such a way, that the face detection and obfuscation can be easily changed in the future in a relatively simple way to ensure that the facial obfuscation application is future-proof.

I decided to begin my implementation with the face detection and blurring components to build an API-like system that can be simply called from the GUI by simply passing the image/ video to it to do all the processing. I did this to create a versatile system that can be easily integrated into other systems.

For image and video processing I decided to use OpenCV as it has many useful features for image and video manipulation which I used as a basis to dissect videos and perform modifications on the video frames/ images.

The source code for the facial obfuscation application can be found in a GitHub repository

(Borsak (2024)).

4.2 Face Detection

For face detection, I began by building a Processor abstract class which acted as the base class for the Image Processor and Video Processor classes which processed images and videos respectively. The Processor class implements a single function which takes in an image which is read using OpenCV, and detects and returns face coordinates of faces that are present in the image.

The Image and Video Processor classes implement the Processor class and therefore can use the detect_faces function, with the only difference being, that the Image Processor passes the image it is trying to process as a parameter, whereas, the Video Processor first dissects the video into frames and passes the frame as the image into the detect_faces function one by one. This is where the flexibility of my implementation comes in, by simply altering this single function within the Processor class, we can change how the faces are detected, making my implementation highly versatile, capable of quickly and easily altering the face detection component if a better implementation is discovered. I decided to leave this function empty until I finished testing the various face-detection frameworks which were covered in the Design Chapter, with the results observed in the Evaluation Chapter.

4.3 Blurring

Recalling the obfuscation techniques covered in the Literature Review Chapter, blurring and pixelation seem to be the most suitable techniques for our requirements. These two obfuscation methods successfully obfuscate the face area to hide the individual's identity while having the ability to not obfuscate the face too much so that the facial features are still detectable based on user-specified obfuscation level.

Deciding on the computer vision library for facial obfuscation was quite a straightforward choice as most of the libraries that were covered in the Design Chapter, did not contain the required obfuscation that was required for the facial obfuscation application. However, OpenCV had such functionality of both blurring and pixelation and since we were already employing parts of OpenCV's functionality for image processing, I decided to use it for the application's obfuscation needs.

Following the aforementioned architecture of having an API-like structure for the facial obfuscation component of the application, and building on the existing Processor class, I decided to incorporate the obfuscation functionality into the Processor class. I began by passing new parameters for obfuscation to be stored by the class, namely obfuscation level, which determines the extent to which the user wants obfuscation to be applied, and obfuscation type. I decided to implement both pixelation and blurring and allow the user to choose between the two for additional functionality. Along with this, I also decided to give the user the option to obfuscate the entire image rather than just the faces.

The classes that implement the Processor Class such as the Image and Video Processor Classes simply have to call a function to obfuscate faces after which they are returned with the processed image. The code for this function can be observed in Listing 4.1. This function implements the aforementioned function to detect faces followed by a function to obfuscate all detected face regions since an image may contain multiple faces. This function simply goes over every region that was passed to it and calls another function to perform obfuscation on that specific region and return the image, the image is then set as the current image to ensure that each face is obfuscated. The obfuscation function for a single face region checks whether the type of obfuscation is blurring or pixelation, and performs the obfuscation accordingly.

```
def obfuscate_faces(self, image: cv2.typing.MatLike) -> cv2.
   typing.MatLike:
    faces = self.detect_faces(image)
    image = self.obfuscate_regions(image, faces)

    return image
```

Listing 4.1: Obfuscate Faces Function

Similarly to the obfuscate faces function, if the user wants to obfuscate the entire image, he can call an obfuscate everything function which performs similarly to the obfuscate region function only this time applying the blurring or pixelation to the entire image.

4.4 GUI Implementation

Analyzing the various GUI frameworks available in the Design Chapter, I decided to try using PySide as it had a wide variety of functionality and supported style sheets, meaning that its initial outdated look could be improved. Before I began implementing the GUI however, I thought it would be a good idea to draw up a mock-up of how I wanted it to look to have a rough path to take during development. Once this was finished, I began implementing the GUI by creating a button at the top of the GUI window which allows the user to upload the images/ videos that they want to obfuscate. Following this, I decided to add a big section with various settings that the user can alter to customize the obfuscation to their liking, these included a drop-down with all available obfuscation types, which was blurring or pixelation. I did this for the ability to implement additional obfuscation techniques in the future if there was a need. Along with this, I also added a check which determined whether the facial obfuscator was to obfuscate the entire image or leave it to obfuscate specifically the faces. Finally, since I already implemented an obfuscation level variable into the aforementioned Processor class, I decided to give the user the option to select this themselves through a slider. Finally, I added a button to begin the process of obfuscation which would call the back-end obfuscation API. At this stage the application looked quite outdated meaning it was time for me to apply style sheets to modernize it, however, once I began delving deeper into how to implement this, I realised that the style sheets were quite poor and did basic visual changes such as changing the colour. Due to this along with struggling to get some desired functionality working, I decided to try porting my design to another GUI Library.

Once again recalling the GUI frameworks that were covered in the Design Chapter, I thought about using Tkinter from the beginning as it was the most versatile and well-known GUI library in Python, however, it would have the same issue as PySide, which is an outdated look. CustomTkinter on the other hand fixed this issue while still implementing most of the functionality of the original framework, so I decided to attempt to implement my desired GUI utilizing CustomTkinter which wasn't too difficult as I already had experience in writing the UI using PySide and had a good idea of how I wanted it to be structured. I decided to structure in a similar way as previously mentioned, with a large button to upload files to run the obfuscation on, only this time I also added a list of selected files so the user can keep track of the files that they have added. I also slightly altered the settings layout lightly, keeping all the same functionality, I put the obfuscation level slider first, along with adding an image that would dynamically change based on the slider and the obfuscation type, which I put as the second option. Finally,

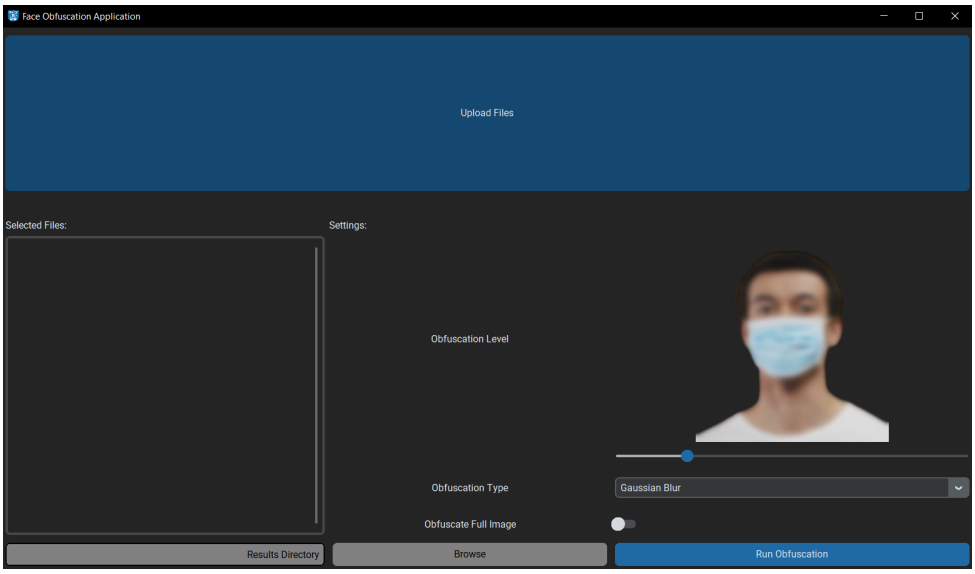


Figure 4.1: Facial Obfuscation Application

the last setting was a toggle to obfuscate the full image. I also added functionality to select a destination directory to which the obfuscated images/ videos will be saved. Lastly, I implemented the same button to run the obfuscation process, except this time I added some pop-up notifications that would inform the user if no target files or no destination path were selected. The final look of the application can be observed in Fig 4.1.

4.5 Summary

During the implementation of the facial obfuscation application, I took great consideration in ensuring that my implementation was full of useful functionality so that the user had full control over the obfuscation process. I also ensured that my code was clear and structured, allowing for simple future upgrades and additional implementation. These practices allowed me to produce a fully-fledged obfuscation system that can be easily integrated into various other systems in the future.

Chapter 5

Evaluation

Evaluation was an important aspect in the development of the facial obfuscation application as I had to determine the best computer vision framework to use for the face detection component. I also had to ensure that once the obfuscation process was complete, the facial features could still be detected for any potential future research.

5.1 Experiments

To experiment with the various face detection frameworks, I created a testing module which utilized OpenCV for general image processing and created a structure that read in images or a video frame by frame and called a function to detect faces for each library.

This allowed me to simply inject each library into the testing framework by creating a function that would detect faces in an image or frame and draw a rectangle around each found face using OpenCV. An example of such a function can be seen in Listing 5.1.

For the data that was used for evaluation, I decided I would test 6 cases. Each case had an image and a video to test both target media types which were taken from open source.

- **No faces:** This was done to see if any of the frameworks would produce false positives. The image was a cartoon illustration of an avocado (cromaconceptovisual (2022)) and the video is of a frog in the wild (Meditation_hypnosis (2022)).
- **Single face without facemask:** Test the basic functionality of being able to detect a single face within an image or video. For the image, I used a random picture of a student in public, looking at the camera with their head being skewed slightly to the side (CaiHuuThanh (2021)). For the video, I chose recording of a person working at a computer with the camera facing them at an angle, which may pose

as an issue due to some facial detection methods struggling with angled or skewed faces (MudanTV (2019)).

- **Single face with facemask:** Test the more advanced basic functionality of the framework being able to detect a face with a facemask. An image of a man in in a facemask in the woods facing slightly away from the camera is used (afreedpatan (2020)), along with a video of a doctor wearing a facemask is used as the test case (Amorn_mimi (2022)). The doctors face is moving and often skewed and angled which once again may pose an issue for some detection algorithms.

- **Multiple faces without facemasks:** This was to ensure that the framework I was choosing was capable of detecting multiple faces. For the image test case I implemented an image of people in a business meeting (089photoshootings (2017)). The video test case is a recording of people walking down a hall and talking as their faces increasingly get larger as they approach the camera (AlexKopeykin (2020)), this was to test the algorithms at varying face sizes.

- **Multiple faces with facemasks:** Similarly, the library also needed to be able to detect multiple faces with facemasks. The image is an image of a couple wearing facemasks (IqbalStock (2020)). The video is a 3840 x 2160 (4k) resolution video of a mother and a daughter walking in the park wearing facemasks (Fring (2020)). I decided to include this as I wanted to see if the resolution of the video would impact the processing time of some of the algorithms.

- **Multiple faces with and without facemasks:** This case was to validate that the choice of framework would be able to detect both faces with and without facemasks. This test case was the most important as this aligned the closest with our motivation to obfuscate faces that were in an uncontrolled environment, both with and without face masks. For the video I used a tutorial on how to wear a facemasks where multiple participants put on face masks (studio (2020)). For the image, I took a screenshot from the same video where one person wasn't wearing a mask while the other participants wore theirs.

I decided to test for two metrics, speed and accuracy, with the speed metric timing the speed of detecting faces within an image, drawing a bounding box around every found face, and saving the result. The accuracy was tested by manually labelling how many faces were present within each image and comparing them to the output. This was more complex for a video as it essentially consists of multiple images, meaning that I would have

to dissect each video into frames and hand label each frame and then dissect the resulting video, and compare the results frame by frame to get the most accurate evaluation. All tests were run on my machine which has an AMD Ryzen 7 4700U with Radeon Graphics (2.00 GHz) and 8.00 GB of DDR4 RAM.

To completely cover all the targets which arose from our motivation for this paper, I decided to add another experiment. This experiment was to evaluate if facial features could still be detected post-obfuscation.

```
def detect_faces_mtcnn(image):
    detector = MTCNN()

    detected_faces = detector.detect_faces(image)

    for face in detected_faces:
        (x, y, w, h) = face['box']
        cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0),
            2)

    return image
```

Listing 5.1: Detect Faces Using MTCNN

5.2 Results

5.2.1 Image Results

We can see the results of each of the 6 test cases for a single image in Table 5.1 for the speed evaluation and Table 5.2 for the results of the accuracy evaluation. From these results, we can preliminarily say that OpenCV and Dlib were the fastest to process a single image, however, this speed is accompanied by a lower accuracy with Dlib even being unable to detect masked faces. OpenCV on the other hand did average with along with being faster, this might have been thanks to my fine-tuning of the OpenCV Haar Cascade Classifier.

Looking further, we see that MTCNN was quite slow and yielded similar accuracy to that of OpenCV. Interestingly, OpenCV was not able to detect a face with no facemask

Figure 5.1: RetinFace Face Detection Results

which was quite clear, with the only difficulty was that it was at an angle, meaning that OpenCV's Haar Cascade Classifier could potentially struggle with face orientation. Continuing on and looking at the 3rd fastest and the most accurate framework, scoring 100% across all cases, is RetinaFace, which is quite impressive as it managed to do so in a reasonable time in comparison to face_recognition which also scored high in the accuracy benchmark but took substantially more time than the rest. For single-image processing, it is clear that RetinaFace is the best as it achieved perfect accuracy across all cases while doing so within average time. Surpisingly so, it was able to detect all faces in Fig 5.1, where one of the faces is practically unseen and all its features hidden, which I labelled as not a face initially but once I ran RetinFace and processed the image, I decided to be more strict with how I label the data as RetinFace allowed me to set a higher standard.

5.2.2 Video Results

Looking at the single-image results could be potentially deceiving since they're at a set environment that doesn't change, whereas for video this is completely different as each video consists of a multitude of images that can change drastically throughout the course of the video. With this, the speed will also take a massive hit compared to a single image, since for example, in a 10 second, 60 frames per second video, the system needs to analyze 600 images, not including basic operations such as reading in the video and writing out the output.

	OpenCV	**Dlib**	**face_recognition**	**MTCNN**	**RetinaFace**
No faces	0.27s	0.57s	315.36s	2.12s	1.81s
Single face without facemask	0.64s	0.56s	275.57s	7.02s	5.87s
Single face with facemask	0.26s	0.48s	206.71s	1.28s	2.44s
Multiple faces without facemasks	1.16s	0.60s	253.41s	7.66s	5.63s
Multiple faces with facemasks	0.36s	0.49s	267.47s	2.13s	1.95s
Multiple faces with & without facemasks	0.11s	0.38s	76.62s	1.13s	2.40s

Table 5.1: Speed Comparison for Image Face Detection with Various Frameworks

	OpenCV	**Dlib**	**face_recognition**	**MTCNN**	**RetinaFace**
No faces	0%	0%	0%	0%	0%
Single face without facemask	0%	100%	100%	100%	100%
Single face with facemask	0%	0%	0%	0%	100%
Multiple faces without facemasks	50%	67%	83%	67%	100%
Multiple faces with facemasks	50%	0%	100%	0%	100%
Multiple faces with & without facemasks	80%	60%	80%	80%	100%

Table 5.2: Accuracy Comparison for Image Face Detection with Various Frameworks

The speed results spiked drastically for all frameworks which was expected and can be seen in Table 5.3. These results follow a similar trend to the single-image results, where OpenCV and Dlib were the fastest compared to the rest, MTCNN and RetinaFace being around 10 times slower, and face_recognition taking such a long time that I could not get a result. If we take the average time of face_recognition to analyze each single image, and multiply that by 600, assuming we're analyzing a 10 second video at 60 frames per second, this would take us on average $600 * 232.52s = 139512s$, where 243.52 seconds is the average time taken by face_recognition to analyze and image. We see that this is quite infeasable as that equivilates to 38.75 hours per 10 second 60 frames per second video. I could not get a result for running face_recognition due to time constraints which were caused by resource constraints if I had been able to run this on a more powerful machine,

I could have gotten a result in a more timely manner, however, this was due to not having a more powerful machine available along with wanting to run it on an average machine to simulate an average user who might a better or worse machine than mine.

Now lets turn to the accuracy of these frameworks to detect faces in a video which can be seen in Table 5.4. We see that OpenCV once again hide quite poor results alongside Dlib where OpenCV generated a high amount of false positives along with struggling with faces that are skewed which is the case for both single face cases. This tells us that Dlib and OpenCV achieve their high processing speed by trading accuracy. MTCNN performed quite well but similarly to OpenCV, it struggled with skewed faces along with producing a number of false positives. Finally our previous favourite once again outdid itself by achieving an almost perfect score across all the test cases with the only issue arising then the face present was quite small and was practically fully occluded by a face mask and hair.

	OpenCV	**Dlib**	**face_recognition**	**MTCNN**	**RetinaFace**
No faces	55.73s	155.86s	*NaN*	835.42s	1154.85s
Single face without facemask	35.01s	86.05s	*NaN*	238.76s	614.07s
Single face with facemask	50.89s	119.45s	*NaN*	308.00s	651.16s
Multiple faces without facemasks	94.92s	224.32s	*NaN*	787.48s	1636.62s
Multiple faces with facemasks (4k)	765.00s	462.69s	*NaN*	1370.11s	987.85s
Multiple faces with & without facemasks	336.17s	540.42s	*NaN*	1317.54s	1133.05s

Table 5.3: Speed Comparison for Video Face Detection with Various Frameworks

5.3 Summary

Looking back at the results for both image and video test cases, we see that RetinaFace had the best performance compared to any of its competitors, with only face_recognition coming in close. However, another metric that we need to look at is the speed of each framework, and based on this metric RetinaFace's speed was about average compared to all the other frameworks, with face_recognition, which was the closest competitor to RetinaFace in terms of accuracy, didn't even complete as it took such a massive amount of time to analyze a single image compared to the other libraries.

	OpenCV	**Dlib**	**face_recognition**	**MTCNN**	**RetinaFace**
No faces	23.13%	0%	*NaN*	6.76%	0%
Single face without facemask	0%	0%	*NaN*	0%	100%
Single face with facemask	0%	66.67%	*NaN*	0%	100%
Multiple faces without facemasks	56.4%	30.77%	*NaN*	80%	100%
Multiple faces with facemasks (4k)	62.22%	46%	*NaN*	47.27%	92.73%
Multiple faces with & without facemasks	80.29%	63.82%	*NaN*	82.94%	100%

Table 5.4: Accuracy Comparison for Video Face Detection with Various Frameworks

Based on these results I decided to utilize RetinaFace's face detection in the facial obfuscation application as it was the most accurate out of all the other analyzed along with yielding an average performance compared to it's closest competitor in accuracy, face_recognition. As mentioned in the Implementation Chapter, changing the face detection component is quite simple which makes it easily upgradable and adaptable in the future. The function to detect faces can be observed in Listing 5.2.

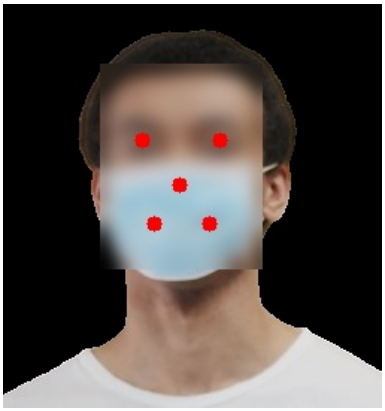
After integrating RetinaFace into the facial obfuscation application, I ran the program to observe its functionality. I tested both the pixelation and blurring, with blurring level being left as default and pixelation level being a little over half since in reality the pixelation impact will be a lot less than in the obfuscation simulation in the GUI since the pixelation error is smaller, meaning pixelation level needs to be greater as it has less pixels to shift. To test if the final requirement of our application that arose based on our motivation was complete, I attempted to detect facial features from the obfuscated images utilizing RetinaFace and its feature extraction functionality, results of which can be seen in Fig 5.2 for pixelation and Fig 5.3 for blurring. Based on these results, it is clear that our obfuscation application is capable of hiding the identity of the individual while retaining the ability to detect the location of their facial features to allow us to perform future research if we need to all while the individual can be rest assured that their identity will be hidden.

(a) Pixilated Image

(b) Feature Detection on the Pixilated Image

Figure 5.2: Post-Pixilation Feature Detection

(a) Blurred Image



(b) Feature Detection on the Blurred Image

Figure 5.3: Post-Blurring Feature Detection

```
def detect_faces(self, image: cv2.typing.MatLike) -> dict:
    faces = RetinaFace.detect_faces(image)
    return faces
```

Listing 5.2: Obfuscate Faces Function

Chapter 6

Conclusions & Future Work

Research and development within the topics of facial detection and recognition should always be accompanied by great consideration for privacy to not impact people's individuality. This is especially true in children who are the most susceptible to influence which may result in them losing their uniqueness and force them to subscribe to a society-written narrative which we want to prevent.

Looking back at the Evaluation Chapter of this paper, it is clear that our chosen face detection framework and overall system perform the target tasks efficiently and accurately. However, the field is ever-evolving meaning that eventually a better technique or framework will be developed which is better suited for our requirements than RetinaFace, luckily, I ensured that the implementation accounts for this and makes replacement simple.

6.1 Future Work

Upon completion, I would've liked to do some more work on the implementation of the application such as training a model myself if I had access to a better machine that would only detect and obfuscate certain faces. This would be particularly useful for obfuscating general citizens in surveillance systems while exposing criminals and wanted people.

The developed facial obfuscation application ensures that it tackles and solves the initial motivation, by being able to analyze the entire dataset of children doing various physical activities and obscuring their faces so that their identity is hidden and cannot be recognized ensuring their privacy is kept safe. However, this application can be much more than that, by building it to be API-like I allowed potential future integration into other systems, not just an application, such as surveillance systems in schools, however, changes

need to be made to only obfuscate faces of children but not adults. I would also have liked to deploy this API on a server which would allow developers to call the API by passing the image or video along with some settings to customize the obfuscation to their needs, after which it would return the obfuscated image. Unfortunately, I do not possess the funding that would allow me to upkeep such a server.

Overall, the facial obfuscation fulfilled the initial requirements by providing a customizable system that allows the user to obfuscate faces to their preference, by selecting the obfuscation level, obfuscation type, and whether they want just the faces to be obfuscated, or the entire image.

Bibliography

089photoshootings (2017). People business meeting - free photo on pixabay - pixabay.

afreedpatan (2020). Man model casual facemask - free photo on pixabay - pixabay.

AI, T. (2023). Top 10 open source facial recognition libraries and tools.

AlexKopeykin (2020). Office people business work team - free video on pixabay - pixabay.

Amorn_mimi (2022). Medical hospital health doctor - free video on pixaby - pixaby.

Andrejevic, M. and Selwyn, N. (2019). Facial recognition technology in schools: critical questions and concerns. *Learning, Media and Technology*, 45:1–14.

Bank, R. (2021). Pyqt6.

Bledsoe, W. W. (1964). The model method in facial recognition. *Technical Report*.

Bolger, R. (2018). Cafe app that knows how you take your coffee sparks security concerns.

Borsak, V. (2024). Borsakv/cs7092-dissertation.

Burgess, M. (2018). Facial recognition tech used by uk police is making a ton of mistakes.

CaiHuuThanh (2021). Girl student asian uniform - free photo on pixabay - pixabay.

Chandrappa, D., Ravishankar, M., and RameshBabu, D. (2011). Face detection in color images using skin color model algorithm based on skin color information. In *2011 3rd International Conference on Electronics Computer Technology*, volume 1, pages 254–258. IEEE.

CM, B. (2023). Face recognition in python: A comprehensive guide.

Cootes, T., Baldock, E., and Graham, J. (2000). An introduction to active shape models. *Image processing and analysis*, 328:223–248.

Croft, W. L., Sack, J.-R., and Shi, W. (2022). Differentially private facial obfuscation via generative adversarial networks. *Future Generation Computer Systems*, 129:358–379.

cromaconceptovisual (2022). Avocado food fruit - free illustration on pixabay - pixabay.

Cuimei, L., Zhiliang, Q., Nan, J., and Jianhua, W. (2017). Human face detection algorithm via haar cascade classifier combined with three additional classifiers. In *2017 13th IEEE International Conference on Electronic Measurement & Instruments (ICEMI)*, pages 483–487. IEEE.

Dang, K. and Sharma, S. (2017). Review and comparison of face detection algorithms. In *2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence*, pages 629–633. IEEE.

DataScientist (2024). Pygtk: The python gui creation tool.

Davebaraka (2020). Davebaraka/face-obfuscator: Browser extension that uses face detection and recognition algorithms to dynamically block pre-trained faces from images loaded in chrome.

Deng, J., Guo, J., Zhou, Y., Yu, J., Kotsia, I., and Zafeiriou, S. (2019). Retinaface: Single-stage dense face localisation in the wild. *arXiv preprint arXiv:1905.00641*.

Dlib (2022). Dlib c++ library.

Dongare, A., Kharde, R., Kachare, A. D., et al. (2012). Introduction to artificial neural network. *International Journal of Engineering and Innovative Technology (IJEIT)*, 2(1):189–194.

Farfade, S. S., Saberian, M. J., and Li, L.-J. (2015). Multi-view face detection using deep convolutional neural networks. In *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, pages 643–650.

Faruqe, M. O. and Hasan, M. A. M. (2009). Face recognition using pca and svm. In *2009 3rd international conference on anti-counterfeiting, security, and identification in communication*, pages 97–101. IEEE.

Flanagan, D. (2005). *Java in a Nutshell*. " O'Reilly Media, Inc.".

Fring, G. (2020). Mother and daughter with face masks at the park - free video on pexels-pexels.

Geitgey, A. (2020). Face-recognition.

GfG (2023). Introduction to tkinter.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, 63(11):139–144.

Guo, G. and Zhang, N. (2019). A survey on deep learning based face recognition. *Computer Vision and Image Understanding*, 189:102805.

Harmon, A. (2019). As cameras track detroit's residents, a debate ensues over racial bias.

Harwell, D. (2018). Facial recognition may be coming to a police body camera near you.

Henstridge, J. (2003). Pygtk.

Hillman, C., Logan, N., and Shigeta, T. (2019). A review of acute physical activity effects on brain and cognition in children. *Translational Journal of the American College of Sports Medicine*, 4.

Hummel, R. A., Kimia, B., and Zucker, S. W. (1987). Deblurring gaussian blur. *Computer Vision, Graphics, and Image Processing*, 38(1):66–80.

Insaf, A., Ouahabi, A., Benzaoui, A., and taleb ahmed, A. (2020). Past, present, and future of face recognition: A review. *Electronics*, 9:1188.

IqbalStock (2020). Couple social distancing - free illustration on pixabay - pixabay.

Ismail, N. and Sabri, M. I. M. (2009). Review of existing algorithms for face detection and recognition. In *8th WSEAS International Conference on Computational Intelligence, Man-Machine Systems and Cybernetics*, pages 30–39. Citeseer.

Jiang, H. and Learned-Miller, E. (2017). Face detection with the faster r-cnn. In *2017 12th IEEE international conference on automatic face & gesture recognition (FG 2017)*, pages 650–657. IEEE.

Jordan, D. (1990). Implementation benefits of c++ language mechanisms. *Communications of the ACM*, 33(9):61–64.

Justadudewhohacks (2018). Justadudewhohacks/face-api.js: Javascript api for face detection and face recognition in the browser and nodejs with tensorflow.js.

Kaplan, S. (2023). To be a face in the crowd: Surveillance, facial recognition, and a right to obscurity. In Samuelsson, L., Cocq, C., Gelfgren, S., and Enbom, J., editors, *Everyday Life in the Culture of Surveillance*, pages 45–66. NORDICOM.

Kass, M., Witkin, A., and Terzopoulos, D. (1988). Snakes: Active contour models. *International journal of computer vision*, 1(4):321–331.

Kaste, M. (2018). Orlando police testing amazon's real-time facial recognition.

Kernighan, B. W. and Ritchie, D. M. (2002). *The C programming language*. Pearson Education Asia.

Khan, F. (2022). Modern gui using tkinter.

Khan, M., Chakraborty, S., Astya, R., and Khepra, S. (2019). Face detection and recognition using opencv. In *2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, pages 116–119. IEEE.

King, D. E. (2009). Dlib-ml: A machine learning toolkit. *The Journal of Machine Learning Research*, 10:1755–1758.

Kivy (2010). Kivy/kivy: Open source ui framework written in python, running on windows, linux, macos, android and ios.

Kumar, A., Kaur, A., and Kumar, M. (2019). Face detection techniques: a review. *Artificial Intelligence Review*, 52(2):927–948.

Kummarikuntla, T. (2024). A detailed guide to choosing the best python gui framework.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

Lin, L. and Purnell, N. (2019). A world with a billion cameras watching you is just around the corner.

Ludvigsson, J. (2020). Children are unlikely to be the main drivers of the covid-19 pandemic - a systematic review. *Acta Paediatrica*, 109.

Lundh, F. (1999). An introduction to tkinter. *URL: www. pythonware. com/library/tkinter/introduction/index. htm*.

Ly, B., Dyer, E., Feig, J., Chien, A., and Bino, S. (2020). Research techniques made simple: Cutaneous colorimetry: A reliable technique for objective skin color measurement. *The Journal of investigative dermatology*, 140:3–12.e1.

Marco Ciotti, Massimo Ciccozzi, e. a. (2020). The covid-19 pandemic. *Critical Reviews in Clinical Laboratory Sciences*, 57(6):365–388. PMID: 32645276.

Meditation_hypnosis (2022). Frog reeds green - free video on pixabay - pixabay.

MudanTV (2019). Typing computer working office - free video on pixabay - pixabay.

Navneet, D. (2005). Histograms of oriented gradients for human detection. In *International Conference on Computer Vision & Pattern Recognition, 2005*, volume 2, pages 886–893.

Nederkoorn, C. (2021). Which gui framework is the best for python coders?

of Massachusetts, U. (2018). Labeled faces in the wild home.

OpenCV (2020).

O'Mahony, N., Campbell, S., Carvalho, A., Harapanahalli, S., Hernandez, G. V., Krpalkova, L., Riordan, D., and Walsh, J. (2020). Deep learning vs. traditional computer vision. In *Advances in Computer Vision: Proceedings of the 2019 Computer Vision Conference (CVC), Volume 1 1*, pages 128–144. Springer.

Padhee, S. (2012). Automatic face detection using color based segmentation. *International Journal of Scientific Research*.

Payne, R. (2024). Advantages and disadvantages of java.

Phillips, P., Flynn, P., Scruggs, T., Bowyer, K., Chang, J., Hoffman, K., Marques, J., Min, J., and Worek, W. (2005). Overview of the face recognition grand challenge. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 947–954 vol. 1.

Phillips, P., Wechsler, H., Huang, J., and Rauss, P. J. (1998). The feret database and evaluation procedure for face-recognition algorithms. *Image and Vision Computing*, 16(5):295–306.

Phung, S., Bouzerdoum, A., and Chai, D. (2005). Skin segmentation using color pixel classification: Analysis and comparison. *IEEE transactions on pattern analysis and machine intelligence*, 27:148–54.

Polo, G. (2017). Pygtk, pyqt, tkinter and wxpython comparison.

Rewar, E. and Lenka, S. (2013). Comparative analysis of skin color based models for face detection. *Signal & Image Processing : An International Journal*, 4:69–75.

Rosebrock, A. (2023). Face detection with dlib (hog and cnn).

Rowley, H. A., Baluja, S., and Kanade, T. (1998). Neural network-based face detection. *IEEE Transactions on pattern analysis and machine intelligence*, 20(1):23–38.

Saikia, S., Fidalgo, E., Alegre, E., and Fernández-Robles, L. (2017). Object detection for crime scene evidence analysis using deep learning. In *International Conference on Image Analysis and Processing*, pages 14–24.

Schmidhuber, J. (2014). Deep learning in neural networks: An overview. *Neural Networks*, 61.

Serengil, S. I. (2021). Retina-face.

Shang-Hung, L. (2000). An introduction to face recognition technology. *Informing Science The International Journal of an Emerging Transdiscipline*, 3.

Sharma, S. P. (2022). What makes python a poor choice for large-scale full-stack development?

Shavers, C., Li, R., and Lebby, G. (2006). An svm-based approach to face detection. In *2006 Proceeding of the Thirty-Eighth Southeastern Symposium on System Theory*, pages 362–366. IEEE.

Streamingo.ai (2023). Face blur: An ai-powered solution for privacy protection.

Stroustrup, B. (2013). *The C++ programming language*. Pearson Education.

studio, c. (2020). People showing how to wear face mask while the man is being funny - free video on pexels- pexels.

Sun, X., Wu, P., and Hoi, S. C. (2018). Face detection using deep learning: An improved faster rcnn approach. *Neurocomputing*, 299:42–50.

Tnouvel (2022). Tnouvel/face-obfuscation: A simple face obfuscation program written in python and opencv.

TomSchimansky (2021). Tomschimansky/customtkinter: A modern and customizable python ui-library based on tkinter.

Turk, M. and Pentland, A. (1991). Eigenfaces for Recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86.

van Rossum, G. and Drake, F. L. (2006). *An introduction to Python*. Network Theory Limited.

Viduka, D., Kraguljac, V., and Ličina, B. (2021). A comparative analysis of the benefits of python and java for beginners. *Quaestus*, pages 318–327.

Worby, C. J. and Chang, H.-H. (2020). Face mask use in the general population and optimal resource allocation during the covid-19 pandemic.

wxWidgets (2012). Wxwidgets/phoenix: Wxpython's project phoenix. a new implementation of wxpython, better, stronger, faster than he was before.

Yuille, A. L., Hallinan, P. W., and Cohen, D. S. (1992). Feature extraction from faces using deformable templates. *International journal of computer vision*, 8:99–111.

Zhang, K., Zhang, Z., Li, Z., and Qiao, Y. (2016). Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE signal processing letters*, 23(10):1499–1503.