ATTENDANCE MONITORING SYSTEM USING FACE RECOGNITION A CASE STUDY OF THE DEPARTMENT OF COMPUTER SCIENCE UNIVERSITY OF BENIN (UNIBEN)

BY

YANGE LUTHER
(PSC1808996)

DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF PHYSICAL SCIENCE
UNIVERSITY OF BENIN
BENIN CITY
NIGERIA.

SEPTEMBER, 2023

ATTENDANCE MONITORING SYSTEM USING FACE RECOGNITION A CASE STUDY OF THE DEPARTMENT OF COMPUTER SCIENCE UNIVERSITY OF BENIN (UNIBEN)

BY

YANGE LUTHER

(PSC1808996)

PROJECT REPORT SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE, FACULTY OF PHYSICAL SCIENCES, UNIVERSITY OF BENIN, EDO STATE, NIGERIA, IN PARTIAL FULFILMENT OF THE REQUIRMENTS FOR THE AWARD OF BACHELOR OF SCIENCE (B.Sc.) DEGREE IN COMPUTER SCIENCE.

ATTESTATION

I, Yange Luther attest that this project work titled " ATTENDANCE MONITORING SYSTEM USING FACE RECOGNITION A CASE STUDY OF THE DEPARTMENT OF COMPUTER SCIENCE UNIVERSITY OF BENIN" is an original work and was carried out by me.

_____ _____

Yange Luther **DATE**

Student

CERTIFICATION

This is to certify that this project work was carried out by **Yange Luther**, with matriculation number **PSC1808996** in the Department of Computer Sciences, University of Benin, and it is adequate in scope and content for the award of Bachelor Science Degree in Computer Science of the University of Benin.

Mr. Oliomogbe, S.O.P _____

_____ **DATE**

APPROVAL

This project is hereby approved by the department of computer science in partial fulfillment of the requirement for the award of Bachelor of Science Degree (B.Sc.) in Computer Science of the University of Benin, Benin City, Nigeria.

_____

PROF. (MRS.) A.O. EGWALI _____

(Head of Department) **DATE**

DEDICATION

This project is dedicated to God Almighty, beloved family, close friends, and supportive associates including my Course mates, expressing gratitude for their unwavering love, encouragement, and invaluable contributions throughout my academic journey.

ACKNOWLEDGEMENTS

My profound gratitude is to God Almighty for the knowledge, wisdom and understanding that he bestowed on me to carry out this project.

I appreciate my supervisor MR. OLIOMOGBE S.O.P for the supervision and support that he gave me, which helped in the progress and smoothness of this project.

I am also grateful to the entire computer science department of the University of Benin, the H.O.D Prof.(Mrs.)Egwali, and all the lecturers and all non-academic staff of the Department of Computer Science, University of Benin, Benin City, Edo State, Nigeria, who all prepared me for the skills I currently have in Computer Science.

A very special thanks to all my friends and especially my family. This project would not have been possible without them who supported me morally and financially. May God Almighty continue to bless each and every one you.

Contents

ABSTRACT

Face detection and recognition are computer vision techniques used to identify and locate human faces within digital images or video frames, and to subsequently analyze and verify the identity of individuals based on their unique facial features. Some of the algorithms used for implementing face detection and recognition are Local Binary Pattern, Histogram of oriented Gradient, Linear discriminant analysis, and convolutional neural networks with classifier such as Support Vector machine, e.t.c. Face detection and recognition system has become relevant in security and access control, automated Attendance tracking, user authentication, Biometric identification, Human Computer Interaction and many other areas.

This project is implemented using python programming language, because python programming language allows programmers flexibility, therefore is of no threat to write-ability, readability and reliability, it has it libraries for the implementation of the project. And the test run of the project result is contained in Appendix D

CHAPTER ONE

1.0 INTRODUCTION

A facial recognition system represents an advanced technology capable of identifying a human face within a digital image or video frame. This identification process occurs by comparing the facial features extracted from the image to a pre-existing database of faces. Facial recognition systems serve a multitude of purposes, predominantly within user authentication and ID verification contexts, allowing for the precise analysis and measurement of facial characteristics to establish an individual's identity.

The evolution of facial recognition systems dates back to the 1960s when it emerged as a pioneering computer application. Over the ensuing decades, these systems have expanded their reach, finding extensive applications in smartphones and various technological domains, including robotics. Given its reliance on measuring physiological traits, facial recognition falls under the broader category of biometrics.

Facial recognition systems have been adopted across a spectrum of domains, including advanced human-computer interaction, video surveillance, and automatic image indexing. Governments and private enterprises around the world have eagerly embraced these systems for a myriad of purposes.

A historical perspective on facial recognition technology reveals its early stages, characterized by human involvement in establishing facial feature coordinates to aid computer recognition. Significant breakthroughs occurred in the 1990s with the introduction of techniques such as principal component analysis (PCA) and Eigenface, which substantially reduced data processing requirements. The Bochum system, incorporating Gabor filters and face structure grids, further advanced the field, outperforming its contemporaries. The advent of the Viola-Jones algorithm marked a pivotal moment, enabling real-time face detection in videos, thereby facilitating practical applications and enhancing user interfaces.

One notable application of facial recognition technology is observed in Ukraine, where Clear view AI facial recognition software is employed to identify deceased Russian soldiers and establish contact with their families. This application forms part of psychological warfare tactics aimed at destabilizing the Russian government. However, it is imperative to acknowledge that the utilization of such technology in warfare scenarios raises profound ethical concerns, potentially impacting public perception and international relations.

Facial recognition systems can be delineated into four distinct modules, as depicted in Figure1.1 on Appendix B. Face localization, normalization, feature extraction, and matching. The versatility of these systems is evident in their widespread adoption, spanning activities such as person identification on social media platforms, disease diagnosis, autonomous driving, mobile phone unlocking, and their frequent integration with fingerprints for robust fraud detection. Additionally, their prevalence in security applications is notable, primarily driven by the advantages of contactless person verification (Anil Poudel, 2021).

1.1 BACKGROUND OF STUDY

Facial recognition technology has gained significant attention in recent years due to its wide range of applications and potential to revolutionize various industries. It is a biometric technology that utilizes computer algorithms to identify and verify individuals based on their facial features. By analyzing unique facial patterns, such as the arrangement of eyes, nose, and mouth, facial recognition systems can accurately match faces against a database of known individuals or detect and identify unknown individuals. The development of facial recognition technology can be traced back to the 1960s when researchers began exploring the feasibility of automated face recognition. Over the years, advancements in computer vision, machine learning, and data processing have contributed to significant improvements in the accuracy and performance of facial recognition systems. The widespread adoption of facial recognition technology has led to its application in numerous fields. In the field of security

and access control, facial recognition is used to enhance building and facility security by replacing traditional access methods like keycards or passwords. It enables quick and accurate identification of authorized individuals, thereby reducing the risk of unauthorized access. Facial recognition technology also plays a crucial role in law enforcement and public safety. It aids in criminal investigations by matching faces captured in surveillance footage with known offenders in criminal databases, helping to identify suspects and solve crimes. Additionally, it assists in locating missing persons and monitoring public spaces for potential security threats (Anil Paul, 2021). Moreover, facial recognition has found its way into consumer applications, providing personalized user experiences. It is used for targeted advertising, content recommendations, and customized services based on individual preferences and demographics. Additionally, facial recognition technology has been implemented in educational institutions and workplaces to automate attendance tracking, replacing manual processes and improving efficiency.

1.2 SCOPE OF THE PROJECT

With the goal of developing a Face Detection and Recognition Time Monitoring System, the scope of the project is itemized below;

- **Facial Recognition Algorithm:** The development and integration of an appropriate facial recognition algorithm that can effectively analyze and match facial features for identification and verification purposes.
- **Database Management:** The creation and management of a database to store the facial data of authorized individuals, along with their corresponding time-related information.
- **User Interface:** The design and development of a user-friendly interface that allows users to interact with the system by registering their faces.

- **Real-time Monitoring:** The implementation of real-time monitoring capabilities, enabling the system to capture and process facial images in real-time, ensuring accurate and up-to-date attendance records.
- **Reporting System:** The system deploys the Comma Separated Value (CSV) which can be viewed in an excel spreadsheet format. Identity and Time are recorded on these file.

1.3 SIGNIFICANCE OF THE PROJECT

This project is significant in posting benefits in the following areas;

a. Improved Efficiency: The system offers an automated and efficient approach to monitoring individual, eliminating the need for manual processes such as paper-based registers or manual check-ins.
b. My model is advanced in enhancing accuracy since it utilizes advanced face detection and recognition algorithms, so it minimizes the risk of errors or fraudulent records, ensuring the reliability of data.
c. It also allows for prompt identification of individuals absence within a given frame of time.
d. The model do not allow impersonation compared to RFID's
e. Time Management: Individuals do not have to stand and wait at a RFID's stand before taking record of their time.
f. Scalability and Adaptability: The study contributes to the development of a flexible system that can be easily scaled and adapted to various educational institutions, organizations, or workplaces. It can accommodate different settings, such as classrooms, offices, or events, making it applicable in diverse environments.

1.4 AIM AND OBJECTIVES OF THE PROJECT

The aim of this project is to design and implement a face detection and recognition time monitoring system with the use of some exiting algorithms, it is set to be achieved with some objectives as stated below;

i. Replacing RFID system with automated monitoring system.
ii. Implementing this model and minimize fraudulent activities and ensure only authorized personnel access the premises.
iii. Provide reliable information to make informed decision.
iv. Handling large data in an organization is made easy.

1.5 MOTIVATION FOR THE PROJECT

This project is motivated by;

a. The failure of RFID card to stamp a student check in time.
b. Impersonation of identity, where another individual get to use card not belonging to them.
c. The use of card was a time consuming process, individual get to stay on a queue to wait in turns.

1.6 LIMITATIONS OF THE PROJECT

In the cause of this project I encountered some challenges, some of them are itemized below;

i. Financial Limitation.

- Cost of getting a computer with a good processor.
- Cost of getting high graphic resolution camera.
- Cost of paying for internet subscription to make some researches.

ii. **Data Limitation**

- Gathering dataset for training a model
- Getting quality images for testing efficiency

1.7 METHODOLOGY FOR THE PROJECT.

a. **Review of relevant literatures:** Review of exiting literature on face detection and recognition algorithms relevant to this project.

b. **Analyzing existing system:** With a view to collate and compare the strength and weaknesses of the existing system.

c. **The use software engineering model:** The software engineering model are often referred to as software development models or software development methodologies, they are systematic approaches used to plan, design, build, test, and maintain software systems. These models provide a structured framework for organizing and managing the software development process, examples of such model includes Waterfall model, Agile Model, iterative model, e.t.c.

For this project will would be adopting the water fall model.

1.7.1 WATERFALL MODEL

The Waterfall Model is one of the oldest and most traditional software development methodologies. It follows a sequential and linear approach to software development, where each phase must be completed before moving on to the next. It is called the "Waterfall" model because progress flows in one direction, just like a waterfall.

The phases involved are itemized below

a. **Requirements Analysis:** In this initial phase, the project team gathers and documents all the requirements for the software. This includes understanding the

needs of stakeholders, defining functional and non-functional requirements, and creating a detailed requirements specification.

b. **System Design:** Once the requirements are well-defined, the system design phase begins. Here, the high-level architecture and structure of the software are planned. This phase involves creating system diagrams, specifying hardware and software components, and defining data structures.

c. **Implementation (Coding):** In this phase, developers start writing the actual code for the software based on the design specifications. They follow coding guidelines and best practices to ensure code quality.

d. Testing: After the code is written, it moves to the testing phase. Testers execute various types of testing, including unit testing, integration testing, system testing, and user acceptance testing (UAT). The goal is to identify and fix defects and ensure that the software meets the specified requirements.

We deploy the bottom up testing approach;

Bottom-Up Testing is a software testing approach that begins by testing individual components or units of a software system and gradually integrates them to test higher-level components and the entire system. Key points about bottom-up testing include:

- It starts with the smallest units, such as functions or modules, and progressively moves to test larger components, eventually reaching the top-level components or the entire system.
- The focus is on ensuring the correctness of individual components, identifying issues at the unit level, and then gradually validating the interactions between these components.
- Bottom-up testing does not require stubs or placeholders for lower-level components, as each unit is tested independently.

- It is effective at early detection of issues in individual components, allowing for isolated testing and debugging.

e. **Deployment (Integration and Deployment):** Once the software has passed all testing phases and is deemed stable and ready, it is deployed to the production environment. This phase may involve installation, configuration, and data migration activities.

f. **Maintenance and Support:** The final phase is the maintenance and support of the software in the production environment. It includes addressing user issues, fixing defects, and implementing updates or enhancements as needed. Maintenance can extend throughout the software's lifecycle. **See Figure 1.2 on Appendix B**

1.8 DEFINATION OF TERMS

The definitions of some terms used for this project are in **Table 1.9 of Appendix A**

1.9 ALGORITHMS AND COMPLEXITIES

Algorithms are step-by-step procedures or instructions used to solve a problem or perform a specific task. They are the foundation of computer programming and are designed to be executed by a computer or any computational device.

Algorithmic complexity, also known as computational complexity, refers to the analysis of the resources required by an algorithm to solve a problem. It measures the efficiency and scalability of an algorithm as the input size increases. The most commonly analyzed resources are time complexity and space complexity.

Time complexity is a measure of the amount of time an algorithm takes to run as a function of the input size. It helps us understand how the algorithm's performance scales with larger inputs. Time complexity is typically expressed using Big O notation, which provides an upper bound estimation of the growth rate of the algorithm. For example, an

algorithm with a time complexity of O(n) means that the running time grows linearly with the input size. Space complexity is a measure of the amount of memory or storage space an algorithm requires as a function of the input size. It helps us understand how **efficiently an algorithm** utilizes memory resources. Space complexity is also expressed using Big O notation, similar to time complexity. For example, an algorithm with a space complexity of O(n) means that the memory usage grows linearly with the input size.

Different algorithms have different complexities, and it's essential to choose an algorithm that suits the problem's requirements. Analyzing algorithms and understanding their complexities is crucial for designing efficient and scalable software systems. It helps in evaluating and comparing different algorithms, optimizing code, and predicting performance. (John Bullinaria, 2019).

In the next chapter we should be reviewing some literatures on algorithms used in face detection and recognitions such as Convolutional Neural Networks, Local Binary Pattern, Histogram of Oriented gradient, Linear Discriminant Analysis e.t.c.

CHAPTER TWO
LITERATURE REVIEW

2.1 OVERVIEW OF SOME ALGORITHMS USED IN FACE DETECTION AND RECOGNITION

Some of the algorithms overviewed are; [1] Local Binary Pattern (LBP), [2] Histogram of Oriented Gradient (HOG), [3] Eigenfaces and Principal Component Analysis (PCA), [4] Viola-Jones Algorithm, [5] Scale-Invariant feature transform (SIFT), [5] Convolutional Neural Networks, [6]FaceNet, [7] DeepFace, [8] Multi-Task Cascaded Convolutional Network(MTCNN), [9] Ada Boost algorithm and the cascade classifier, [10] Linear Discriminant Analysis (LDA), [11] Support Vector Machine (SVM).

2.3 LOCAL BINARY PATTERN

LBP is a texture descriptor used for image analysis and pattern recognition tasks. It aims to capture the local structure of an image by comparing each pixel with its neighboring pixels. The properties of linear binary pattern includes the it computational efficiency, invariant to monotonic gray-scale level changes, and robust to noise, it was found by Timo Ojala, Matti Pietikäinen, and Topi Mäenpää in the year 1994. LBP is a texture descriptor used for image analysis and pattern recognition tasks. It aims to capture the local structure of an image by comparing each pixel with its neighboring pixels. The basic LBP operator computes a binary code based on the intensity values of a pixel's neighbors. Uniform LBP categorizes patterns into uniform and non-uniform, considering the number of transitions in the binary code. Rotation Invariant LBP ensures rotation invariance by incorporating circularly shifted versions of the pattern. Extended LBP includes spatial relationships by assigning weights to neighbors' codes. Completed LBP addresses incomplete patterns at image borders. Variations like LTP, LDP, and LTrP offer different relationships for texture

classification. Multi-scale LBP involves applying LBP at different scales to capture patterns at various levels of detail. The choice of approach depends on the specific application and requirements. LBP has a Linear Time Complexity of O(n) (Maturana et al, 2009)**.**. The limitation of LBP is that it's sensitive to variations in illumination and may not perform well in scenarios with complex textures or significant scale variations.

2.4 HISTOGRAM OF ORIENTED GRADIENT (HOG)

The Histogram of Oriented Gradients (HOG) algorithm extracts features from images by analyzing local texture information. It involves preprocessing the image, computing gradients to capture intensity variations, and quantizing gradient orientations into bins. Histograms of gradient orientations are formed for small image cells. Block normalization is applied to account for lighting and contrast variations. The normalized histograms are concatenated to create a feature descriptor vector representing the entire image. This descriptor captures the distribution of local texture patterns. The HOG descriptor can be used with a classifier, such as SVM, for tasks like object detection or classification. *HOG is effective for pedestrian detection*, object recognition, and image segmentation. It provides a reliable representation of images based on gradient orientations. The algorithm is widely used in computer vision due to its ability to capture local texture. It was derived by Navneet Dalal and Bill Triggs in 2005. computes the gradient magnitude and orientation for each pixel in an image. These gradient values are then used to form a histogram of gradient orientations, which represents the object's shape and texture details (N. Dalal and Triggs, 2005). It has widely used in object detection tasks, particularly in pedestrian detection. HOG can struggle with handling fine-scale details and small object sizes. It may also be affected by variations in lighting conditions and occlusions.

2.5 EIGENFACES AND PRINCIPAL COMPONENT ANALYSIS

In the 1980s, the Eigenfaces algorithm, introduced by Turk and Pentland, revolutionized the field of face recognition. Eigenfaces utilized Principal Component Analysis (PCA) to extract a low-dimensional representation of face images, capturing the most discriminative features. This technique laid the foundation for subsequent developments in feature extraction and dimensionality reduction for face recognition. It uses the eigenvectors of the covariance matrix of face images to create a low-dimensional representation of faces. Eigenfaces is a face recognition technique that utilizes Principal Component Analysis (PCA) to extract essential features from face images. The approach involves preprocessing the images, representing them as vectors, and applying PCA to compute eigenfaces. Eigenfaces capture the most significant variations in the face dataset and can be used for dimensionality reduction. Face recognition is performed by projecting new faces onto the eigenfaces and comparing them to a database using similarity measures. Various enhancements and variations have been introduced to improve eigenfaces' performance. While deep learning methods have surpassed eigenfaces in recent years, they remain a fundamental concept in face recognition and feature extraction.(Georgescu D, 2011).

It time complexity is of $O(n^3)$. Eigenfaces and PCA may be sensitive to variations in lighting conditions, facial expressions, and pose changes. They may not be as effective in scenarios with significant appearance variations.

2.6 VIOLA JONES

In 2001, Viola and Jones presented a pioneering algorithm for real-time face detection. Their method utilized Haar-like features and introduced the concept of the integral image to efficiently compute features. By employing an AdaBoost classifier, the Viola-Jones algorithm achieved significant speed and accuracy improvements,

making it widely adopted for face detection applications. Complexity of O(n). Viola-Jones may struggle with detecting objects in complex backgrounds or with severe occlusions. It can also have higher false positive rates in certain scenarios. **The Viola/Jones Face Detector (2001)**

2.7 SCALE INVARIANT FEATURE TRANSFORM (SIFT)

SIFT is a feature extraction algorithm used for image matching and object recognition. It aims to find distinctive and invariant features in an image that are robust to scale, rotation, and affine transformations. Is was authored by David G. Lowe in the year 1999. SIFT detects interest points in an image based on the local intensity extrema in the scale space. It then describes these interest points using histograms of gradient orientations in the surrounding regions. SIFT may be sensitive to variations in viewpoint and occlusions (Pavithra R. et al, 2014). It may also have challenges with real-time processing due to its computational demands.

2.8 CONVOULTIONAL NEURAL NETWORKS (CNN)

CNNs, or Convolutional Neural Networks, have a rich history in computer vision. They were inspired by the Neocognitron model proposed by Kunihiko Fukushima in the 1980s. LeNet-5, developed by Yann LeCun in the early 1990s, introduced the concept of convolutional layers. Further advancements came with AlexNet in 2012, which popularized deep CNNs, and subsequent models like VGGNet, GoogLeNet, and ResNet. These models propelled CNNs to dominate image classification challenges, and their success continued with advancements in object detection, semantic segmentation, and other visual tasks. Today, CNNs are foundational in computer vision and have greatly influenced the field. Overall, the methodology and

approach of CNNs involve designing an architecture with appropriate layers, training the network using backpropagation, tuning hyperparameters, and evaluating the model's performance. This hierarchical and locally focused approach has been successful in various computer vision tasks, enabling the analysis (ABHISHEK et al, 2017**)**.. Deep learning models require significant amounts of labeled training data. They can also be prone to overfitting and may lack interpretability compared to traditional algorithms.

2.9 FACENET ALGORITHM

FaceNet is an algorithm for face recognition that was introduced in a paper by Florian Schroff, Dmitry Kalenichenko, and James Philbin in 2015. FaceNet was developed as a response to the challenges in face recognition, aiming to address the limitations of traditional methods such as Eigenfaces and local feature-based approaches. It leverages the power of deep learning and convolutional neural networks (CNNs) to learn discriminative face embedding's. Deep Convolutional Neural Network: FaceNet employs a deep CNN architecture to learn facial features and generate face embeddings. It consists of multiple convolutional layers, non-linear activation functions (e.g., ReLU), and fully connected layers. The network is trained on large-scale labeled face datasets. FaceNet requires a substantial amount of labeled face data for training. Data augmentation techniques, such as random cropping, flipping, and rotation, are applied to increase the robustness and generalization of the model. Whenever there's a low resolution of images, it affects it performance, which means it may not perform optimally thereby results becomes unreliable.

2.10 DEEPFACE

DeepFace is a facial recognition algorithm developed by Facebook's AI Research (FAIR) team in 2014. It utilizes deep learning techniques, particularly convolutional

neural networks (CNNs), to achieve highly accurate facial verification and recognition. The algorithm consists of face detection, feature extraction using CNNs, and classification stages. DeepFace was trained on a large dataset and demonstrated impressive results in facial recognition tasks. However, it has limitations, including scalability due to resource requirements, potential bias and privacy concerns, sensitivity to lighting and pose variations, and ethical considerations surrounding surveillance and privacy. Since its introduction, newer facial recognition approaches have been developed, building on the original Deep-Face algorithm (Ansam A, 2019). DeepFace performs better on frontals, but it struggles with variations in pose, illumination and occlusion.

2.11 MULTI-TASK CASCADED CONVOLUTIONAL

MTCNN was proposed by Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao in 2016. It built upon previous face detection methods and improved accuracy and efficiency. The Cascade classifier approach was introduced by Paul Viola and Michael Jones in 2001, revolutionizing real-time face detection with its fast detection speed and low computational requirements.

- **Face Proposal Network (P-Net):** The first stage uses a shallow CNN called P-Net to generate candidate face bounding boxes by sliding a small window over the image and classifying the regions as face or non-face based on their appearance.
- **Refinement Network (R-Net):** The second stage, R-Net, refines the face proposals from P-Net by rejecting false positives and adjusting the bounding box coordinates to better fit the faces. It uses a deeper CNN to further classify the proposals and regress the coordinates.

- **Output Network (O-Net)**: The final stage, O-Net, performs a more accurate localization and facial landmark detection. It refines the bounding box coordinates and predicts facial landmarks, such as eye corners, nose, and mouth, for facial alignment (Yang, et al, 2014).

2.12 ADAVBOOST ALGORITHM AND CASCADE CLASSIFIER

AdaBoost (Adaptive Boosting) is a machine learning algorithm that combines multiple weak classifiers to create a strong classifier. It iteratively trains the weak classifiers on weighted training data, assigning higher weights to misclassified samples. The final classification is determined by a weighted majority vote of the weak classifiers. AdaBoost has been widely used in various tasks, such as face detection and object recognition, due to its ability to handle complex classification problems. Cascade classifier, inspired by AdaBoost, is a technique used in object detection, particularly in the Viola-Jones face detection algorithm. It consists of a series of stages, each comprising multiple weak classifiers. The cascade structure allows for fast detection by quickly discarding non-face regions based on early stages, while more computationally expensive stages are applied only to potential face regions according to Mehmood K and Ahmed B in 2013 implementation of face recognition system using Adboost, This results in high detection rates with low false positives.

AdaBoost was proposed by Yoav Freund and Robert Schapire in 1995. It gained attention for its robustness and effectiveness in handling challenging classification tasks. The cascade classifier concept was introduced by Viola and Jones in 2001, applying AdaBoost to achieve real-time face detection. Since then, AdaBoost and the cascade classifier have become widely used techniques in the field of computer vision and object detection.

2.13 LINEAR DESCRIMINANT ANALYSIS

LDA was originally proposed by Ronald A. Fisher in 1936 as a statistical technique for classification and discriminant analysis. It has since been widely used in pattern recognition and machine learning. Linear Discriminant Analysis (LDA) is a supervised learning algorithm used in face detection and recognition tasks. It aims to find a linear projection that maximizes the separation between different classes while minimizing within-class variation. LDA can enhance the reparability of face and non-face regions in face detection. For face recognition, it finds a subspace that maximizes the separation between individuals, enabling efficient recognition. LDA reduces dimensionality while preserving discriminative information, improving efficiency. However, it may not handle complex variations in face images well. LDA is often used in combination with other algorithms and is based on statistical assumptions. Advanced techniques may be required for complex face variations.

2.14 SUPPORT VECTOR MACHINE (SVM) CLASSIFIER

SVMs were introduced by Vladimir Vapnik and Alexey Chervonenkis in the 1990s. They were initially developed for binary classification tasks and gained popularity due to their strong theoretical foundation and ability to handle high-dimensional data. It finds an optimal hyperplane that separates data points of different classes with the largest possible margin. SVM utilizes support vectors, the closest data points to the hyperplane, for defining the decision boundary. It can handle nonlinear problems through the kernel trick, which transforms the data into a higher-dimensional space. The regularization parameter (C) controls the balance between maximizing the margin and minimizing misclassifications. SVM can be extended to regression tasks as Support Vector Regression (SVR). It is a versatile algorithm with good generalization performance, although the choice of kernel function and regularization parameter

affects its performance (Li S et al, 2009). SVM works well in various domains but can be computationally expensive for large datasets. The time complexity for SVM is given as $O(n^3)$. SVM may be sensitive to the choice of kernel function and its hyperparameters. It can also be computationally demanding, especially with large datasets.

CHAPTER THREE
ANALYSIS AND DESIGN

3.1 OVERVIEW

The Face Detection and Recognition Time Monitoring System is an efficient solution designed to accurately track employees' time in and out using facial detection and recognition algorithm and technology. The system is built to run on a web based system which will allow monitoring report to be accessed anywhere by the manager or administrator.

The system deploys face detection algorithms to locate and identifies human faces in real-time from live camera feeds, it utilizing the Histogram of Oriented Gradient (HOG) algorithm for face recognition. The system monitors known faces registered in the pre-registered database of faces, and tracks their time in and time out as they are detected and recognized, enabling instant updates for managers and administrators. The face detection and recognition process is fully automated, eliminating the need for physical contact or manual entry, ensuring a hygienic and contactless procedure. The system prioritizes data security and privacy, ensuring that only administrator and manager are given access to the detailed report that is generated by the system, when the system stops running.

3.2 ANALYSIS OF EXISTING SYSTEM

RFID systems are composed of various components that work together to identify, track, and manage items or entities equipped with RFID tags as. These components include;

3.3 INPUT OF THE SYSTEM

- **RFID Tags:** These are small electronic devices containing a unique identification number or data. When exposed to an RFID reader's radio waves, they transmit their data to be captured.

- **Antennas:** RFID antennas emit radio waves to power RFID tags and receive data from them. Antennas are a critical input component in RFID systems.
- **RFID Readers:** RFID readers send signals to the antennas, receive data from RFID tags, and communicate with the processing unit. Readers are responsible for initiating tag communication.

3.4 PROCESSING REQUIREMENT

- **RFID Algorithms:** Various algorithms are used in RFID systems to process the data received from tags. These include anti-collision algorithms to manage multiple tags in the reader's field, encryption algorithms for security, and filtering algorithms to enhance data accuracy.
- **Data Processing:** The RFID reader processes the data received from tags, which may include decoding, error checking, and data consolidation. Processing may involve identifying which tags are in the reader's field and handling data collisions.
- **Database Integration:** RFID systems often integrate with databases to store and manage tag data efficiently. Data from RFID tags is processed and stored in databases for further analysis and tracking.

3.5 OUTPUT OF THE SYSTEM

- **Tag Identification Data:** The primary output of an RFID system is the identification data obtained from RFID tags. This data consist of unique serial numbers and more extensive information on the tag.

3.6 STUDY OF PEACE PEAK ACADEMY

Radio Frequency Identity (RFID) is implemented at Peace Peak academy in Saki west Local government of Oyo state, Nigeria to track the check-in time of individuals (Staff and Student) in the school premises and the following areas of concern;

a. Privacy Concerns:

- RFID monitoring systems raise significant privacy concerns among students, and staff. The continuous tracking of individual could lead to discomfort.
- The data collected through RFID tags could potentially be misused or breached, compromising personal information.

b. Health and Safety Worries:

- Stakeholders raised concerns about the potential health effects of constant exposure to RFID signals. Although RFID technology operates at low power levels, doubts about long-term health implications could arise.

c. Technical Glitches:

- Like any technology, RFID systems are susceptible to technical failures. Malfunctioning RFID readers or tags could result in inaccurate tracking, causing confusion and disruptions in the school's or organizations operations.
- System downtime due to technical glitches could lead to wasted time for individual working in the organization.

d. Cost and Implementation Challenges:

- Implementing an RFID monitoring system requires a significant upfront investment in terms of hardware, software, and training. This can strain the budget and resources.
- Integrating the system with existing infrastructure and databases might pose challenges, leading to delays and additional costs.

e. Maintenance and Sustainability:

- RFID systems require ongoing maintenance to ensure smooth operation. This includes replacing malfunctioning tags or readers, updating software, and troubleshooting issues.

- Over time, the costs associated with maintenance can accumulate and strain the school's financial resources.

f. Lack of Flexibility:

- RFID systems, once implemented, might lack the flexibility to adapt to changing needs and circumstances. The fixed infrastructure could become obsolete as technology advances, necessitating further investments for updates.

3.7 THE PROPOSED SYSTEM

The Face Detection and Recognition Time Monitoring System begins by registering a set of faces with their corresponding Face IDs, When the system is operational, a camera is activated. As individuals pass in front of the camera, it captures a frame that includes their face. The system then proceeds to compare the facial geometry of the captured face with the faces stored in the database. If a match is found between the captured face and a face in the database, the system records both the identity of the individual and the timestamp of the event. This information is then stored in a CSV file for future reference and analysis.

3.8 INPUT REQUIREMENT

- Image or Video Source: The system requires input images or video frames containing human faces. These will be sourced from the Camera device.
- Data Format: The input data format will be compatible with face detection and recognition such as JPEG, PNG or video format.
- Quality and Resolution: For accurate face detection and recognition, the input images or frames should have sufficient quality and resolution to capture facial features clearly.

3.9 PROCESSING REQUIREMENT

- **Face Detection**: The system utilises algorithms to detect faces within the input video frames. This process identifies the location and size of faces, often represented by bounding boxes.
- **Face Recognition:** Once the faces are detected, the system uses face recognition algorithms to identify or verify the detected faces against a known database of faces. This process usually involves feature extraction and comparison or face matching.
- **Time Monitoring**: The system should include mechanisms to track the time taken for face detection and recognition processes. This involves time stamping.

3.10 OUTPUT REQUIREMENT.

- **Detected Faces:** The resulting output of the system include gray scale image and face geometry.
- **Recognized Individuals:** For recognized individuals, the output includes the name or identity of the person matched in the database.
- **Real-Time Monitoring:** The system record the time for recognition of the individual in the database of faces.
- **Logging and Reporting**: The system reports logs recognized faces to a CSV file where it can be accessed at any given time by system administrator for future references.

3.11 ARCHITECTURAL DESIGN

The system requires many components to work together as seen bellow;

- **Input Source:** The system starts by obtaining input data, which can be images or video frames containing human faces. This input can come from a camera source.
- **Pre-processing:** Before performing face detection and recognition, the input data will undergo pre-processing steps to enhance the quality of the images or frames. Pre-processing can involve tasks like image resizing, normalization, and noise reduction.
- **Face Detection Module:** The first key component is the face detection module, which uses computer vision algorithms to identify and locate faces within the input images or video frames. The output of this module includes the coordinates of the detected faces, typically represented as bounding boxes or facial landmarks.
- **Database:** The face recognition module relies on a database that stores facial features of known individuals. This database is used as a reference for matching detected faces with known identities.
- **Face Recognition Module:** Once faces are detected, the system employs face recognition algorithms to match the detected faces with a list of known faces in the database. The face recognition module extracts facial features from the detected faces and performs a similarity comparison to identify or verify individuals. The output includes the identity of the recognized individual if a match is found, or indicates that the face is known if known face is found.
- **Time Monitoring Component**: This component tracks the time taken for the face recognition processes. It records timestamps for the time the face it has detected using the OS timestamp.

- **Reporting:** The system report relevant information, such as timestamps of detected and recognized faces to a CSV file per time. This data is useful for monitoring system performance, auditing, and generating reports.

  See figure 3.1

3.12 USE CASE DIAGRAM

Table 3.1 USE CASE SYMBOLS USED

The definition of the symbols used are depicted in **Table 3.1 on Appendix A**

Figure 3.1 USE CASE DIAGRAM

The use Case diagram is depicted in **Figure 3.1 on Appendix B**

3.13 DATABASE DESIGN

Folder-Based Database will be deployed. A folder-based face image database is a straightforward storage solution for managing and organizing face images using a file system. In this setup, each face image is saved as an individual file within a designated folder. This folder serves as the "database," providing a structured way to store and retrieve face image data.

3.14 HOG ALGORITHM

Code section is captured in the A**ppendix B**, the following are the reasons for using the HOG algorithm;

- Robustness to Lighting Conditions: HOG features are based on the gradient of pixel intensities, making them relatively robust to changes in lighting conditions. This makes HOG effective in scenarios where lighting can vary significantly.
- Simplicity and Efficiency: HOG is computationally efficient and relatively simple to implement. It doesn't require complex deep learning models and can provide reasonably good results with less computational overhead.

- Localized Feature Representation: HOG captures local image patterns by considering the orientation of edges and gradients. This localized feature representation makes it suitable for capturing facial features like edges, contours, and shapes.
- Invariance to Small Deformations: HOG features can be made partially invariant to small translations and deformations. This is useful in handling variations in facial expressions and head poses.
- Effective for Pedestrian Detection: While HOG is widely known for pedestrian detection, its principles can also be applied to face detection. It has a proven track record in detecting objects and patterns within images.
- Training Data Efficiency: HOG-based detectors can perform well with relatively smaller amounts of training data compared to deep learning approaches, which often require large datasets for training.
- Interpretability: HOG features are interpretable, which means it's easier to understand and visualize what the algorithm is capturing in an image, making it useful in some applications where interpretability is essential.

3.15 PROGRAM PSEUDOCODE

The program pseudo code is on **Appendix A**

CHAPTER FOUR
IMPLEMENTATION, TESTING AND INTEGRATION

4.0 IMPLEMENTATION

In this chapter, we will translate the pseudo code from the previous chapter into Python code and compile it. This step is crucial to verify that our proposed system meets all requirements and objectives while assessing its performance.

4.1 SOFTWARE AND DEPENDENCIES

- **Tkinter:** Tkinter is a Python library or frame-work used for creating graphical user interfaces (GUIs). It is valuable for developing a user-friendly interface for our proposed sytem, (face detection and recognition system). We are using the version of Tkinter associated with the version of python used.
- **OpenCV (Open Source Computer Vision Library):** OpenCV is a widely-used library for computer vision tasks, including face detection. It provides pre-trained models for face detection and various image processing functions. And we deploy version 4.7.3.
- **Dlib:** It's a C++ library often used for facial landmark detection and face recognition. It has Python bindings and provides tools for face detection and shape prediction, the version used for the project is 19.24.2..
- **face_recognition:** This high-level library, built on top of Dlib, is designed specifically for face recognition tasks. It simplifies the process of face encoding and recognition, the version used is version1.3.0.
- **NumPy:** NumPy is essential for handling and manipulating numerical data, which is common in image processing and deep learning. Installed for the project is version1.25.2.
- **CMake:** Cmake helps build and compile all our libraries together, and the versionof cmake used is version3.27.1.If you're using Dlib, you'll need CMake for building and compiling the library.
- **Face dataset:** Depending on your specific use case, you may need a face dataset for training and testing your recognition model. The dataset could be in various formats, such as CSV files or image directories.

4.2 HARDWARE CONSIDERATION

- **CPU**: A powerful multicore CPU is essential for system management and orchestration of tasks. A high-end processor like an Intel Core i7.
- **GPU**: A dedicated GPU will be advice able to accelerate deep learning-based face detection and recognition algorithms. Mid-range GPUs like NVIDIA GeForce GTX 1660 or AMD Radeon RX 5600 XT should work well.
- **Memory (RAM):** Aim for at least 8GB of RAM, but 16GB or more would be better for handling large datasets and concurrent tasks efficiently.
- **Storage:** Fast and ample storage, preferably SSDs, for storing the dataset, model weights, and temporary files
- **Network Connectivity**: High-speed network connectivity for keeping the camera together especially while using a wireless camera on a wifi network
- **Power Supply:** A stable and sufficient power supply is necessary to handle the hardware load, including the GPU.

4.3 PYTHON IMPLMENTATION

- SECTION 1

  In this section we examine the system user interface implementation with tkinter;
- **GUI Initialization:** The code initializes a Tkinter window (**root**) for the monitoring system with the title "Smart_Technologies_Monitoring_System" and a dark blue background.
  - **Title Label:**
    - It creates a title label at the top of the window with the text "**SMART_TECH_MONITORING_SYSTEM**" in red and a bold font.
  - **Button Configuration:**
    - The code defines button styles and configurations for a set of buttons that will be added to the GUI.
  - **List of Scripts:**
    - A list of script information is defined, including the script name, script file, and associated image resources. These scripts can be executed by clicking the corresponding buttons on the GUI.

- **Button Creation:**
  - The code dynamically creates buttons based on the script information and adds them to the GUI.
  - Each button has a label, an associated image icon, and a command that runs a Python script when clicked.
  - The last button is configured to exit the application when clicked.
- **Button Grid Layout:**
  - Buttons are organized in a grid layout with three columns, and the number of rows depends on the number of buttons.
  - Buttons are evenly spaced with padding.
- **Button Actions:**
  - When a script-related button is clicked, it runs the corresponding Python script using the **sub-process.run** function.
  - The last button (Script 6) is configured to close the application when clicked.
  - An additional button (Script 3) is configured to open a CSV file in Microsoft Excel.
- **Main Loop:**
  - The script enters the Tkinter main event loop (**root.mainloop()**) to handle user interactions and keep the GUI running.

SECTION 2

Here we have our faces (Dataset registered)

I. Imports:

- The script imports necessary libraries:
- **os**: For interacting with the operating system.
- **Tkinter**: For creating the GUI.
- **filedialog** from Tkinter: For opening a file dialog to select an image.
- **PIL** (Python Imaging Library): For image-related operations.

II. select_image Function:

- This function is called when the "Select Image" button is clicked.

- It opens a file dialog (**filedialog.askopenfilename**) that allows the user to choose an image file with extensions **.jpg**, **.jpeg**, or **.png**.
- The selected image's file path is stored in the **selected_image_path** variable.
- The label **selected_image_label** is updated to display the name of the selected image.

III. add_image Function:

- This function is called when the "Add Image" button is clicked.
- It first checks if an image has been selected (**selected_image_path** is not empty).
- It then retrieves the desired image name from the **image_name_entry** widget.
- If an image name is provided, it reads the binary data of the selected image file and saves it to a specified folder with the provided image name.
- The result of the operation is displayed in the **result_label**.

IV. Folder Path:

- The variable **folder_path** specifies the folder where images will be saved. You should replace it with the actual path where you want to store the images.

V. Tkinter Setup:

- The script creates a Tkinter window (**root**) with the title "Register below."
- It sets up a main frame (**frame**) to fill the entire screen for a clean layout.

VI. GUI Elements:

- **selected_image_label**: Displays the name of the selected image.
- **select_button**: Allows the user to select an image file.
- **image_name_label**: Label for entering the desired image name.
- **image_name_entry**: Entry widget for entering the image name.
- **add_button**: Button for adding the selected image to the specified folder.
- **result_label**: Displays the result or status of the image addition process.

VII. Main Loop:

- The script enters the Tkinter main event loop (**root.mainloop()**) to handle user interactions and keep the GUI running.

SECTION 3

Input source is initiated and started;

i. **Importing Libraries:**

- The script starts by importing the necessary libraries, including OpenCV (**cv2**), NumPy (**numpy**), **face_recognition**, **os**, and **datetime**.

ii. **Image Path and Initialization:**

- **path** is set to the directory path where images of known faces are stored.
- An empty list **images** and **classNames** are initialized to store known face images and their corresponding class names (person names).

iii. **Loading Known Faces:**

- The script lists the files in the specified **path** using **os.listdir**.
- It iterates through each image file (**cl**) in the directory, loads the image using **cv2.imread**, and appends it to the **images** list.
- The base filename (without extension) is extracted and appended to the **classNames** list.

iv. **Encoding Known Faces:**

- The **findEncodings** function is defined to encode the known face images in the **images** list using **face_recognition.face_encodings**.
- Encoded face data is stored in the **encodeListKnown** variable.

v. **Recording Timestamps:**

- A function **recordtime** is defined to record timestamps in a CSV file (**Report.csv**) whenever a recognized face is detected.
- It reads existing entries from the CSV file and checks if the name of the recognized person is not already in the list.
- If the name is not in the list, it records the current timestamp and the person's name in the CSV file.

vi. **Video Capture Setup:**

- Video capture is initiated using **cv2.VideoCapture(0)** to capture video from the default camera (index 0).
- A loop is started to continuously process frames from the video stream.

vii. Face Recognition Loop:

- Inside the loop, each frame is captured using **cap.read()**.
- The captured frame is resized and converted to RGB format.
- **face_recognition.face_locations** is used to locate faces in the frame.
- **face_recognition.face_encodings** extracts encodings for the detected faces in the current frame.
- For each detected face, the script compares the face encodings with the known face encodings (**encodeListKnown**) to find matches.
- If a match is found, the person's name is determined using the **classNames** list.
- A rectangle is drawn around the recognized face, and the person's name is displayed on the frame.
- The **recordtime** function is called to record the timestamp for the recognized person.

viii. Displaying the Result:

- The processed frame with recognized faces is displayed using **cv2.imshow**.

ix. Exit Condition:

- The loop continues until the '1' key is pressed (**cv2.waitKey(1) & 0xFF == ord('1')**), at which point the script breaks out of the loop and exits.

4.4 INTEGRATION

All component of the system are coupled together as the code is executed with python compiler firstly with the default system camera, secondly with an external camera, externally installed as the source of input. See figures in the Appendix.

4.5 TESTING

All the component of the system is tested to ensure they meet the requirement.

- The Camera is tested to ensure it functionality.
- The whole system integrated is done and tested to ascertain the objective of the system is met.

4.6 USER INSTRUCTION (GUIDE)

- Start the application by clicking the icon

- Dashboard opens, select the function you wish to perform by clicking on the icons as labeled for the action to be executed.
- Functions available are;
    a. Registering new Dataset (New faces) of individuals.
    b. Connecting a smart-phone to use it camera. (To perform these action, install DroidCam on your Android device, launch it and copy the IP Address to the input on the Droidcam section. Ensure the phone is connected to the same network as the system).
    c. Option to use a connected camera via USB.
    d. Option to use the system webcam
    e. Option to View Report of monitored faces
    f. Option to quit the application.

    Any of the buttons clicked can only be executed once at a time.

CHAPTER FIVE

5.0 SUMMARY

The implemented Face Detection and Recognition Time in Monitoring System, developed using Python, represents a cutting-edge solution that combines computer vision and biometric technology to revolutionize paper works and access control. This system employs sophisticated algorithms to perform two essential functions: face detection and face recognition, all in real-time.

- **Face Recognition:** Once faces are detected, the system employs a custom or pre-trained deep learning model, such as Histogram of Oriented Gradients (HOG), for facial recognition. By comparing the unique facial features and patterns of detected faces against a database of authorized individuals, the system accurately identifies and verifies individuals.
- **Time Tracking:** The system integrates seamlessly with the system time of the host computer, ensuring that it records precise timestamps for each detected face. This feature enables the system to maintain real-time monitoring of individuals as they enter or exit a monitored area, such as a classroom, office, or event venue.
- **Database Management:** A zipped folder based database was deployed for storing the pre-trained dataset of individual.
- **User Interface:** A user-friendly graphical interface is developed using Python libraries like Tkinter enabling users to interact with the system effortlessly. Users can register their faces, view monitored records using Microsoft excel.

5.1 CONCLUSION

The implementation of the Face Detection and Recognition Time in Monitoring System using Python represents a remarkable fusion of cutting-edge technologies, transforming traditional attendance tracking and access control into a seamless, efficient, and secure process. By harnessing the capabilities of computer vision and biometrics, this system can rapidly and accurately detect and recognize faces in real-time. It leverages Python's OpenCV library for robust face detection and deploys Histogram of oriented gradient models for facial recognition, all synchronized with precise timestamp recording through system time integration. The central database management ensures the efficient storage of authorized individuals' data, supporting reporting and analytics for informed decision-making. Furthermore, the user-friendly interface simplifies user interaction with the application.

5.2 RECOMMENDATIONS

- The Application can be maintained and improved subsequently by combining more algorithms to run simultaneously in real time monitoring and a web-based asynchronous system.
- This application can be further integrated with real time database in companies, schools, small organization and event centers.

APPENDIX A

PSEUDO CODE

1. Start
2. Load and pre-process image data
3. Load the input image(s)
4. Convert the image to grayscale
5. Resize the image to a consistent size
6. Calculate HOG features

7. Extract features

8. Calculate gradient magnitude and orientation for pixels

9. Create histograms of oriented gradients in the cell

10. Training Phase (Face Detection):

11. Collect images from database

12. Calculate HOG features for each sample

13.Train an SVM classifier using the positive and negative samples

14. Save the trained SVM model

15. Testing Phase (Face Detection):

16. Open Camera and capture images across

17. For each image Captured:

18. Calculate HOG features for the window

19. Use the trained SVM model to classify whether the images are recognized.

20. Recognition Phase (Face Recognition):

21. For each detected face region:

22. Extract the face region from the original image

23.Pre-process the face region (e.g., resize to a fixed size, normalize pixel values)

24.Feed the pre-processed face image into a face recognition

25. Retrieve the predicted identity label and record the timestamp to a CSV file

26. End

APPENDIX B

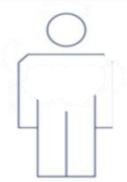
TABLES

Term	Summary of purpose
Face	The concept of "face" encompasses not only its physical aspects but also its psychological, social, cultural dimensions and texture.

Detectionp	detection methods utilize various techniques, such as machine learning, deep learning, feature extraction, and pattern recognition, to analyze visual data and make informed decisions about the presence, location, and identity of object
Recognition	Facial recognition serves various purposes, including identity verification for secure authentication, access control and security enhancement, assistance in law enforcement and public safety, personalization and improved user experience, as well as attendance and time tracking automation. Additionally, it is utilized in border control and immigration to strengthen security and expedite passenger processing. However, responsible and ethical use of facial recognition technology requires careful consideration of privacy, data protection, and potential risks.
Time	Measurable period during which an action, process, or condition exists, or continue. measured in seconds, minutes, hour, e.t.c.

Monitoring	Monitoring involves observing, collecting data, and analyzing information to track, evaluate, and make informed decisions about activities, processes, or systems. Monitoring can include direct observation, data collection and analysis, real-time tracking, or remote monitoring using technology. It is used in various fields to improve performance, enhance safety, and enable informed decision-making.

TABLE 1.1 Definition of Terms Used

Objects	Symbol	Description
Actor		They are the system's users. The actor may be a person, group, or external system. They play a part in how the system works.
Use Case		Use case is a list of steps, typically defining interactions between an actor and a system to achieve a goal.
System		A system is a rectangle spanning all the use cases in the system that defines the scope of your system.

BLE 3.1 Definitions of Symbols Used

APPENDIX C

FIGURES

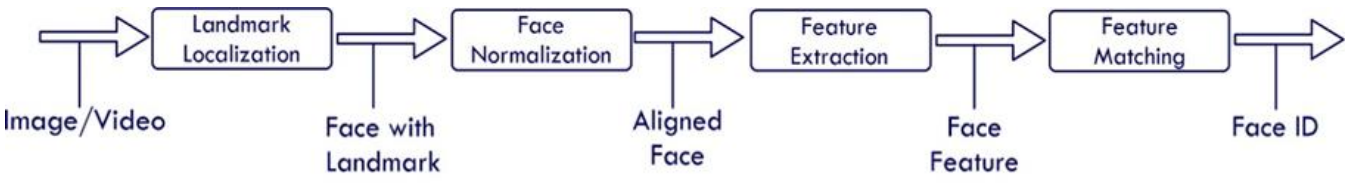


Figure 1.1 Face Recognition Processes

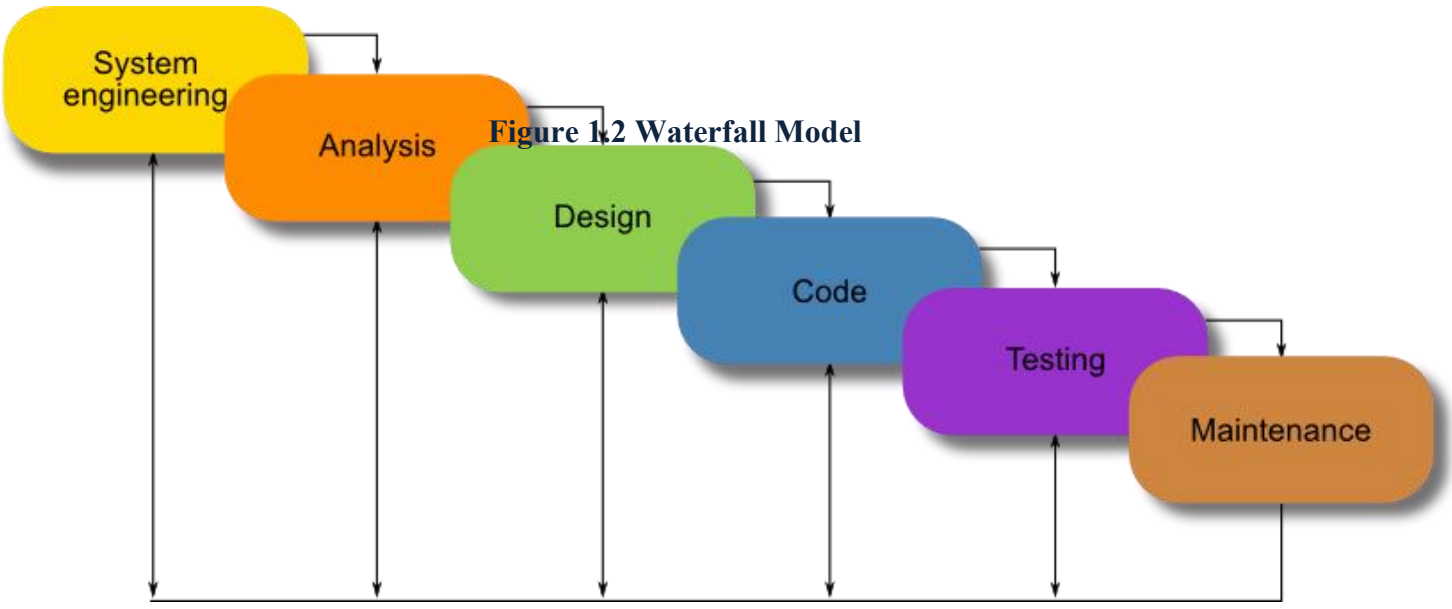


Figure 1.2 Waterfall Model

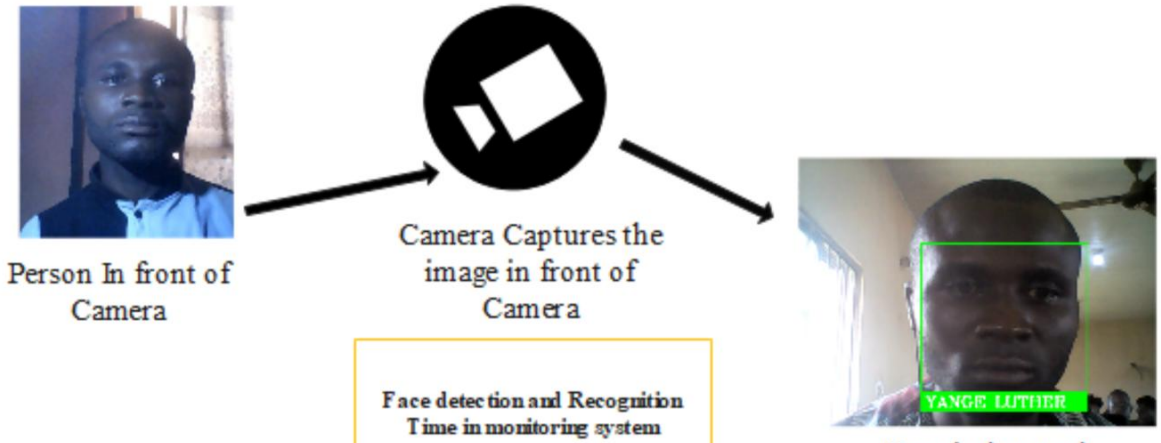


Figure 3.1 System Architecture

APPENDIX D

SOURCE CODES

#MENU/DASHBOARD GUI

```
import tkinter as tk
import subprocess
import sys
def run_script(script_name):
    try:
        subprocess.run(['python', script_name])
    except Exception as e:
        print(f"Error running {script_name}: {e}")
def quit_application():
    root.quit()
    sys.exit()
def open_csv_in_excel():
    csv_file_name = 'Report.csv'
    subprocess.run(['start', csv_file_name], shell=True)
root = tk.Tk()
```

```
root.title("Smart_Technologies_Monitoring_System")

root.configure(bg="#00008B")

title_label = tk.Label(root, text="SMART_TECH_MONITORING_SYSTEM",
font=("Helvetica", 20, "bold"), fg="white", bg="#00008B")

title_label.grid(row=0, column=0, columnspan=3, pady=(10, 5))

dpi = 96

width_inches = 1

height_inches = 1

width_pixels = int(width_inches * dpi)

height_pixels = int(height_inches * dpi)

button_style = {

    "width": width_pixels,

    "height": height_pixels,

    "padx": 90,

    "pady": 60,

    "cursor": "hand2",

    "relief": tk.RAISED,

    "borderwidth": 3,

}

scripts = [

    {"name": "Script 1", "script": "add_images.py", "resources":
"resources/add.png"},

    {"name": "Script 2", "script": "external.py", "resources":
"resources/webcam.png"},

    {"name": "Script 3", "script": "script3.py", "resources":
"resources/reports.png"},
```

```
    {"name": "Script 4", "script": "droid.py", "resources": "resources/droid.png"},
    {"name": "Script 5", "script": "main.py", "resources":
"resources/systemcam.png"},
    {"name": "Script 6", "script": " ", "resources": "resources/quit.png"}
]
buttons = []
for i, script_info in enumerate(scripts):
    script_name = script_info["script"]
    resources = script_info["resources"]
    image = tk.PhotoImage(file=resources)
    button = tk.Button(root, image=image, text=script_info["name"],
compound="top", command=lambda name=script_name: run_script(name))
    button.image = image
    button.config(**button_style)
    buttons.append(button)
    if i == len(scripts) - 1:
        button.config(command=quit_application)
    if i == len(scripts) - 4:
        button.config(command=open_csv_in_excel)
for i, button in enumerate(buttons):
    row_num = i // 3 + 1
    col_num = i % 3
    button.grid(row=row_num, column=col_num, padx=10, pady=10)
root.mainloop()
```

#MAIN PROGRAM SOURCE CODE

```
import cv2
import numpy as np
import face_recognition
import os
from datetime import datetime
path = 'imagesFolder'
images = []
classNames = []
myList = os.listdir(path)
# print(myList)
for cl in myList:
    curImg = cv2.imread(f'{path}/{cl}')
    images.append(curImg)
    classNames.append(os.path.splitext(cl)[0])
# print(classNames)
def findEncodings(imges):
```

```
    encodeList = []
    for img in images:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        encode = face_recognition.face_encodings(img)[0]
        encodeList.append(encode)
    return encodeList
encodeListKnown = findEncodings(images)
# print(len(encodeListKnown))
def recordtime(name):
    with open('Report.csv','r+') as f:
        myDataList = f.readlines()
        nameList = []
        # print(myDataList)
        for line in myDataList:
            entry = line.split(',')
            nameList.append(entry[0])
        if name not in nameList:
            now = datetime.now()
            # todayDate = todayDate.strftime('%y-%m-%d')
            dtString = now.strftime('%Y-%m-%d %H:%M:%S')
            f.writelines(f'\n{name},{dtString}')
encodeListKnown = findEncodings(images)
print('Please wait!')
cap = cv2.VideoCapture(0)
```

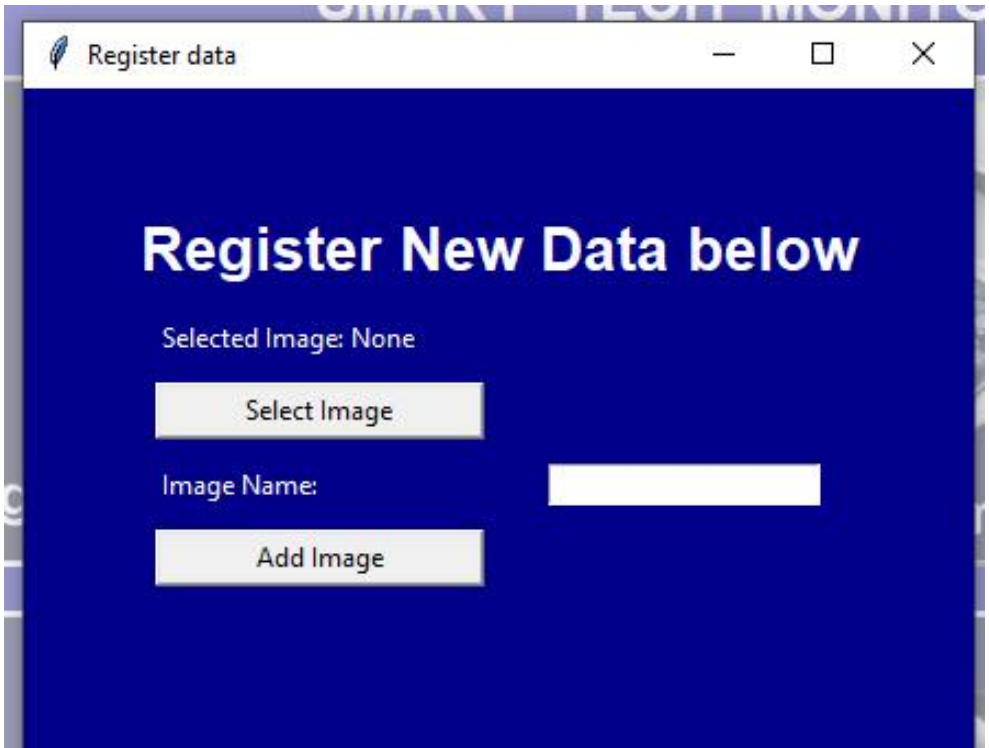
```
while True:
    success, img = cap.read()
    imgS = cv2.resize(img, (0,0), None, 0.25, 0.25)
    imgS = cv2.cvtColor(imgS, cv2.COLOR_BGR2RGB)
    facesCurFrame = face_recognition.face_locations(imgS)
    encodesCurFrame = face_recognition.face_encodings(imgS, facesCurFrame)

    for encodeFace, faceLoc in zip (encodesCurFrame, facesCurFrame):
        matches = face_recognition.compare_faces(encodeListKnown, encodeFace)
        faceDis = face_recognition.face_distance(encodeListKnown,encodeFace)
        # print(faceDis)
        matchIndex = np.argmin(faceDis)
        if matches[matchIndex]:
            name = classNames[matchIndex].upper()
            # print(name)
            y1,x2,y2,x1 = faceLoc
            y1,x2,y2,x1 = y1*4,x2*4,y2*4,x1*4
            cv2.rectangle(img,(x1,y1),(x2,y2), (0,255,0), 2)
            cv2.rectangle(img,(x1,y2-35), (x2,y2), (0,255,0),cv2.FILLED)
            cv2.putText(img, name, (x1+6,y2-6),
cv2.FONT_HERSHEY_COMPLEX,1,(255,255,255),2)
            recordtime(name)
    cv2.imshow('using system_cam hit 1 to close', img)
    if cv2.waitKey(1) & 0xFF == ord('1'):
        break
```

APPENDIX E

SCREENSHOTS


Smart_Technologies_Monitoring_System
SMART_TECH_MONITORING_SYSTEM
Register New Faces
WebCam
Report
DroidCam
SystemCam
Quit

SYSTEM MENU


Register data
Register New Data below
Selected Image: None
Select Image
Image Name:
Add Image

PROMPT TO REGISTER NEW DATA

PROMPT TO USER FOR A DROIDCAM URL

DETECTED AND RECOGNIZED FACE

	A	B	C	D	E	F
1	Report_Sheet					
2	Name	Time				
3	ADETOKUNBO MAYOWA	9/12/2023 10:00				
4	GODSPOWER OGBONA	9/12/2023 10:00				
5	YANGE LUTHER	9/12/2023 10:00				
6	NAOMI EZIUKWU	9/12/2023 10:01				
7						
8						
9						
10						
11						

REPORT OBTAINED


Adetokunbo Mayowa


Endurance


Fancis Pascal


Godspower Ogbona


Naomi Eziukwu


Yange Luther

REGISTERED DATASET

REFERENCES

Chen, D., Ren, S., Wei, Y., Cao, X., & Sun, J. (2014). Joint cascade face detection and alignment. In European Conference on Computer Vision, pp. 109–122.

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Vol. 1, pp. 886–893, June 2005).

Dollar, P., Belongie, S., & Perona, P. (2010). The fastest pedestrian detector in the West. In Proceedings of the British Machine Vision Conference (pp. 68.1–68.11). BMVA Press.

Felzenszwalb, P. F., Girshick, R. B., McAllester, D., & Ramanan, D. (2010). Object detection with discriminatively trained part-based models. IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(9), 1627–1645, September 2010.

Grossi, E., & Buscema, M. (2008). Introduction to artificial neural networks. European Journal of Gastroenterology & Hepatology, 19, 1046–1054, January 2008.

Li, H., Lin, Z., Shen, X., Brandt, J., & Hua, G. (2015). A convolutional neural network cascade for face detection. In IEEE Conference on Computer Vision and Pattern Recognition, pp. 5325–5334.

Liu, Z., Luo, P., Wang, X., & Tang, X. (2015). Deep learning face attributes in the wild. In IEEE International Conference on Computer Vision, pp. 3730–3738.

Luxand Incorporated. (n.d.). Luxand face SDK. http://www.luxand.com/

O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. arXiv: 1511.08458 [cs.NE].

Pattanayak, S. (2019). Intelligent Projects Using Python. Packt Publishing, pp. 1–4.

Phung, & Rhee. (2019). A high-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets. Applied Sciences, 9, 4500, October 2019.

Porikli, F. (2005). Integral histogram: A fast way to extract histograms in Cartesian spaces. IEEE Conference on Computer Vision and Pattern Recognition, 1, 829–836.

Vondrick, C., Khosla, A., Malisiewicz, T., & Torralba, A. (2013). HOGgles: Visualizing Object Detection Features. International Conference on Computer Vision, 1–8.

Xiong, X., & Torre, F. (2013). Supervised descent method and its applications to face alignment. In IEEE Conference on Computer Vision and Pattern Recognition, pp. 532–539.

Yang, S., Luo, P., Loy, C. C., & Tang, X. (2015). WIDER FACE: A Face Detection Benchmark. arXiv preprint arXiv:1511.06523.

Zhang, C., & Zhang, Z. (2014). Improving multiview face detection with multi-task deep convolutional neural networks. IEEE Winter Conference on Applications of Computer Vision, pp. 1036–1041.

Zhang, J., Shan, S., Kan, M., & Chen, X. (2014). Coarse-to-fine auto-encoder networks (CFAN) for real-time face alignment. In European Conference on Computer Vision, pp. 1–16.

Zhang, Z., Luo, P., Loy, C. C., & Tang, X. (2014). Facial landmark detection by deep multi-task learning. In European Conference on Computer Vision, pp. 94–108.