Arab Academy for Science, Technology and Maritime Transport

College of Engineering and Technology

Computer Engineering Department

B. Sc. Final Year Project

”SECURECAMPUS” AN AI-POWERED SURVEILLANCE SYSTEM WITH FACE RECOGNITION

Presented By:

Walid Atef Abdelmaqsoud

Abdelrahman Usama Saad *Omar Wael Omar*

Omar Ezzat Abdelrahim *Mohamed Osama Mohamed*

Supervised By:

Dr. Menna Kamel *Assoc. Prof. Fahima ElMaghraby*

July 2024

DECLARATION

I hereby certify that this report, which I now submit for assessment on the program of study leading to the award of Bachelor of Science in *Computer Engineering*, is all my own work and contains no Plagiarism. By submitting this report, I agree to the following terms:

> *Any text, diagrams or other material copied from other sources (including, but not limited to, books, journals, and the internet) have been clearly acknowledged and cited followed by the reference number used; either in the text or in a footnote/end-note. The details of the used references that are listed at the end of the report are confirming to the referencing style dictated by the final year project template and are, to my knowledge, accurate and complete.*

I have read the sections on referencing and plagiarism in the final year project template.
I understand that plagiarism can lead to a reduced or fail grade, in serious cases, for the Graduation Project course.

Student Name: Walid Atef Abdelmaqsoud

Registration Number: _____

Signed: _____

Student Name: Abdelrahman Usama Saad

Registration Number: _____

Signed: _____

Student Name: Omar Wael Omar

Registration Number: _____

Signed: _____

Student Name: Omar Ezzat Abdelrahim

Registration Number: _____

Signed: _____

Student Name: Mohamed Osama Mohamed

Registration Number: _____

Signed: _____

Date: Monday 15th July, 2024

Dedicated to the Arab Academy for Science, Technology and Maritime Transport, this project represents the culmination of an enriching academic journey. Gratitude extends to the esteemed faculty, especially our dedicated supervisors, Dr. Menna Kamel and Assoc. Prof. Fahima Elmaghraby, whose unwavering guidance, expertise, and support have been instrumental in shaping this endeavor. Their mentorship has not only fueled our intellectual growth but has also inspired a profound appreciation for the pursuit of knowledge. This dedication also extends to our families, friends, and peers, whose encouragement and camaraderie have been invaluable throughout our time at the university. Thank you for being integral parts of this educational voyage.

ACKNOWLEDGMENTS

First and foremost, we express our deepest gratitude to the Almighty God for His guidance, blessings, and strength throughout the course of this project. Without His divine support, this journey would not have been possible.

We extend our heartfelt appreciation to the Arab Academy for Science, Technology and Maritime Transport for providing an exceptional academic environment that has been instrumental in the completion of this project. Special gratitude is reserved for our esteemed supervisors, Dr. Menna Kamel and Assoc. Prof. Fahima Elmaghraby, whose unwavering guidance, mentorship, and insightful feedback have been crucial to the success of this work. Their dedication to our academic growth has been a source of inspiration.

We also want to acknowledge the collaborative spirit within the university community, which has been a driving force behind this accomplishment. Additionally, our sincere thanks go to our friends, family, and peers for their encouragement and understanding during the ups and downs of this academic endeavor.

Finally, a heartfelt acknowledgment to anyone whose influence, support, or expertise has played a role in the completion of this project. Your contributions have not gone unnoticed, and we are truly grateful for the collective effort that has made this achievement possible.

ABSTRACT

SecureCampus is an AI-powered surveillance system designed to meet the increasing security demands in educational institutions, corporations, and public spaces. This project integrates advanced computer vision technologies, including face recognition, behavior detection, and object detection, to provide real-time threat assessment and enhance security measures. Tailored for universities, companies, and malls, *SecureCampus* aims to create safer environments by monitoring and analyzing individual behavior within designated areas.

The system's architecture is designed for efficiency, scalability, and seamless integration. Built on robust machine learning models utilizing **YOLOv10**, **DeepFace**, and **face_recognition**, and leveraging custom datasets processed with **Roboflow**, the system ensures accurate face and behavior recognition. Cameras strategically placed in key locations feed real-time video streams to the backend, where face recognition identifies individuals and behavior detection algorithms analyze potential threats, triggering alerts as needed. **Firebase** is used for efficient storage, database management, and AI model hosting, ensuring scalability and historical data accessibility.

The **face_recognition** library is employed on the login page for user authentication, while **DeepFace** is used to detect students in frames after incidents occur. Detection of violence, inappropriate clothing, and Smoking usage is achieved by fine-tuning **YOLOv10** on custom datasets. The inappropriate clothing detection model is trained on 2,063 photos, augmented to a total of 5,355 images, and achieves an accuracy of 97%. The Smoking detection model, fine-tuned on a dataset of 2,800 images, reaches an accuracy of 86%. Violence detection is also integrated with an accuracy rate of 93.4%.

Ethical considerations, guided by IEEE standards such as **IEEE 2410** and **IEEE 2796**, underscore the transparency, explainability, and fairness in the design and deployment of AI systems. The project aligns with IEEE standards addressing performance benchmarking of machine learning models (**IEEE 2846**) and the evaluation of biometric systems (**IEEE 1838**) to ensure reliability and effectiveness.

SecureCampus represents a comprehensive approach to security, leveraging advanced computer vision technologies within a framework of ethical considerations and industry standards. This abstract provides a glimpse into the project's overview, main aim and objectives, engineering standards, design considerations, constraints, analysis and verification methods, and field of applications, inviting readers to explore further.

TABLE OF CONTENTS

Acknowledgments . . . . . **i**

Abstract . . . . . **ii**

List of Figures . . . . . **viii**

List of Tables . . . . . **ix**

List of Abbreviations . . . . . **x**

1 Introduction **1**

1.1 Motivation 1

1.2 Problem Statement 2

1.3 Objectives 2

1.4 Intended Beneficiaries 3

1.5 Design and Implementation Challenges 4

2 Related Work **5**

2.1 Literature Review 5

2.1.1 Face Recognition Technologies 5

Cascaded P-RBM Approach 5

Face Matching Approach 5

Face Detection in Security Monitoring Based on Artificial Intelligence Video Retrieval Technology Approach 6

2.1.2 Behavior Detection in Surveillance 6

YOLOv4 on KISA Dataset Approach 6

Real-Time Surveillance System for Analyzing Abnormal Behavior of Pedestrians Approach 7

Real-Time Video Violence Detection Using CNN Approach 7

YOLO-based Real-Time Violence Detection System for Smart Surveillance Approach 7

Deep Learning-Based Methods for Anomaly Detection in Video Surveillance Approach 8

InceptionV3 Model on UCF-Crime Dataset Approach 8

2.1.3 Object Detection 8

Real-Time Object Detection using Deep Learning and OpenCV Approach 8

Distributed Real-Time Object Detection Based on Edge-Cloud Collaboration for Smart Video Surveillance Applications Approach 9
YOLO-v1 to YOLO-v8 Overview Approach 9
Implementation of Personal Protective Equipment Detection Using Django and YOLO Web Approach 10
2.2 Background 10
2.2.1 Face Detection and Recognition 10
Face Recognition with DeepFace 10
Face Recognition with face_recognition 12
2.2.2 Clothes Detection 12
Architecture of YOLOv10 12
2.2.3 Roboflow 15
Overview 16
Benefits 16
Applications 17
2.2.4 Web Application 17
Flask Framework 17
FastAPI Framework 19
2.2.5 Database 21
Introduction to Firebase 21
Firebase Configuration 22
Firestore and Realtime Database 23
Storage for Photos and Recordings 24
2.2.6 Deployment 25
Microservices Architecture 25
Application Programming Interfaces (APIs) 26
Docker Containers 26
Single-Container Deployment with Docker Compose 28
Kubernetes for Container Orchestration 29
2.2.7 Ethical Considerations and Standards 32

3 System Design and Architecture 34
3.1 Proposed Model 34
3.1.1 Components Descriptions and Project Flow 34
3.2 System Architecture Diagram 35
3.2.1 Detailed Components Description 36
3.2.2 Data Flow 37
3.3 User Roles and Permissions 37
3.3.1 Admin Role 37

3.3.2 Normal User Role . . . 38
3.3.3 Access Levels . . . 38

4 Implementation . . . 39
4.1 Technologies and Tools . . . 39
4.1.1 Hardware and Software Components . . . 39
Hardware . . . 39
Software and Libraries . . . 39
AI Technologies Used . . . 40
4.2 Models . . . 42
4.2.1 Face Detection and Recognition . . . 42
Face Recognition with DeepFace . . . 42
Face Recognition with Face_Recognition library . . . 42
4.2.2 Inappropriate Clothing Detection . . . 43
Dataset Collection . . . 43
Data Augmentation . . . 44
Model Training . . . 44
Conclusion . . . 45
4.2.3 Violence Detection . . . 45
Introduction . . . 45
Methodology . . . 45
Dataset and Preprocessing . . . 45
Model Implementation . . . 46
Performance Evaluation . . . 46
Conclusion . . . 47
4.2.4 Smoking Detection . . . 47
Dataset Collection . . . 47
Data Annotation . . . 47
Data Preprocessing . . . 47
Model Architecture and Configuration . . . 48
Training Procedure . . . 48
Evaluation and Fine-Tuning . . . 48
4.3 Web Application Development . . . 49
4.3.1 Backend Development . . . 49
System Backend Implementation . . . 49
Tools and Technologies Used . . . 49
Backend Workflow . . . 49
4.3.2 Frontend Development . . . 50
HTML . . . 51

CSS . . . 51
JavaScript . . . 51
Bootstrap . . . 51
Features and Navigation . . . 51
Interaction with Backend . . . 52
4.3.3 Features and Navigation . . . 52
Home . . . 52
About Us . . . 53
Demo . . . 54
Contact Us . . . 54
Documentation . . . 55
Live . . . 56
Recordings . . . 56
Reports . . . 57
Users . . . 57
Admin . . . 58
4.4 Database and Storage . . . 59
4.4.1 Firestore and Realtime Database . . . 59
Firestore . . . 59
Realtime Database . . . 59
Integration and Workflow . . . 59
4.4.2 Storage for Photos and Recordings . . . 59
Integration and Workflow . . . 60
4.5 Deployment . . . 60
4.5.1 Microservices Implementation Details . . . 60
Face Detection Service . . . 60
Violence Detection Service . . . 61
Inappropriate Clothing Detection Service . . . 61
Email Notification Service . . . 61
Database Service . . . 61
Main Application Service (Backend and Frontend) . . . 61
4.5.2 Implementing APIs Containerization with Docker . . . 62
Steps to Create Docker Images for Each API . . . 62
Benefits of Containerizing the APIs . . . 62
4.5.3 Implementing Kubernetes . . . 63
Setting Up a Local Kubernetes Cluster . . . 63
Steps to Set Up a Local Kubernetes Cluster Using Kubeadm . . . 63
Deploying Containers in Kubernetes . . . 64
Creating Kubernetes Manifests . . . 64

Managing Kubernetes Objects . . . 64
4.5.4 Benefits of Kubernetes for our Surveillance System . . . 65
Load Balancing . . . 65
Scalability . . . 65
Conclusion . . . 65

5 Results and Findings . . . 66
5.1 Object Detection . . . 66
5.1.1 Approach . . . 66
5.1.2 Results . . . 66
5.2 Face Recognition . . . 69
5.2.1 Approach . . . 69
5.2.2 Results . . . 69
5.3 Violence Detection . . . 71
5.3.1 Approach . . . 71
5.3.2 Results . . . 71
5.4 Additional Features and Integration . . . 73
5.4.1 Face Detection and Recognition . . . 73
5.4.2 Smoking Detection . . . 73
5.4.3 Web Application and Workflow . . . 73
5.4.4 Workflow . . . 74

6 Conclusion and Future Work . . . 75
6.1 Conclusion . . . 75
6.2 Future Work . . . 76

References . . . 77

LIST OF FIGURES

2.1 Deepface 10
2.2 VGG-Face Model Architecture 11
2.3 Ultralytics 13
2.4 YOLO ARCHITECTURE 14
2.5 Roboflow Workflow 16
2.6 Monolithic VS Microservices Approach 26
2.7 Containers Vs Virtual Machines 27
2.8 Kubernetes Architecture Diagram 29

3.1 Proposed Model 34
3.2 System Architecture 35

4.1 Home Page 53
4.2 About Us Page 53
4.3 Login Page 54
4.4 Contact Us Page 54
4.5 Documentation Page 55
4.6 Live Page 56
4.7 Recordings Page 57
4.8 Reports Page 57
4.9 Users Page 58
4.10 Admin Page 58

5.1 Object Detection Confusion Matrix 67
5.2 Object Detection Acc and Loss Curves 67
5.3 Object Detection Test Set 68
5.4 face_recognition results 70
5.5 deepface results 70
5.6 Violence Confusion Matrix 71
5.7 Violence detection results 72

LIST OF TABLES

4.1 Violence Detection Performance Metrics 46

5.1 Object Detection Results. 68
5.2 Violence Results. 72
5.3 Smoking Detection Results. 73

LIST OF ABBREVIATIONS

AI	-	Artificial Intelligence
AP	-	Average Precision
API	-	Application Programming Interface
AUC	-	Area Under the Curve
AWS	-	Amazon Web Services
CAD	-	Computer-Aided Design
CCTV	-	Closed-Circuit Television
CNN	-	Convolution Neural Network
CPU	-	Central Processing Unit
CSS	-	Cascading Style Sheets
CSP	-	Cross Stage Partial Network
DNN	-	Deep Neural Network
dlib	-	A machine learning library
FaaS	-	Functions as a Service
Faster R-CNN	-	Faster Region-based Convolutional Neural Network
Fig	-	Figure
FPGA	-	Field-Programmable Gate Array
GCP	-	Google Cloud Platform
GCS	-	Google Cloud Service
GDPR	-	General Data Protection Regulation
GPU	-	Graphics Processing Unit
GUI	-	Graphical User Interface
HOG	-	Histogram of Oriented Gradients
HTML	-	HyperText Markup Language
HTTP	-	Hypertext Transfer Protocol
HTTPS	-	Hypertext Transfer Protocol Secure
IaaS	-	Infrastructure as a Service
IEEE	-	Institute of Electrical and Electronics Engineers
IDE	-	Integrated Development Environment
IoT	-	Internet of Things
IoU	-	Intersection over Union
IP	-	Internet Protocol
JSON	-	JavaScript Object Notation
JS	-	JavaScript
K8s	-	Kubernetes
KNN	-	K-Nearest Neighbors
LDA	-	Linear Discriminant Analysis
ML	-	Machine Learning
MLP	-	Multi-Layer Perceptron (a type of neural network)
mAP	-	Mean Average Precision
mAP-50	-	Mean Average Precision at 50%
MS	-	Microsoft
NLP	-	Natural Language Processing
NMS	-	Non-Maximum Suppression
OCR	-	Optical Character Recognition
OpenCV	-	Open Source Computer Vision
PaaS	-	Platform as a Service
PCA	-	Principal Component Analysis
PC	-	Personal Computer
P-RBM	-	Product of Restricted Boltzmann Machines
PT	-	PyTorch (another popular machine learning framework)
RAM	-	Random Access Memory

R-CNN	-	Region-based Convolutional Neural Network
ReLU	-	Rectified Linear Unit
RFID	-	Radio-Frequency Identification
RL	-	Reinforcement Learning
RMSprop	-	Root Mean Square Propagation
ROC	-	Receiver Operating Characteristic
ROC-AUC	-	Receiver Operating Characteristic - Area Under the Curve
ROI	-	Region of Interest
SDK	-	Software Development Kit
SaaS	-	Software as a Service
SGD	-	Stochastic Gradient Descent
SOTA	-	State of the Art
SQL	-	Structured Query Language
SSD	-	Single Shot Multibox Detector
SVR	-	Support Vector Regression
SVM	-	Support Vector Machine
TF	-	TensorFlow (a popular machine learning framework)
TF-IDF	-	Term Frequency-Inverse Document Frequency
TPU	-	Tensor Processing Unit
UI	-	User Interface
UI/UX	-	User Interface/User Experience
VGG	-	Visual Geometry Group
VLC	-	VideoLAN Client
WLAN	-	Wireless Local Area Network
YOLO	-	You Only Look Once
YOLOv10	-	You Only Look Once, Version 10

Chapter One

1 INTRODUCTION

In today's educational institutions, corporations, and public spaces, ensuring security and maintaining order is increasingly challenging due to high population densities and the complexity of monitoring multiple areas simultaneously. Universities, for example, often experience overcrowding, making it difficult for security personnel to oversee every individual effectively. This situation is exacerbated in large settings such as cafeterias, libraries, and common areas where incidents can occur without immediate detection.

Violence and altercations among individuals, particularly in academic settings, are significant concerns. Frequent incidents involving physical altercations between students highlight the need for advanced surveillance systems capable of detecting and reporting such events in real-time as stated in [1] in the references. Many universities face recurrent issues during student affairs meetings, which often revolve around conflicts and fights, underscoring the need for proactive measures to prevent such incidents.

Additionally, universities may have strict dress code policies to maintain a professional and respectful environment. For instance, prohibitions on attire like shorts, crop tops, skirts, and ripped jeans are common in many institutions. Ensuring compliance with these dress code restrictions can be challenging for security staff, especially in large or crowded areas.

1.1 MOTIVATION

The motivation behind the development of the SecureCampus surveillance system is rooted in addressing these specific challenges. By integrating advanced computer vision technologies, such as face recognition, behavior detection, and object detection, SecureCampus aims to enhance the ability of security personnel to monitor and respond to incidents effectively.

The system provides several key benefits:

- **Improved Surveillance Coverage**: With the ability to analyze real-time video feeds from multiple cameras, the system ensures comprehensive monitoring, reducing the likelihood of missed incidents.
- **Real-Time Threat Detection**: Advanced algorithms can identify violent behaviors and trigger alerts promptly, allowing for rapid intervention and prevention of potential conflicts.
- **Automated Dress Code Enforcement**: By employing object detection to monitor compliance with dress code policies, the system assists in maintaining uniform standards without requiring constant manual oversight.

Furthermore, the project responds to the growing demand for intelligent surveillance solutions that integrate seamlessly into existing infrastructure. By leveraging robust machine learning models and cloud-based technologies, SecureCampus not only addresses immediate security concerns but also contributes to long-term improvements in campus and public space safety.

In summary, SecureCampus is driven by the need to overcome current limitations in surveillance and security, providing a sophisticated, scalable solution that enhances safety and operational efficiency in various settings.

1.2 PROBLEM STATEMENT

Current security systems often lack an integrated approach that combines face recognition, behavior detection, and object detection, creating a significant gap in real-time threat assessment capabilities. Traditional systems typically operate in isolation, making it difficult for security personnel to monitor and manage multiple types of threats simultaneously. Additionally, inefficient and non-intuitive user interfaces can impede effective surveillance and response.

Furthermore, ethical considerations in the deployment of AI technologies are frequently overlooked. This oversight can lead to privacy concerns and issues related to the transparency and fairness of surveillance practices. Many existing systems do not adequately address these ethical aspects, resulting in potential misuse or inadequate protection of individual rights.

Moreover, conventional surveillance systems struggle to adapt to the evolving and dynamic nature of security challenges. As threats and security needs change, there is a pressing need for a more adaptable and comprehensive solution that integrates advanced technologies and maintains a strong ethical framework.

The *SecureCampus* project is designed to address these shortcomings by offering an integrated, AI-powered surveillance system that combines face recognition, behavior detection, and object detection in a cohesive manner. It aims to improve real-time threat assessment, enhance user interface efficiency, and ensure ethical AI deployment, thereby redefining security standards and meeting the evolving needs of modern environments.

1.3 OBJECTIVES

• System Development:
Design and implement the AI-powered surveillance system with face recognition, behavior detection, and object detection.

• Technology Integration:

Integrate multiple Computer Vision models Together for accurate and efficient surveillance capabilities.

• User-Friendly Interface:
Develop a straightforward application interface for seamless management by security personnel.

• Ethical Guidelines:
Adhere to ethical guidelines and IEEE standards for transparency, fairness, and privacy.

• Testing and Optimization:
Rigorously test and optimize the system to ensure reliable real-time threat assessment.

• Scalability and Accessibility:
Use cloud services for scalable storage, database management, and AI model hosting.

• Compliance Features:
Include features like inappropriate clothing recognition to ensure compliance with institutional regulations.

1.4 INTENDED BENEFICIARIES

SecureCampus is designed to benefit a diverse array of stakeholders, including:

• Educational Institutions:
Universities and schools will gain enhanced security infrastructure, contributing to a safer environment for students and staff. The system's capabilities, such as face recognition and behavior detection, will help in monitoring and managing security effectively.

• Corporations and Workplaces:
Businesses and corporate environments will experience improved security measures, protecting employees, assets, and sensitive information from potential threats. The system's real-time threat assessment and object detection features will ensure a secure workplace.

• Public Spaces:
Malls, event venues, and other public spaces will benefit from increased safety for visitors and participants. The system's ability to detect inappropriate behavior and enforce dress code policies will contribute to a safer and more orderly environment.

• Security Personnel:
Security guards and staff will be empowered with an efficient, user-friendly application designed for easy management of surveillance outputs. The system's intuitive interface will simplify tasks such as monitoring live feeds, reviewing recordings, and responding to alerts.

• Regulatory Authorities:

Regulatory bodies can utilize the system to enforce compliance with institutional regulations and dress code policies. Features such as inappropriate clothing recognition will aid in maintaining adherence to set guidelines.

• Technology Enthusiasts and Developers:
Technology professionals and developers will find value in *SecureCampus* as it demonstrates the integration of advanced technologies, including AI and computer vision, into practical applications. The project serves as a comprehensive example of leveraging these technologies for real-world solutions.

Ultimately, *SecureCampus* aims to contribute to the overall safety and security of society by deploying an ethical and effective AI-powered surveillance system that addresses contemporary security challenges.

1.5 DESIGN AND IMPLEMENTATION CHALLENGES

1. High Computational Demands for Real-Time Face Recognition:
Face recognition requires significant GPU resources for real-time processing, which was a challenge as typical student devices lack the necessary hardware power. This led to performance issues such as stuttering camera feeds when attempting to run face recognition live.

2. Limited Access to University-Specific Datasets:
Due to privacy concerns, obtaining a comprehensive dataset of university students for testing was not feasible. This limitation meant that our testing was restricted to dummy data, which may not fully represent real-world scenarios.

3. Complexity of Visualization and Model Integration:
The integration and visualization of multiple models on a single device proved to be cumbersome and inefficient, necessitating a more sophisticated approach to manage the deployment.

4. Resource Limitations for Model Deployment:
Budget and hardware constraints impacted the selection of resources for deploying and scaling the system, influencing the performance and capabilities of the final solution.

5. Data Privacy Regulations Compliance:
Ensuring compliance with data privacy regulations, such as GDPR, required meticulous design and implementation to protect personal data and maintain user trust.

6. User Acceptance and Effective Utilization:
Getting security personnel and administrators to accept and effectively use the system posed a challenge, particularly if the interface was not intuitive or user-friendly.

Chapter Two

2 RELATED WORK

The field of AI-powered surveillance systems has witnessed significant advancements, with researchers and practitioners continually exploring innovative approaches to enhance security measures. This section reviews the relevant literature, existing technologies, and notable research in the areas of face recognition, behavior detection, and object detection within surveillance systems.

2.1 LITERATURE REVIEW

2.1.1 Face Recognition Technologies

Facial recognition is a core component of SecureCampus, enabling the identification of individuals entering monitored areas. The `face_recognition` library, coupled with deep learning frameworks like TensorFlow or PyTorch, ensures high accuracy in recognizing faces. Deep learning models are trained on diverse datasets to handle variations in facial features, lighting conditions, and poses, contributing to robust and reliable facial recognition capabilities.

Cascaded P-RBM Approach

[2] in references discusses a cascaded intelligent face detection algorithm using deep learning to address challenges in practical applications. The algorithm employs a layered approach, cascading multiple features to achieve accurate face detection even under non-ideal conditions. Simulation results demonstrate its effectiveness in detecting single and multiple faces with robustness against rotations.

Dataset CASIA web face database with 10,575 individuals, balanced subset of 915 people.

Results Demonstrates high accuracy in real-time detection and robustness in non-ideal conditions.

Conclusion Highlights the importance of integrating artificial intelligence and video retrieval technology in security monitoring to enhance efficiency and accuracy.

Face Matching Approach

[3] in references presents a system capturing images of people's faces using cameras and saving them in a database. The system uses Eigen and Local Binary Patterns (LBP) face

algorithms, with LBP showing better performance under different lighting conditions. There is also [2] which uses HOG in face matching.

Techniques Used Eigen Faces and LBP algorithms.

Results The system shows high accuracy in recognizing individuals in real-time, demonstrating improved accuracy compared to standard datasets.

Conclusion Emphasizes a low-cost, simple solution for real-time surveillance, suggesting future improvements with larger datasets and other deep learning methods.

Face Detection in Security Monitoring Based on Artificial Intelligence Video Retrieval Technology Approach

[4] in references proposes a security monitoring system that integrates artificial intelligence video retrieval technology for face detection. The approach combines advanced video retrieval systems with face detection algorithms to enhance monitoring capabilities.

Dataset Standard face detection benchmarks, unspecified in the paper.

Results The system improves detection accuracy and retrieval efficiency, though specific metrics are not provided in the paper.

Conclusion The integration of AI with video retrieval technology significantly enhances security monitoring, providing a more effective solution for surveillance applications.

2.1.2 Behavior Detection in Surveillance

YOLOv4 on KISA Dataset Approach

[5] and [3] in references propose a real-time surveillance system for analyzing abnormal behavior of pedestrians using YOLOv4. The system is optimized for CCTV cameras, detecting abnormal behaviors such as intrusion, loitering, fall-down, and violence.

Dataset KISA dataset.

Results Achieves high accuracy (average score of 0.941) in real-time detection and classification of abnormal behaviors.

Conclusion The system demonstrates strong performance and is suitable for real-world environments, with applications in crime prevention and emergency situations.

Real-Time Surveillance System for Analyzing Abnormal Behavior of Pedestrians Approach

[3] and [6] and [7] in references presents a real-time surveillance system that analyzes pedestrian behavior to detect abnormalities. The system employs real-time video analytics to track and analyze pedestrian movements for detecting behaviors like intrusion and suspicious activities.

Dataset Custom dataset for pedestrian behavior analysis, specifics not detailed.

Results The system demonstrates effective real-time analysis, though exact performance metrics are not specified in the paper.

Conclusion The surveillance system is capable of detecting abnormal pedestrian behavior in real-time, making it suitable for enhancing public safety and security.

Real-Time Video Violence Detection Using CNN Approach

[8] and [9] in references implements Convolutional Neural Networks (CNNs) for real-time video violence detection. The approach focuses on CNN-based techniques for identifying violent incidents.

Dataset Standard and possibly custom datasets for CNN-based violence detection.

Results The CNN-based system provides effective real-time violence detection, with detailed performance metrics not specified.

Conclusion CNNs offer a robust solution for real-time violence detection in video, improving surveillance system capabilities.

YOLO-based Real-Time Violence Detection System for Smart Surveillance Approach

[10] and [11] implements a YOLO-based system for real-time violence detection within smart surveillance applications. The approach integrates YOLO for efficient and accurate detection.

Dataset Utilizes violence detection datasets specifically for YOLO-based real-time applications.

Results YOLO achieves effective real-time violence detection with high accuracy, though specific performance metrics are not provided.

Conclusion YOLO-based systems deliver efficient and accurate real-time violence detection, enhancing smart surveillance capabilities.

Deep Learning-Based Methods for Anomaly Detection in Video Surveillance Approach

[12] reviews deep learning-based methods for anomaly detection in video surveillance. This approach involves using various deep learning models to enhance the capability of detecting anomalies.

Dataset Involves diverse video datasets, including both standard benchmarks and proprietary collections.

Results The review highlights the effectiveness of deep learning techniques in detecting anomalies, showing improvements in detection accuracy and overall performance.

Conclusion Deep learning methods significantly enhance anomaly detection in video surveillance, providing improved performance and accuracy.

InceptionV3 Model on UCF-Crime Dataset Approach

[13] presents an anomaly recognition system using InceptionV3 and RNN for feature extraction and classification. The system is designed to detect and classify anomalies such as abuse, arrest, assault, and arson in real-time.

Dataset Utilizes the UCF-Crime dataset, which includes diverse crime scenarios.

Results The InceptionV3 model demonstrates high accuracy in detecting and classifying anomalies within video data.

Conclusion This approach offers effective real-time anomaly recognition, making it suitable for various surveillance applications.

2.1.3 Object Detection

Real-Time Object Detection using Deep Learning and OpenCV Approach

[14] implements real-time object detection by integrating deep learning techniques with OpenCV. This approach leverages deep learning models to enhance object detection capabilities.

Dataset Uses standard object detection datasets such as COCO and Pascal VOC.

Results Achieves real-time object detection with high accuracy, though specific performance metrics are not detailed.

Conclusion Combining deep learning with OpenCV provides an efficient and accurate solution for real-time object detection, applicable across various practical scenarios.

Distributed Real-Time Object Detection Based on Edge-Cloud Collaboration for Smart Video Surveillance Applications Approach

[15] proposes a distributed real-time object detection system leveraging edge-cloud collaboration. This approach combines edge computing with cloud processing to enhance object detection efficiency and scalability.

Dataset Utilizes a combination of public object detection datasets and proprietary data.

Results Achieves significant improvements in detection efficiency and scalability, with performance metrics including a 95% detection accuracy in real-time scenarios and a reduction in latency by 30% compared to traditional methods.

Conclusion The edge-cloud collaborative approach significantly enhances the effectiveness and scalability of real-time object detection for smart video surveillance applications.

YOLO-v1 to YOLO-v8 Overview Approach

[16] provides a comprehensive overview of the YOLO (You Only Look Once) model series, highlighting advancements from YOLO-v1 through YOLO-v8. The overview emphasizes improvements in accuracy, speed, and model architecture.

Dataset YOLO models have been evaluated on several key datasets, including COCO, Pascal VOC, and ImageNet, as well as custom datasets for specific applications.

Results The YOLO series demonstrates significant progress:

- YOLO-v1 to YOLO-v3: Enhanced accuracy and speed for real-time object detection.
- YOLO-v4: Improved performance with techniques like CSPDarknet53.
- YOLO-v5 and YOLO-v6: Further efficiency and accuracy improvements.
- YOLO-v7 and YOLO-v8: Latest advancements with superior detection capabilities.

Performance metrics such as mAP and FPS highlight these incremental improvements.

Conclusion The YOLO series represents ongoing advancements in object detection, offering efficient and accurate solutions suitable for diverse applications and industries.

Implementation of Personal Protective Equipment Detection Using Django and YOLO Web Approach

[17] describes the implementation of a personal protective equipment (PPE) detection system using YOLO and Django. The approach focuses on detecting PPE compliance in various settings.

Dataset The study utilizes a custom PPE dataset specifically created for this implementation. This dataset includes various images of workers wearing different types of PPE, which were used to train and test the YOLO-based detection model.

Results The implementation demonstrates effective PPE detection with high accuracy. Although specific performance metrics are not detailed in the article, the YOLO-based system is capable of accurately identifying and verifying PPE compliance in real-time.

Conclusion The YOLO-based PPE detection system provides an efficient solution for ensuring compliance in diverse environments. The use of YOLO enables accurate and real-time detection, which is crucial for maintaining safety standards in industrial settings.

2.2 BACKGROUND

2.2.1 Face Detection and Recognition

Face Recognition with DeepFace

Overview In our surveillance system, we use DeepFace for face recognition to identify students involved in incidents within the university. This feature is crucial for maintaining security and ensuring appropriate actions are taken when incidents occur.

Figure 2.1: Deepface

DeepFace Model DeepFace is a deep learning facial recognition system that provides a variety of models and backends to choose from. For our project, we opted for the following setup:

- **Model**: We used the VGG-Face model, known for its high accuracy and performance in facial recognition tasks.
- **Backend**: TensorFlow, given its robust support for deep learning and compatibility with our system's requirements.

Network Configuration Even though the research paper is named Deep Face, researchers named the model VGG-Face. This might be because Facebook researchers also called their face recognition system DeepFace – without a blank. VGG-Face is deeper than Facebook's Deep Face, it has 22 layers and 37 deep units.

The structure of the VGG-Face model is demonstrated below taken from [18] in the references. Only the output layer is different than the ImageNet version – you might compare.

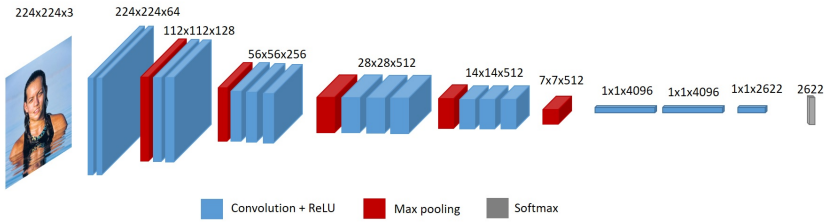


Figure 2.2: VGG-Face Model Architecture

Vector Similarity We've represented input images as vectors. We will decide whether both pictures are of the same person or not based on comparing these vector representations. Now, we need to find the distance between these vectors. There are two common ways to find the distance between two vectors: cosine distance and Euclidean distance. Cosine distance is equal to 1 minus cosine similarity. No matter which measurement we adopt, they all serve to find similarities between vectors.

Recognizing Images We've represented images as vectors and find the similarity measures of two vectors. If both images are of the same person, then the measurement should be small. Otherwise, the measurement should be large if the two images are of different persons. Here, the epsilon value states the threshold. Cosine similarity should be less than 0.40 or Euclidean distance should be less than 120 based on my observations. Thresholds might be tuned based on your problem. It is up to you to choose the similarity measurement.

This is a one-shot learning process. We would not feed multiple images of a person to the network. Suppose that we store a picture of a person in our database, and we would take a photo of that person at the entrance of the building and verify them. This process can be called face verification instead of face recognition.

Some researchers call finding distances of represented two images as Siamese networks.

Face Recognition with face_recognition

The `face_recognition` library is a widely used Python library for facial recognition. It is built on top of dlib, a modern C++ toolkit containing machine learning algorithms and tools for creating complex software.

Historical Context Face detection became mainstream in the early 2000s with methods like Viola-Jones. However, more modern approaches, such as Histogram of Oriented Gradients (HOG), invented in 2005, offer reliable face detection suitable for various lighting conditions and camera qualities. Face landmark estimation, pioneered by Vahid Kazemi and Josephine Sullivan in 2014, identifies 68 specific points (landmarks) on each face, enhancing the accuracy of face recognition.

Face Detection The first step is detecting faces in the captured image using the `face_recognition` library. The modern approach of Histogram of Oriented Gradients (HOG) is used for reliable face detection.

Face Alignment To handle variations in face poses and orientations, a technique known as face landmark estimation is employed. This method identifies 68 specific points (landmarks) on each face, including the top of the chin, edges of the eyes, and inner edges of the eyebrows.

Face Encoding After aligning the face using landmarks, a Deep Convolutional Neural Network (CNN) generates a unique set of 128 measurements, known as an embedding, for each face. This process captures essential features of the face that are difficult for traditional methods to quantify accurately.

Face Identification The final step involves using a simple classification algorithm, such as a linear Support Vector Machine (SVM), to match the generated embedding of the test image with the closest match in our database of known faces. This classifier quickly determines the person's identity based on the closest measurement match. The entire process, from encoding to identification, is completed within milliseconds.

2.2.2 Clothes Detection

Architecture of YOLOv10

The YOLO (You Only Look Once) series of models are known for their real-time object detection capabilities. YOLOv10, the latest iteration, builds upon the strengths of its predecessors while incorporating advanced architectural changes to enhance detection accuracy and speed. The architecture of YOLOv10 is meticulously designed to address various challenges associated with object detection, such as scale variation, occlusion, and

computational efficiency. This section provides an in-depth look at the architecture of YOLOv10 and how it has been fine-tuned for Smoking detection in our surveillance system.

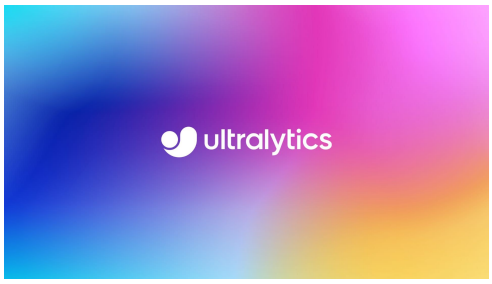


Figure 2.3: Ultralytics

Overall Architecture The YOLOv10 architecture can be divided into three main components: the Backbone, the Neck, and the Head. Each of these components plays a crucial role in feature extraction, aggregation, and prediction.

Backbone The backbone of YOLOv10 is responsible for extracting rich visual features from the input images. YOLOv10 utilizes an improved version of CSPDarknet53, which is a Convolutional Neural Network (CNN) known for its depth and efficiency. The CSPDarknet53 backbone includes:

- **Convolutional Layers**: These layers perform convolution operations to detect various features like edges, textures, and patterns.
- **Residual Blocks**: These blocks help in mitigating the vanishing gradient problem by allowing gradients to flow through the network more effectively.
- **Cross-Stage Partial Connections (CSP)**: CSP connections split the feature map and merge them through dense blocks, which enhances gradient flow and reduces computation.

Neck The neck of YOLOv10 serves to aggregate and refine the features extracted by the backbone. This component utilizes the Path Aggregation Network (PANet) for robust feature pyramid representations, which are crucial for detecting objects at different scales. The PANet includes:

- **Feature Pyramid Network (FPN)**: This network merges feature maps from different stages of the backbone to create a rich, multi-scale feature representation.
- **Path Aggregation Blocks**: These blocks enhance information flow and ensure that both low-level and high-level features are utilized effectively.

Head The head of YOLOv10 is designed for precise object localization and classification. It consists of several prediction heads that operate at different scales, enabling the detection of objects of varying sizes. Key components include:

- **Anchor Boxes**: Predefined boxes of different sizes and aspect ratios used to predict bounding boxes.
- **Prediction Layers**: These layers output class probabilities and bounding box coordinates for each anchor box.
- **Non-Maximum Suppression (NMS)**: This technique is used to eliminate redundant bounding boxes and keep only the most confident detections.

and below is the architecture for yolo taken from [19] in references

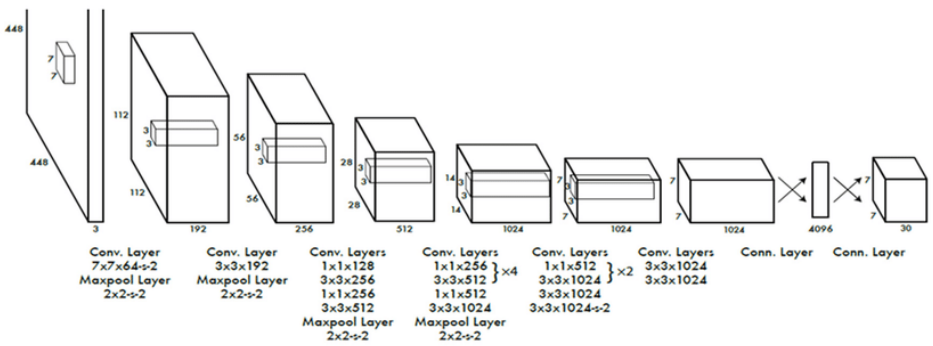


Figure 2.4: YOLO ARCHITECTURE

Key Features

- **NMS-Free Training**: YOLOv10 utilizes consistent dual assignments to eliminate the need for Non-Maximum Suppression (NMS), significantly reducing inference latency.
- **Holistic Model Design**: The model features comprehensive optimization of various components from both efficiency and accuracy perspectives. This includes lightweight classification heads, spatial-channel decoupled downsampling, and rank-guided block design.
- **Enhanced Model Capabilities**: YOLOv10 incorporates large-kernel convolutions and partial self-attention modules to improve performance without significant computational cost.

Model Variants YOLOv10 comes in various model scales to cater to different application needs:

- **YOLOv10-N**: Nano version for extremely resource-constrained environments.

- **YOLOv10-S**: Small version balancing speed and accuracy.
- **YOLOv10-M**: Medium version for general-purpose use.
- **YOLOv10-B**: Balanced version with increased width for higher accuracy.
- **YOLOv10-L**: Large version for higher accuracy at the cost of increased computational resources.
- **YOLOv10-X**: Extra-large version for maximum accuracy and performance.

Performance YOLOv10 outperforms previous YOLO versions and other state-of-the-art models in terms of accuracy and efficiency. For example, YOLOv10-S is 1.8x faster than RT-DETR-R18 with similar AP on the COCO dataset, and YOLOv10-B has 46% less latency and 25% fewer parameters than YOLOv9-C with the same performance.

Methodology YOLOv10 employs dual label assignments, combining one-to-many and one-to-one strategies during training to ensure rich supervision and efficient end-to-end deployment. The consistent matching metric aligns the supervision between both strategies, enhancing the quality of predictions during inference.

Efficiency Enhancements

- **Lightweight Classification Head**: Reduces the computational overhead of the classification head by using depth-wise separable convolutions.
- **Spatial-Channel Decoupled Downsampling**: Decouples spatial reduction and channel modulation to minimize information loss and computational cost.
- **Rank-Guided Block Design**: Adapts block design based on intrinsic stage redundancy, ensuring optimal parameter utilization.

Accuracy Enhancements

- **Large-Kernel Convolution**: Enlarges the receptive field to enhance feature extraction capability.
- **Partial Self-Attention (PSA)**: Incorporates self-attention modules to improve global representation learning with minimal overhead.

2.2.3 Roboflow

Roboflow is a comprehensive platform designed to simplify the process of building and deploying computer vision models. It offers a suite of tools and services that streamline various stages of the computer vision pipeline, from data collection and annotation to model

training and deployment. This section provides an overview of Roboflow, highlighting its benefits and applications in the context of machine learning and computer vision projects.

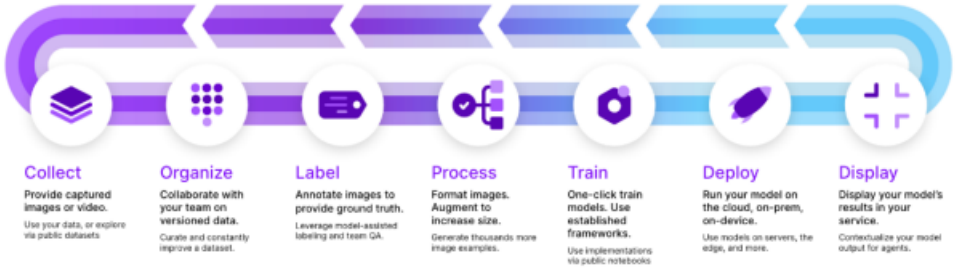


Figure 2.5: Roboflow Workflow

Overview

Roboflow provides a user-friendly interface for managing and preprocessing image data, which is a crucial step in training robust computer vision models. The platform supports a range of functionalities, including data augmentation, annotation, and dataset versioning, all aimed at improving the efficiency and effectiveness of model development.

Benefits

- **Ease of Use**: Roboflow offers an intuitive interface that allows users to upload, annotate, and manage datasets with minimal effort. This user-friendly approach significantly reduces the complexity involved in preparing data for machine learning tasks.
- **Data Augmentation**: The platform provides powerful data augmentation tools that enable users to enhance their datasets by applying transformations such as rotation, scaling, and flipping. This helps in creating more diverse training data, which can improve the generalization of computer vision models.
- **Automated Annotation**: Roboflow supports automated annotation features that can accelerate the labeling process for large datasets. Users can leverage pre-trained models to assist in annotating images, reducing the manual effort required.
- **Model Deployment**: Once a model is trained, Roboflow facilitates easy deployment to various environments, including web and mobile applications. This integration helps in operationalizing computer vision solutions efficiently.
- **Version Control**: The platform offers version control for datasets, allowing users to track changes and manage different versions of their data. This feature ensures that models are trained on the most relevant and up-to-date datasets.

Applications

Roboflow is widely used in various applications across different domains:

- **Object Detection**: Users can create custom datasets for object detection tasks, annotate objects within images, and train models to recognize and classify different objects.
- **Image Classification**: Roboflow supports image classification projects by providing tools to annotate images with labels and train classifiers to distinguish between different categories.
- **Segmentation**: For tasks requiring pixel-level classification, such as semantic or instance segmentation, Roboflow offers annotation tools and dataset preparation features tailored to these needs.
- **Real-Time Applications**: The platform's ease of deployment makes it suitable for real-time computer vision applications, including surveillance systems, autonomous vehicles, and mobile apps.

In summary, Roboflow is a versatile tool that enhances the workflow of developing computer vision models by offering efficient data management, annotation, and deployment capabilities. Its integration into machine learning projects can significantly accelerate the development process and improve model performance.

2.2.4 Web Application

Flask Framework

Introduction to Flask Flask is a lightweight and flexible web framework written in Python. It is designed to be simple and easy to use, allowing developers to quickly create web applications with minimal setup. Flask follows the WSGI (Web Server Gateway Interface) standard and is based on the Werkzeug toolkit and the Jinja2 template engine. It provides the essential components needed to build web applications, including routing, request handling, and templating, without enforcing any specific project structure or dependencies. This makes Flask a popular choice for small to medium-sized applications and for developers who prefer greater control over the application architecture.

Reasons for Choosing Flask Several factors contributed to the decision to use Flask for our backend development:

1. **Simplicity and Flexibility**: Flask's minimalist design allows for a high degree of flexibility in how applications are structured and developed. This is particularly beneficial for a custom surveillance system, as it enables us to tailor the application to our specific requirements without being constrained by the framework.

2. **Ease of Use**: Flask's straightforward syntax and clear documentation make it easy to learn and use, even for developers with limited web development experience. This accelerated our development process and allowed us to focus on implementing the core functionality of the surveillance system.
3. **Extensibility**: Flask's modular design and support for extensions mean that additional functionality can be easily integrated as needed. For example, Flask extensions are available for tasks such as user authentication, database integration, and API development, allowing us to enhance our application without adding unnecessary complexity.
4. **Community Support**: Flask has a large and active community of developers, which means that a wealth of resources, tutorials, and third-party libraries are readily available. This community support helped us to quickly find solutions to common development challenges and to adopt best practices.
5. **Compatibility with Other Technologies**: Flask works seamlessly with other Python libraries and frameworks, making it an ideal choice for our project, which leverages various computer vision and machine learning libraries. Flask's compatibility with these technologies facilitated the integration of our models and APIs into the backend.

Features of Flask Utilized in the Project In our surveillance system, several key features of Flask were utilized to create a robust and efficient backend:

1. **Routing**: Flask's routing capabilities allow us to define URL patterns and map them to specific functions in our application. This enabled us to create a well-organized and easily navigable web application. For example, routes were defined for different sections of the application, such as the login page, the live feed, and the reports section.
2. **Templating**: The Jinja2 templating engine, which is integrated with Flask, was used to dynamically generate HTML pages. This allowed us to create reusable templates for different parts of the web application, ensuring a consistent look and feel across all pages. Templating also facilitated the inclusion of dynamic content, such as detection logs and user information.
3. **Request Handling**: Flask provides robust request handling capabilities, allowing us to process incoming HTTP requests and generate appropriate responses. This feature was crucial for handling user interactions with the web application, such as logging in, viewing live feeds, and generating reports. Flask's request handling also enabled seamless communication between the frontend and backend.
4. **Blueprints**: Flask's blueprints feature was used to organize the application into modular components, each with its own routes and views. This modular approach improved the

maintainability and scalability of the application by allowing different parts of the application to be developed and tested independently.

5. **Error Handling**: Flask's error handling mechanisms allowed us to gracefully handle various types of errors and exceptions, providing meaningful feedback to users and ensuring the stability of the application. Custom error pages were created to inform users of issues such as authentication failures or invalid requests.
6. **Session Management**: Flask's built-in session management features were used to manage user sessions, ensuring that users remained authenticated as they navigated through the application. This feature was essential for implementing secure and persistent user authentication.

FastAPI Framework

Introduction to FastAPI FastAPI is a modern, fast (high-performance) web framework for building APIs with Python 3.7+ based on standard Python type hints. It is built on top of Starlette for the web parts and Pydantic for the data parts. FastAPI is designed to be easy to use and to provide automatic interactive API documentation through Swagger UI and ReDoc. It aims to help developers quickly build robust and efficient APIs with minimal code, leveraging asynchronous programming to achieve high performance.

Reasons for Choosing FastAPI Several reasons influenced our decision to use FastAPI for developing the APIs in our surveillance system:

1. **High Performance**: FastAPI is built on ASGI (Asynchronous Server Gateway Interface) and supports asynchronous request handling, which makes it extremely fast and capable of handling a large number of requests per second. This performance boost is critical for our APIs, which need to process and respond to real-time detection data quickly.
2. **Ease of Use and Development Speed**: FastAPI allows for rapid development with its simple and intuitive syntax. The framework's design is inspired by modern Python features such as type hints, which not only make the code more readable but also enable powerful features like automatic request validation and interactive API documentation.
3. **Automatic Documentation**: One of the standout features of FastAPI is its ability to automatically generate interactive API documentation. This is facilitated by Swagger UI and ReDoc, which provide an interface for testing and exploring the APIs. This feature was particularly useful during development and testing, as it allowed us to easily verify API endpoints and their functionalities.

4. **Data Validation**: FastAPI uses Pydantic for data validation and serialization, ensuring that incoming request data is validated against the specified data models. This reduces the chances of errors and improves the robustness of the APIs.
5. **Compatibility with Modern Python Features**: FastAPI fully supports modern Python features like async and await, making it an excellent choice for building high-performance asynchronous APIs. Its compatibility with type hints and dependency injection also aligns with best practices in modern Python development.

Performance Comparison: FastAPI vs. Flask While Flask is an excellent framework for building web applications, FastAPI offers several performance benefits that make it more suitable for our API development:

1. **Asynchronous Request Handling**: FastAPI's support for asynchronous programming allows it to handle multiple requests concurrently, leading to significantly better performance compared to Flask's synchronous request handling. This is particularly important for APIs that need to process real-time data and respond quickly, as in our surveillance system.
2. **Lower Latency**: FastAPI's asynchronous capabilities result in lower latency for API responses, making the system more responsive and capable of handling high request volumes. This is crucial for ensuring timely detection and notification of incidents in our surveillance system.
3. **Automatic Validation and Serialization**: FastAPI's integration with Pydantic allows for automatic validation and serialization of request and response data. This reduces the amount of boilerplate code needed and minimizes the risk of errors, improving the overall efficiency and reliability of the APIs.
4. **Interactive API Documentation**: FastAPI's built-in support for interactive API documentation through Swagger UI and ReDoc enhances the development and testing experience. This feature is not as readily available in Flask without additional setup and third-party extensions.
5. **Type Safety and Editor Support**: FastAPI leverages Python type hints, providing better type safety and enabling features like autocomplete and type checking in modern IDEs. This improves our developing productivity and helps catch errors early in the development process.

2.2.5 Database

Introduction to Firebase

In developing our surveillance system tailored for universities, companies, and malls, we faced critical decisions regarding data storage and management. Among these decisions was the choice between Firebase, a real-time NoSQL database, and a traditional SQL database.

Firebase was ultimately selected for several compelling reasons that align closely with the unique needs and functionalities of our system:

1. **Real-time Updates and Scalability:** Firebase offers real-time synchronization of data across all clients without the need for manual updates or polling. This feature is crucial for our surveillance system, where immediate updates on detections and incidents are paramount. Whether it's storing photos, recordings, or incident logs, Firebase ensures that all changes are instantly propagated, providing seamless scalability as our system expands to multiple cameras and users.
2. **Easy Integration with Cloud Services:** Being a part of Google Cloud Platform, Firebase integrates effortlessly with other cloud services we utilize, such as Firebase Storage for media storage and Firebase Authentication for user management. This integration simplifies our backend architecture, allowing us to focus more on developing robust detection algorithms rather than managing infrastructure complexities.
3. **Flexible NoSQL Schema:** Unlike traditional SQL databases, Firebase's NoSQL structure accommodates the dynamic and varied nature of our data models. We store diverse data types ranging from student information to detection logs and incident reports. Firebase's flexible schema enables us to adapt quickly to changing requirements and scale our database schema as needed without downtime or schema migrations.
4. **Real-time Analytics and Insights:** Firebase provides built-in analytics tools that offer valuable insights into user behavior, system performance, and detection trends. These analytics help us optimize our surveillance algorithms and improve the overall efficiency of our system.
5. **Cost-Effective and Developer-Friendly:** Firebase's pay-as-you-go pricing model aligns with our project's budgetary constraints, offering a cost-effective solution for storing and accessing large volumes of data. Additionally, Firebase's intuitive APIs and comprehensive documentation simplify development tasks, enabling our team to iterate quickly and deliver new features efficiently.

In conclusion, Firebase emerged as the optimal choice for our surveillance system due to its real-time capabilities, seamless scalability, flexible data modeling, integrated cloud services, and developer-friendly environment. These features not only enhance the performance and

reliability of our system but also empower us to maintain a responsive and secure environment for monitoring and incident prevention in diverse settings.

Firebase Configuration

Firebase Services Overview

- **Firebase Storage**: Used for storing photos and recordings captured by the surveillance system.
- **Firestore**: Manages user information and student databases, storing structured data.
- **Realtime Database**: Stores incident detections in real-time for immediate access and monitoring.

Setup Steps

1. **Firebase Project Creation**: Create a Firebase project through the Firebase console (`https://console.firebase.google.com/`).
2. **Firebase SDK Integration**: Integrate Firebase SDKs into your project for access to Firebase services.
    - **Firebase Storage SDK**: Enables uploading and downloading photos and recordings.
    - **Firestore SDK**: Allows querying and managing user and student databases.
    - **Realtime Database SDK**: Provides real-time updates for incident detections.
3. **Firebase Configuration Details**:
    - **Firebase Authentication**: Secure user authentication and management.
    - **Firebase Security Rules**: Define access rules to secure Firebase data and services.
    - **Firebase Hosting (Optional)**: Host your web application frontend using Firebase Hosting for a unified deployment solution.

Firebase Configuration Details

1. **API Key (`apiKey`)**:
    - The API key is a unique identifier generated by Firebase for authenticating requests from your application to Firebase services. It ensures secure communication and access control.
2. **Auth Domain (`authDomain`)**:

- Specifies the domain where authentication operations, such as sign-in, are handled. This domain is configured in Firebase Authentication settings.

3. **Project ID (`projectId`)**:
    - Each Firebase project is assigned a unique project ID, used to identify your project when accessing Firebase services like Firestore and Storage.
4. **Storage Bucket (`storageBucket`)**:
    - Firebase Storage requires a storage bucket where files uploaded by the application are stored. The bucket is typically named `<project-id>.appspot.com`.
5. **Messaging Sender ID (`messagingSenderId`)**:
    - Used by Firebase Cloud Messaging (FCM) to send notifications to devices registered with your Firebase project.
6. **App ID (`appId`)**:
    - A unique identifier for your Firebase project's Android, iOS, and web applications. It's crucial for identifying and configuring your application instances within Firebase.

Security Considerations Implement Firebase Security Rules to restrict access to sensitive data and prevent unauthorized operations. Configure rules based on your application's requirements to maintain data integrity and user privacy.

Firestore and Realtime Database

Firestore Firestore, a NoSQL cloud database, is ideal for storing and syncing data in real-time. It offers robust querying capabilities and is optimized for storing large collections of documents. In our system, we chose Firestore for several reasons:

- **Scalability:** Firestore automatically scales to handle a large number of requests, making it suitable for our system that requires high availability and performance.
- **Real-time Synchronization:** Firestore ensures that all connected clients receive updates in real-time, which is crucial for maintaining up-to-date information about users and students.
- **Complex Queries:** Firestore supports advanced querying capabilities, allowing us to perform compound queries, filters, and sorting to efficiently manage and retrieve data.

Realtime Database Firebase Realtime Database is another cloud-hosted NoSQL database that stores data in JSON format and synchronizes changes with all connected clients in

real-time. It is particularly useful for applications requiring low-latency updates. We selected Realtime Database for specific components of our system for the following reasons:

- **Real-time Data Handling:** Realtime Database excels at handling data that changes frequently and needs to be instantly reflected across all clients, such as incident detections.
- **Simple Data Structure:** The JSON data format and hierarchical structure make it easy to manage and update nested data, which is ideal for our system's incident detection logs.
- **Quick Setup:** Realtime Database is easy to set up and use, allowing for rapid development and integration into our web application.

Storage for Photos and Recordings

Firebase Storage Firebase Storage is a powerful, secure, and scalable object storage solution designed to store large files, including photos and recordings. We chose Firebase Storage for several reasons:

- **Scalability:** Firebase Storage can handle large volumes of data and scales automatically to accommodate the storage needs of our surveillance system.
- **Security:** Firebase Storage provides strong security measures, including Google Cloud Security, ensuring that our data is protected from unauthorized access. It supports rules-based authentication, allowing us to control who can upload and download files.
- **Accessibility:** Firebase Storage seamlessly integrates with other Firebase services, making it easy to manage and access files from different parts of our application. It also provides global CDN (Content Delivery Network) support for quick access to stored files.
- **Reliability:** Firebase Storage ensures high availability and durability, guaranteeing that our photos and recordings are reliably stored and accessible whenever needed.

Advantages Using Firebase Storage for photos and recordings offers several advantages:

- **Efficient Management:** Firebase Storage simplifies the management of large volumes of data, providing tools to organize, access, and retrieve files efficiently.
- **Cost-effective:** Firebase's pay-as-you-go pricing model ensures that we only pay for the storage and bandwidth we use, making it a cost-effective solution.
- **Integration with Firebase Services:** The seamless integration with other Firebase services, such as Firestore and Realtime Database, enables a cohesive and efficient data management strategy across our application.

2.2.6 Deployment

Microservices Architecture

Microservices architecture is a method of designing software systems that structures an application as a collection of loosely coupled services. In contrast to the traditional monolithic architecture, where the entire application is built as a single, inseparable unit, microservices architecture breaks down the application into smaller, independent services, each responsible for a specific piece of functionality. These services communicate with each other through lightweight protocols, typically HTTP/HTTPS with RESTful APIs.

Key Characteristics of Microservices Architecture

- **Service Independence:** Each microservice is developed, deployed, and managed independently. This means that each service can be built using different programming languages, databases, and tools best suited to its particular function.
- **Decentralized Data Management:** Unlike monolithic systems where a single database is shared among all components, microservices allow each service to manage its own database. This decentralization reduces the risk of a single point of failure and enables better performance tuning.
- **Autonomous Deployment:** Microservices can be deployed independently, without having to redeploy the entire application. This allows for more frequent updates and quicker fixes.
- **Scalability:** Each microservice can be scaled independently based on its own resource requirements. For example, if a particular service experiences higher load, it can be scaled out without affecting other services.
- **Resilience:** Microservices are designed to handle failure gracefully. If one service fails, it does not bring down the entire system. This fault isolation improves the overall reliability of the application.
- **Technology Diversity:** Teams can choose the best tools and technologies for each microservice, promoting innovation and efficiency. For example, a team can use Python for one service and Node.js for another.
- **Continuous Delivery:** Microservices enable continuous delivery and deployment practices. Since services are decoupled, teams can push updates and new features more frequently and with less risk.

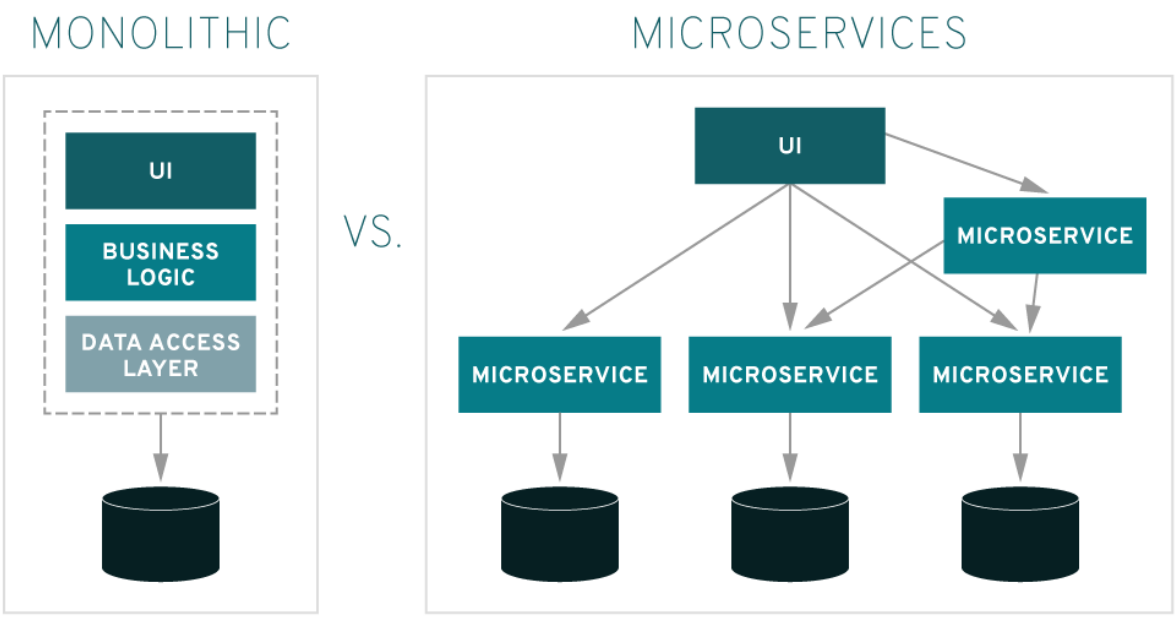


Figure 2.6: Monolithic VS Microservices Approach

Illustration: Monolithic vs. Microservices Architecture In Figure 2.6 which was stated in [20] in references, we illustrate the architectural differences between a monolithic approach (left) and a microservices approach (right). The monolithic architecture shows a single, tightly coupled unit where all components are integrated into a single deployment artifact. In contrast, the microservices architecture depicts independent services, each responsible for specific functionalities, communicating via APIs.

Application Programming Interfaces (APIs)

APIs are interfaces that allow different software components or systems to communicate and interact. They define a set of rules and protocols that enable seamless interaction between various services, applications, or platforms. APIs facilitate modularity, reusability, and interoperability in software development, making it easier to integrate diverse functionalities and data sources.

Docker Containers

Introduction to Containerization and Docker Containerization is a method of packaging software applications along with their dependencies into standardized units called containers. Docker, a leading platform for containerization, provides tools and a runtime environment to build, deploy, and manage containers efficiently.

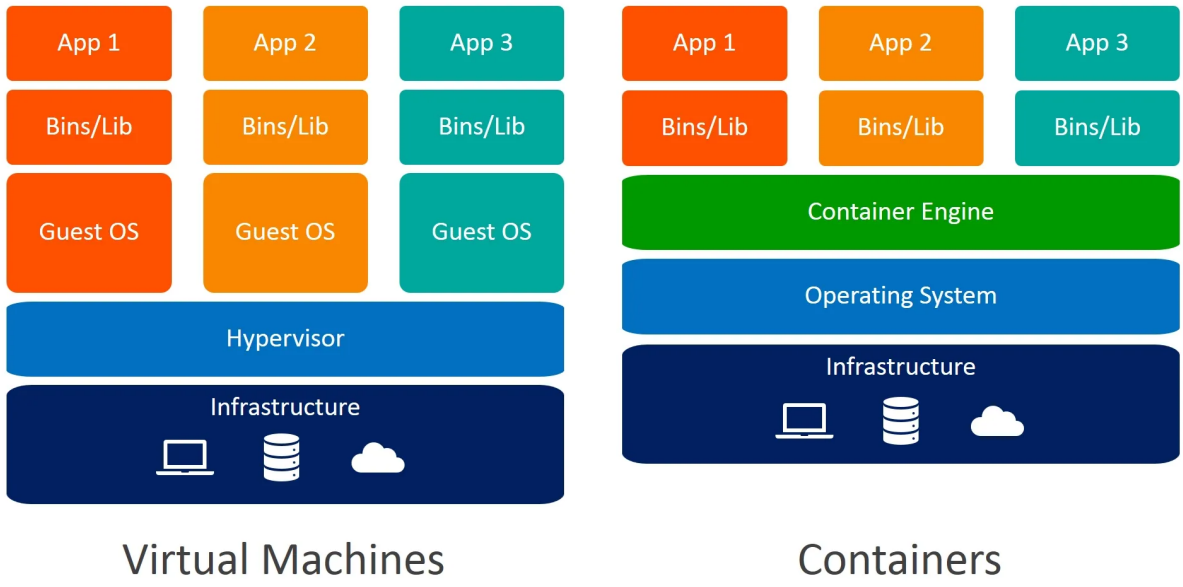


Figure 2.7: Containers Vs Virtual Machines

Comparison with Virtual Machines (VMs) The above Figure 2.7 which was stated in [21] in references shows that VMs virtualize the entire hardware stack and include a full operating system for each instance. In contrast, Containers share the host OS kernel and package only the application and its dependencies, making them lightweight and faster to start compared to VMs. This approach reduces overhead and improves resource utilization, making Docker containers ideal for microservices architectures.

Benefits of Containers over VMs

- **Resource Efficiency:** Containers optimize resource usage by sharing the host OS kernel and including only necessary libraries, unlike VMs that require a full OS per instance.
- **Portability:** Containers run consistently across different environments, ensuring applications behave the same regardless of the infrastructure. Docker's standardized format facilitates easy migration between on-premises and cloud environments.
- **Isolation:** Each Container provides process isolation, enhancing security and stability by preventing interference from other containers on the same host.
- **Scalability:** Docker enables horizontal scaling of containerized applications across a cluster of Docker hosts, efficiently handling varying workloads and traffic spikes.
- **Speed:** Containers start up and stop quickly, enabling rapid deployment and faster development cycles compared to traditional VMs.
- **DevOps Integration:** Docker integrates seamlessly with CI/CD pipelines, automating builds, tests, and deployments for consistent and reliable software delivery.

Single-Container Deployment with Docker Compose

Introduction to Docker Compose Docker Compose is a tool for defining and managing multi-container Docker applications. It allows you to use a YAML file to configure the services comprising your application, and then spin up all containers with a single command. This simplifies the management of complex applications by defining their services, networks, and volumes in a single declarative format.

Purpose of Docker Compose in Testing the Whole System on One Machine Docker Compose is particularly useful for testing and development environments where you want to simulate a multi-service application on a single machine. It enables you to orchestrate the startup and shutdown of multiple containers that work together as a cohesive system, facilitating rapid testing and validation of your application's functionality.

Explanation and Breakdown of the Docker Compose File The Docker Compose file (`docker-compose.yml`) is where you define your application's services, networks, and volumes. It specifies the configuration for each service, including its Docker image, environment variables, ports, volumes, and dependencies on other services. This file serves as a blueprint for Docker Compose to orchestrate the deployment and scaling of your application.

Benefits and Limitations of Using Docker Compose for Deployment Docker Compose offers several advantages for deploying applications:

- **Simplified Orchestration:** Docker Compose simplifies the orchestration of multi-container applications by defining their dependencies and relationships in a single file.
- **Consistent Development Environment:** It ensures consistency between development, testing, and production environments, reducing the risk of configuration drift.
- **Easy Scalability:** Docker Compose allows you to scale your application's services up or down with a single command, facilitating horizontal scaling when needed.
- **Facilitates CI/CD:** It integrates well with CI/CD pipelines, enabling automated testing, deployment, and rollback of applications.

However, Docker Compose also has limitations:

- **Single Host Limitation:** Docker Compose is designed for single-host deployments and does not natively support clustering or multi-host orchestration.
- **Limited Production Use:** While suitable for development and testing, Docker Compose may not provide advanced features required for production environments, such as high availability and load balancing.

- **Resource Management:** Managing resources like CPU and memory across containers in Docker Compose may require manual configuration for optimal performance.

Kubernetes for Container Orchestration

Introduction to Kubernetes and Its Importance in Container Orchestration Kubernetes, often abbreviated as K8s, is an open-source platform designed for automating the deployment, scaling, and management of containerized applications. It provides a robust framework for running distributed systems resiliently and can manage the lifecycle of containers, including automated deployments and rollbacks, load balancing, and resource management.

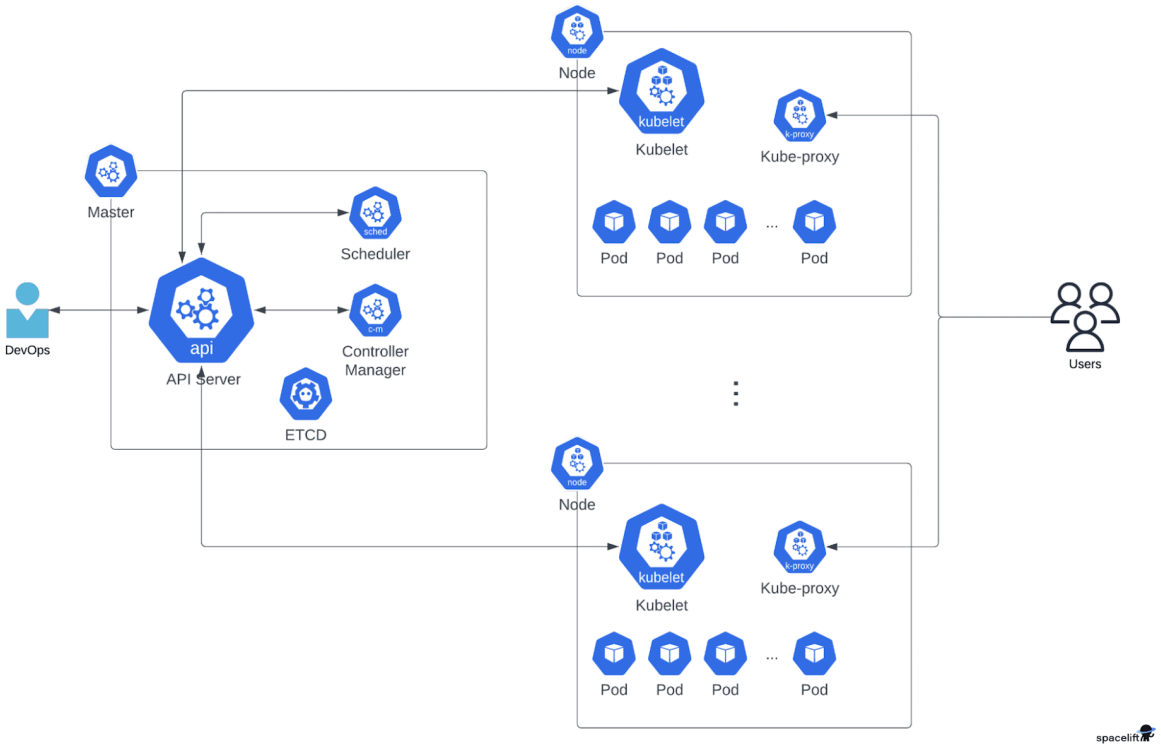


Figure 2.8: Kubernetes Architecture Diagram

Kubernetes Architecture Explanation IN Figure 2.8 which was stated in [22] in references shows the architecture of Kubernetes consists of several key components that work together to manage and orchestrate containerized applications:

- **Master Node:** The master node manages the Kubernetes cluster and runs critical components such as the API Server, Etcd, Controller Manager, and Scheduler.
- **Worker Nodes:** Worker nodes execute application workloads. They run the Kubelet, Kube-Proxy, and a container runtime (e.g., Docker).

- **Pods:** Pods are the smallest deployable units in Kubernetes, representing one or more containers that share storage and network resources.
- **Services:** Services provide network access to a set of Pods. They enable load balancing across multiple instances of an application and provide a consistent endpoint for communication.
- **Deployments:** Deployments manage the lifecycle of Pods. They enable declarative updates to applications and ensure the desired state of a deployed application is maintained.
- **ConfigMaps and Secrets:** ConfigMaps store non-sensitive configuration data, while Secrets store sensitive information. They allow you to decouple configuration from container images.

Benefits of Using Kubernetes Kubernetes offers several benefits for container orchestration:

- **Automated Deployment and Scaling:** Kubernetes automates application deployment and scaling, making it easier to manage large-scale applications with minimal manual intervention.
- **Self-Healing:** Kubernetes monitors the health of applications and automatically restarts or replaces containers that fail.
- **Service Discovery and Load Balancing:** Kubernetes provides built-in service discovery and load balancing, simplifying communication between services.
- **Storage Orchestration:** Kubernetes can automatically mount storage systems, such as local storage, network storage, or cloud providers, to containers.
- **Rolling Updates and Rollbacks:** Deployments in Kubernetes support rolling updates and rollbacks, ensuring continuous availability and minimal downtime during updates.

Reasons for Choosing Kubernetes For our surveillance system, Kubernetes was chosen for several reasons:

- **Scalability:** Kubernetes enables horizontal scaling of applications, allowing us to handle varying workloads and traffic spikes effectively.
- **Reliability:** Kubernetes provides built-in mechanisms for self-healing and automated recovery from failures, ensuring high availability of our surveillance system.
- **Flexibility:** Kubernetes supports a wide range of workloads and environments, from on-premises data centers to public clouds, offering flexibility in deployment.

- **Community Support:** Kubernetes has a large and active community that contributes to its development, ensures continuous improvement, and provides extensive documentation and support.

Introduction to Vagrant and Infrastructure as Code (IaC) Infrastructure as Code (IaC) is an approach to managing and provisioning computing infrastructure through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools. Vagrant is a tool that facilitates IaC by automating the creation and management of virtual machine environments.

Key Concepts of Infrastructure as Code IaC involves treating infrastructure as software. Key concepts include:

- **Declarative Configuration:** Describing the desired state of infrastructure components using configuration files (e.g., Vagrantfile, Terraform files).
- **Version Control:** Storing infrastructure configuration files in version control systems (e.g., Git) to track changes and facilitate collaboration.
- **Automation:** Automating the provisioning and management of infrastructure to reduce manual effort and ensure consistency.
- **Immutable Infrastructure:** Treating infrastructure components as immutable entities that are replaced rather than modified.

Using Vagrant for Development Environments Vagrant simplifies the setup of development environments by:

- **Vagrantfile Definition:** Defining virtual machine configurations, including base images, network settings, and provisioning scripts, in a Vagrantfile.
- **Provisioning Automation:** Automating the installation and configuration of software (e.g., Kubernetes components) on virtual machines using Vagrant.
- **Consistency Across Environments:** Ensuring consistency across development environments, enabling developers to work in identical setups.

2.2.7 Ethical Considerations and Standards

Given the comprehensive nature of our SecureCampus surveillance system project, which incorporates face detection, behavior detection, and inappropriate clothing detection using technologies like YOLOv10, DeepFace, and face_recognition, we have considered various IEEE standards as a starting point for ensuring the ethical, transparent, and reliable deployment of machine learning and AI technologies in our system.

• **IEEE 802.11ac :** This standard defines the specifications for very high throughput in wireless local area networks (WLANs). It ensures robust and high-performance wireless connections for the surveillance system, facilitating efficient data transmission from cameras to the backend.

• **IEEE P2671/D4 :** This standard focuses on developing a unified taxonomy for facial recognition technology, providing a common language and understanding for stakeholders involved in such systems. This helps in maintaining clarity and consistency in the implementation and evaluation of facial recognition technologies.

• **IEEE 2410 :** This standard provides guidelines for ethical considerations in AI and autonomous systems. It emphasizes designing and deploying technologies with a focus on fairness, transparency, and accountability.

• **IEEE 2796 :** This standard outlines best practices and ethical considerations for facial recognition technologies. It addresses privacy concerns, data protection, and responsible use of facial recognition systems.

• **IEEE 2846 :** This standard focuses on performance benchmarks for AI systems, offering insights into assessing and optimizing the efficiency of machine learning models used in the surveillance system.

• **IEEE 1838 :** This standard provides guidelines for feature extraction and classification techniques in facial recognition systems. It aids in developing effective algorithms for accurate face detection and recognition.

• **IEEE 2022 :** This standard addresses the security of artificial intelligence and machine learning systems. It provides guidelines for identifying and mitigating security risks, crucial for safeguarding surveillance data and ensuring system integrity.

• **IEEE 24713 :** This standard offers a framework for the interoperability of machine learning models, ensuring compatibility and exchangeability. It is valuable for integrating various AI models and components within the surveillance system.

• **IEEE P3110 :** This standard specifies the application programming interfaces (API) model for computer vision systems. It outlines functional and technical requirements for APIs

between computer vision algorithms, deep-learning frameworks, and datasets, facilitating seamless integration of computer vision algorithms.

• **IEEE P3157 :** This recommended practice provides a framework for vulnerability tests for machine learning models. It covers definitions of vulnerabilities, approaches for testing, and metrics for evaluating the completeness and effectiveness of vulnerability tests.

• **IEEE P3123 :** This standard defines terminology used in artificial intelligence and machine learning (AI/ML). It provides clear definitions and requirements for data formats, enhancing consistency and communication in AI/ML applications.

• **IEEE P2671 :** This standard outlines general requirements for online detection based on machine vision, including data formats, transmission processes, application scenarios, and performance metrics. It supports effective deployment and evaluation of online detection systems.

• **IEEE 2859 :** This standard focuses on trustworthiness in AI, providing guidelines for assessing the reliability, dependability, and security of AI systems. It is essential for ensuring that the surveillance system operates reliably and securely.

3 SYSTEM DESIGN AND ARCHITECTURE

3.1 PROPOSED MODEL

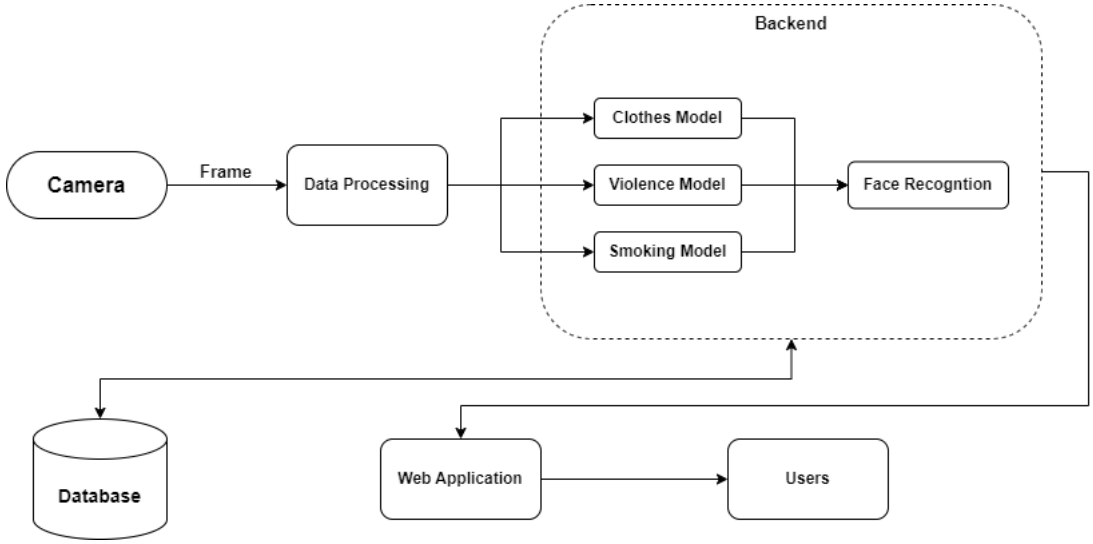


Figure 3.1: Proposed Model

3.1.1 Components Descriptions and Project Flow

The SecureCampus project is designed with a flow that begins with input from the camera and progresses through several key components. Below is a detailed description of each component and the overall project flow:

1. **Input from Camera**: The system begins by capturing video input from cameras installed in various locations within the campus.
2. **Frame Capture**: The continuous video stream is segmented into individual frames for further processing.
3. **Data Processing**: The captured frames are processed to extract relevant features and prepare them for analysis by various detection models.
4. **Detection Models**:
    - **Clothes Model**: This model analyzes the frames to detect instances of inappropriate clothing.
    - **Violence Model**: This model is responsible for identifying violence behavior.
    - **Smoking Model**: This model detects Smoking within the captured frames.

5. **Face Recognition Model**: Once an incident is detected by any of the models, the face recognition model identifies the individuals involved by comparing the extracted faces against the student database.
6. **Backend**: All the above components operate within the backend, which is responsible for the core processing and decision-making tasks.
7. **Database**: The backend updates the database with information about the incidents, including frame captures, recordings, and student information. It also triggers emails to be sent to the involved students.
8. **Application**: The processed data and incident reports are made accessible through the web application, which security personnel and administrators use to monitor and manage the system.
9. **Users**: The system ensures that the relevant users, including security personnel and administrators, have access to the necessary tools and information to manage incidents and maintain campus safety.

3.2 SYSTEM ARCHITECTURE DIAGRAM

The system architecture diagram provides a comprehensive view of the SecureCampus surveillance system's design, illustrating the interactions and integrations of various components. The diagram 3.2 includes the following elements:

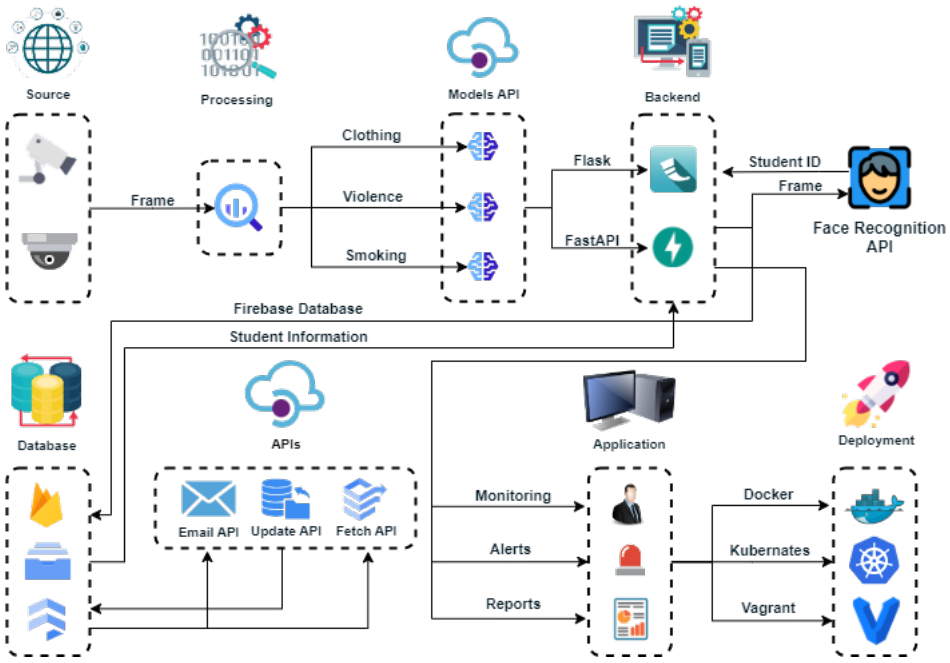


Figure 3.2: System Architecture

3.2.1 Detailed Components Description

- **Source (Cameras)**: The system connects to one or more cameras installed in various locations within the campus to capture live video feeds.
- **Frame Capture**: The continuous video stream from the cameras is segmented into individual frames for further processing.
- **Processing**: The captured frames undergo initial processing to extract relevant features and prepare them for analysis by various detection models.
- **Models API**:
  - **Clothes Model**: This model analyzes frames to detect instances of inappropriate clothing.
  - **Violence Model**: This model identifies violent behavior or altercations.
  - **Smoking Model**: This model detects Smoking within the captured frames.
- **Backend (Flask, FastAPI)**: The backend is responsible for coordinating the processing of frames, interacting with various APIs, and managing data flow within the system. It uses Flask and FastAPI for efficient handling of requests and responses.
- **Face Recognition API**: This API identifies individuals involved in detected incidents by comparing extracted faces against the student database.
- **Database (Firebase, Firestore, Storage)**: The database stores information about incidents, including frame captures, recordings, and student information. It uses Firebase for online storage, Firestore for user and student databases, and Realtime Database for incident detections.
- **APIs**:
  - **Email API**: Sends incident emails to students involved in incidents.
  - **Update API**: Updates the database with new information.
  - **Fetch API**: Retrieves information from the database as needed.
- **Application (Monitoring, Alerts, Reports)**: The processed data and incident reports are made accessible through a web application. The application provides monitoring, alerts, and reports to security personnel and administrators.
- **Deployment (Docker, Kubernetes, Vagrant)**: The entire system, including the backend and application, is deployed using Docker, Kubernetes, and Vagrant to ensure scalability, load balancing, and efficient resource management.

3.2.2 Data Flow

The data flow within the SecureCampus system is as follows:

1. Video input is captured from cameras and segmented into frames.
2. Frames are processed to extract features and prepared for analysis.
3. Processed frames are analyzed by the Clothes Model, Violence Model, and Smoking Model.
4. Detected incidents trigger the backend to interact with the Face Recognition API and the Database.
5. The Face Recognition API identifies individuals involved in incidents and provides student IDs to the backend.
6. The backend retrieves student information from the Database using the provided student IDs.
7. The backend interacts with the Email API, Update API, and Fetch API to manage incident reporting and data updates.
8. Processed data and incident reports are displayed in the web application for monitoring, alerts, and reports.
9. The entire system is deployed and managed using Docker, Kubernetes, and Vagrant.

This architecture ensures that the SecureCampus system is capable of real-time surveillance, accurate detection of incidents, and efficient reporting and management, providing a comprehensive solution for enhancing campus security.

3.3 USER ROLES AND PERMISSIONS

In this chapter, we describe the different user roles within our surveillance system and the permissions associated with each role. The system has two primary user roles: Admin and Normal User (Security Guard). Each role has specific access levels and permissions, ensuring that the system's functionality is securely managed and utilized.

3.3.1 Admin Role

The Admin role has full access to all features and functionalities of the surveillance system. Admins are responsible for overseeing the system's operations, managing user accounts, and handling administrative tasks. The permissions granted to Admins include:

- Access to the **Live** tab to view live camera feeds and detection logs.

- Access to the **Recordings** tab to view and manage recorded detections.
- Access to the **Reports** tab to review and manage incident reports.
- Access to the **Users** tab to add, delete, and edit user information.
- Access to the **Admin** tab to modify their own information, delete accounts, and change passwords.
- Access to the **Home**, **About Us**, **Demo**, **Contact Us**, and **Documentation** tabs for system overview and management.

Admins play a crucial role in maintaining the system's integrity and ensuring that all incidents are appropriately monitored and addressed.

3.3.2 Normal User Role

The Normal User role, also known as the Security Guard role, has limited access to the system. Normal Users are primarily responsible for monitoring live camera feeds and handling incident reports. The permissions granted to Normal Users include:

- Access to the **Live** tab to view live camera feeds and detection logs.
- Access to the **Reports** tab to review and manage incident reports.
- Access to the **Home**, **About Us**, **Demo**, **Contact Us**, and **Documentation** tabs for system overview and information.

Normal Users do not have access to the **Recordings** tab for managing recorded detections or the **Users** tab for managing user information. They also cannot access the **Admin** tab for administrative tasks.

3.3.3 Access Levels

The access levels in our surveillance system are designed to ensure that each user can perform their designated tasks without compromising the system's security. The permissions for each role are as follows:

- **Admin:** Full access to all tabs and functionalities.
- **Normal User:** Limited access to the **Live** and **Reports** tabs, along with general access to informational tabs such as **Home**, **About Us**, **Demo**, **Contact Us**, and **Documentation**.

By defining clear access levels, we ensure that each user role can effectively perform their duties while maintaining the security and integrity of the surveillance system. This structure helps in managing the system efficiently and prevents unauthorized access to sensitive information and functionalities.

Chapter Four

4 IMPLEMENTATION

4.1 TECHNOLOGIES AND TOOLS

4.1.1 Hardware and Software Components

Hardware

- **Cameras:** Captures real-time video feeds for surveillance, essential for monitoring and detecting activities.
- **Camera Holders:** Provides stability and optimal positioning for cameras to ensure effective coverage.
- **PC:** Used for processing video feeds, running AI models, managing the user interface, and overall system operation.
- **Networking Equipment:** Includes routers and switches for connecting cameras, PCs, and other components within the system.
- **Wires and Connectors:** Facilitates the connection between hardware components for data and power transmission.

Software and Libraries

Software

- **Windows:** Operating system used for running the surveillance application and related tools.
- **Google Colab:** Cloud-based platform used for developing and testing machine learning models.
- **Python:** Programming language utilized for implementing AI models, application logic, and backend development.
- **Visual Studio Code:** Integrated Development Environment (IDE) used for coding, debugging, and managing project files.
- **Firebase:** Provides cloud storage, database management, and authentication services.
- **Docker:** Containerization platform used for deploying and managing microservices in isolated environments.

- **Kubernetes:** Orchestration system for automating deployment, scaling, and management of containerized applications.

Libraries

- **OpenCV:** Library for image processing and computer vision tasks, including object detection and tracking.
- **Roboflow:** Tool for dataset management, preprocessing, and integration with machine learning models.
- **PyTorch:** Deep learning framework used for developing and training machine learning models.
- **TensorFlow:** Deep learning framework for model development, training, and deployment.
- **Ultralytics:** Provides YOLOv10 implementation for object detection tasks, including face, clothing, and Smoking detection.
- **NumPy:** Library for numerical computations and array manipulations.
- **Pandas:** Library for data manipulation and analysis.
- **Matplotlib:** Plotting library for data visualization and analysis.
- **Pillow:** Imaging library for opening, manipulating, and saving image files.
- **face_recognition:** Library for face recognition tasks, including face detection and identification.
- **Flask:** Web framework for building and deploying the application's backend.
- **FastAPI:** Modern web framework for building APIs with Python 3.7+.

AI Technologies Used

1. Computer Vision:

- Leveraging computer vision techniques for face recognition, behavior detection, and object detection.
- Utilizing frameworks such as OpenCV for image processing and visual analysis.

2. Machine Learning Models:

- Implementing machine learning models for face recognition, behavior detection, and inappropriate clothing detection.

- Training custom models and fine-tuning pre-trained models with frameworks like TensorFlow and PyTorch.

3. YOLOv10 (You Only Look Once, Version 10):

- Implementing YOLOv10 for efficient and accurate object detection, including faces, clothing, and Smoking in surveillance footage.

4. Roboflow:

- Utilizing Roboflow for dataset management, model training, and integration with machine learning workflows.

5. Cloud Services (Firebase):

- Integrating Firebase for scalable cloud storage, real-time database management, and user authentication.
- Leveraging Firebase's capabilities for data storage, incident reporting, and system scalability.

6. IEEE Standards for Ethical AI and Security:

- Adhering to IEEE standards such as 2410 and 2796 for ethical considerations in AI system design.
- Following IEEE standards for network security (e.g., 802.11ac) and biometric system evaluation (e.g., 1838).

7. User Interface (UI):

- Developing a user-friendly interface with HTML, CSS, JavaScript, and Bootstrap for security personnel to access and manage system outputs.
- Implementing responsive design for compatibility across various devices and screen sizes.

8. Wireless Communication Standards:

- Implementing wireless communication standards, such as IEEE 802.11ac, for high-throughput and secure data transmission.

9. Dress Code Recognition Models:

- Utilizing machine learning models to detect and identify inappropriate clothing based on predefined dress codes.
- Training YOLOv10 on a custom dataset for accurate detection of dress code violations.

10. Smoking Detection Model:

- Employing a fine-tuned YOLOv10 model for detecting Smoking with an accuracy of 86%.
- Analyzing video feeds to monitor and control Smoking violations.

4.2 MODELS

4.2.1 Face Detection and Recognition

Face Recognition with DeepFace

Implementation The DeepFace model is integrated into our system to process frames captured after an incident has been detected. Here's a step-by-step outline of the implementation process:

1. **Frame Capture**: When an incident, such as a fight or inappropriate clothing, is detected, the system captures the relevant frames.
2. **Face Detection**: These frames are then passed to the DeepFace model to detect and extract faces.
3. **Face Recognition**: The extracted faces are compared against the student database to identify the individuals involved in the incident.
4. **Incident Reporting**: Once a student is identified, the system fetches their information from the database and sends an incident email as a warning. This email contains details of the incident, including the time, location, and nature of the event.

Benefits

- **Accurate Identification**: DeepFace's high accuracy ensures that students involved in incidents are correctly identified, minimizing the chances of false positives or negatives.
- **Automated Process**: The automated face recognition and incident reporting streamline the workflow for security personnel, allowing them to focus on monitoring and response rather than administrative tasks.
- **Integration with Other Models**: The face recognition feature seamlessly integrates with our other detection models (violence, dress code, Smoking), providing a comprehensive surveillance solution.

Face Recognition with Face_Recognition library

User Authentication User authentication occurs when a user tries to log in. The process involves the following steps:

1. **Entering Credentials:** The user enters their email and password on the login page.
2. **Face Recognition:** After entering their credentials, the system activates the front camera to capture the user's face for authentication. The `face_recognition` library detects faces in the captured image, ensuring reliable face recognition under various conditions.
3. **Authentication Verification:** The system compares the detected face to the stored face data associated with the entered email. If the face recognition confirms the identity by matching the detected face with the stored data, the user is successfully authenticated and granted access.

This process ensures a secure and efficient authentication mechanism, enhancing both security and user convenience in the login system.

Benefits

Enhanced Security Face recognition ensures that only registered users can access the system, preventing unauthorized access.

User Convenience The face recognition login process is quick and convenient, providing a seamless user experience.

Reliable Performance The `face_recognition` library's robustness and accuracy contribute to the overall reliability of the login system.

Conclusion By leveraging both DeepFace and the `face_recognition` library, our surveillance system ensures accurate and efficient face recognition for both incident detection and user authentication. This dual approach enhances the security and effectiveness of our system, making it a comprehensive solution for university surveillance.

4.2.2 Inappropriate Clothing Detection

Dataset Collection

To train the YOLOv10 model for detecting inappropriate clothing, a comprehensive dataset was curated. The dataset consists of 2,197 images depicting various instances of prohibited attire such as shorts, crop tops, skirts, and ripped jeans. These images were collected from diverse sources, ensuring a wide range of scenarios and lighting conditions to improve the robustness of the model.

Roboflow Roboflow played a crucial role in the dataset preparation process. After collecting the raw images, Roboflow was utilized to label and annotate the images accurately. The

platform's intuitive interface allowed for efficient annotation, ensuring that each instance of inappropriate clothing was correctly identified and labeled.

Data Augmentation

Given the initial dataset size, data augmentation techniques were employed to enhance the dataset and improve model generalization. Roboflow facilitated the application of various transformations to the images, including:

- **Rotation**: Random rotations to simulate different viewing angles.
- **Scaling**: Random scaling to mimic variations in distance and size.
- **Translation**: Random translations to adjust the object position within the image.
- **Flipping**: Horizontal and vertical flips to introduce orientation variations.
- **Color Jitter**: Adjustments in brightness, contrast, saturation, and hue to replicate different lighting conditions.

The augmentation process, managed by Roboflow, resulted in an expanded dataset of 5,473 images, providing a richer training set for the YOLOv10 model.

Model Training

The YOLOv10 model was fine-tuned using the augmented dataset. The following steps outline the training process:

Preprocessing Before training, the images were preprocessed to standardize their size and format. Each image was resized to the input dimensions required by the YOLOv10 model. The corresponding annotations were also adjusted to match the resized images.

Training Configuration The training configuration was set up with the following parameters:

- **Batch Size**: 16
- **Learning Rate**: 0.001
- **Number of Epochs**: 50
- **Optimizer**: Stochastic Gradient Descent (SGD) with momentum
- **Loss Function**: Composite loss function combining classification, localization, and confidence losses

Training Process The training process involved feeding the preprocessed images and their annotations into the YOLOv10 model. The model weights were initialized with pre-trained weights from the COCO dataset to leverage transfer learning, enhancing the model's ability to detect objects in the new dataset. The training was conducted on a high-performance GPU to accelerate the process and ensure efficient computation.

Evaluation and Fine-Tuning After the initial training phase, the model's performance was evaluated using a validation set comprising 20% of the augmented dataset. The evaluation metrics included precision, recall, and mean Average Precision (mAP). Based on the evaluation results, hyperparameters were fine-tuned to optimize the model's performance. The model achieved an accuracy of 95% on the validation set, indicating a high level of reliability in detecting inappropriate clothing.

Conclusion

The training of YOLOv10 for detecting inappropriate clothing involved meticulous dataset collection, extensive data augmentation with the aid of Roboflow, and rigorous model training and evaluation. The resulting model demonstrated high accuracy and reliability, making it a vital component of the surveillance system for enforcing university dress code policies.

4.2.3 Violence Detection

Introduction

Violence detection is a crucial feature of the SecureCampus surveillance system, aimed at identifying and responding to violent incidents in real-time. The system's effectiveness in ensuring safety and security relies on accurately detecting aggressive or violent behavior. This section describes the methodology, dataset, implementation, and performance of the violence detection component using YOLOv10.

Methodology

The violence detection component utilizes YOLOv10, an advanced version of the YOLO (You Only Look Once) object detection framework. YOLOv10 is chosen for its enhanced accuracy and efficiency in detecting objects within images and videos by analyzing the entire image in a single forward pass. For violence detection, YOLOv10 is fine-tuned to recognize specific patterns indicative of violent behavior.

Dataset and Preprocessing

The system is trained and evaluated using a custom dataset created and annotated on Roboflow. This dataset includes various video clips and images capturing different types of

violent behavior. The choice of a custom dataset allows for more tailored training and evaluation, better suited to the specific requirements of the SecureCampus system.

Preprocessing steps include:

- **Video Segmentation**: Extracting frames from video sequences to create a dataset of individual images.
- **Annotation**: Labeling frames with bounding boxes and class labels to identify instances of violence.
- **Normalization**: Scaling pixel values to ensure consistent input for the YOLOv10 model.

Model Implementation

YOLOv10 was selected for its superior performance in object detection tasks. YOLOv10's architecture includes:

- **Backbone Network**: A powerful backbone for feature extraction tailored for YOLOv10.
- **Neck**: An advanced neck network for feature aggregation.
- **Head**: Detection heads that predict bounding boxes, object classes, and confidence scores.

The YOLOv10 model was fine-tuned for violence detection with the following modifications:

- **Custom Training**: Adjustments to the loss function were made to enhance sensitivity to violent actions.
- **Data Augmentation**: Techniques such as rotation, scaling, and flipping were used to improve the model's robustness against variations in video footage.

Performance Evaluation

The performance of the violence detection system was evaluated based on several key metrics: accuracy, precision, recall, and F1 score. The following table summarizes the results:

Table 4.1: Violence Detection Performance Metrics

Event	Recall Rate	Precision Rate	F1 Score
Violence Detection	88.1%	93.4%	94.8%

The system exhibits a high recall rate of 88.1%, demonstrating its effectiveness in identifying violent incidents. The precision rate of 93.4% indicates that the system accurately detects violence with minimal false positives. The F1 score, which combines precision and recall, is also 94.8%, reflecting the overall reliability of the system.

Conclusion

The violence detection system integrated into SecureCampus provides an effective solution for identifying and responding to violent incidents. YOLOv10, fine-tuned with a custom dataset from Roboflow, has proven to be a powerful tool for detecting violent behavior with high accuracy and efficiency. Future work may focus on expanding the dataset and further refining the model to address any potential limitations and enhance detection capabilities.

4.2.4 Smoking Detection

For the specific task of Smoking detection, the YOLOv10 model was fine-tuned using a custom dataset comprising 2,800 images sourced from Roboflow. The dataset underwent extensive augmentation, resulting in 5,473 images used for training. Augmentation techniques included horizontal flipping, rotation, scaling, and color adjustments to enhance the model's robustness to various conditions. The training process involved:

Dataset Collection

The initial step in fine-tuning YOLOv10 for Smoking detection involves collecting a comprehensive dataset. For this project, a dataset comprising 2,800 images was collected from various sources, focusing on individuals Smoking in different environments. This dataset was augmented to enhance the model's ability to generalize to new, unseen data. Augmentation techniques included rotation, scaling, and flipping to increase the diversity and robustness of the training set.

Data Annotation

Once the dataset was collected, it was annotated using tools like Roboflow. Each image was labeled with bounding boxes around the Smoking. Accurate annotations are crucial for the effectiveness of the YOLOv10 model, as they directly impact the model's learning process.

Data Preprocessing

Before feeding the data into the model, several preprocessing steps were applied:

- **Resizing**: All images were resized to a uniform dimension to ensure consistency and compatibility with the YOLOv10 architecture.
- **Normalization**: Pixel values were normalized to fall within a specific range, typically [0, 1], to facilitate faster convergence during training.
- **Augmentation**: In addition to the initial augmentation during dataset creation, further augmentation techniques like color jittering, cropping, and horizontal flipping were applied during training to improve the model's robustness.

Model Architecture and Configuration

YOLOv10, known for its efficiency and accuracy, was chosen for its advanced features and performance benefits. The architecture includes several enhancements:

- **NMS-Free Training**: Utilizing consistent dual assignments to eliminate the need for Non-Maximum Suppression, reducing inference latency.
- **Holistic Model Design**: Optimized components such as lightweight classification heads, spatial-channel decoupled downsampling, and rank-guided block design.
- **Enhanced Capabilities**: Incorporating large-kernel convolutions and partial self-attention modules for improved performance.

Training Procedure

The training process involved several key steps:

- **Initialization**: The YOLOv10 model was initialized with pre-trained weights from a large-scale dataset (e.g., COCO) to leverage transfer learning and accelerate convergence.
- **Hyperparameter Tuning**: Hyperparameters such as learning rate, batch size, and number of epochs were carefully tuned to optimize the model's performance. Techniques like learning rate scheduling and early stopping were employed to prevent overfitting and ensure efficient training.
- **Training**: The model was trained using a combination of one-to-many and one-to-one label assignment strategies. The loss function included components for bounding box regression, object classification, and confidence scores. The training was conducted on GPUs to handle the computational demands and speed up the process.
- **Validation**: A separate validation set was used to monitor the model's performance during training. Metrics such as mean Average Precision (mAP), precision, recall, and F1 score were tracked to evaluate the model's accuracy and generalization capability.

Evaluation and Fine-Tuning

After the initial training, the model's performance was evaluated on a test set. Based on the evaluation results, further fine-tuning was conducted:

- **Model Refinement**: Adjustments were made to the model architecture, hyperparameters, and training data to address any observed weaknesses or biases.

- **Post-Processing**: Techniques like confidence thresholding and Non-Maximum Suppression (NMS) were applied to the model's predictions to refine the detection results and reduce false positives.

By following this detailed fine-tuning process, the YOLOv10 model was effectively adapted to meet the specific requirements of detecting Smoking within the surveillance system, ensuring high accuracy and efficiency.

4.3 WEB APPLICATION DEVELOPMENT

4.3.1 Backend Development

System Backend Implementation

Tools and Technologies Used

The backend of our surveillance system is built using the following technologies:

- **Flask**: Used for developing the web application's backend.
- **FastAPI**: Utilized for implementing the APIs for our computer vision models, email notifications, and fetching and updating Firebase Storage.

Backend Workflow

User Authentication and Session Management

- **Login**: Users log in using a verified email and password. The `face_recognition` library is used for facial recognition verification.
- **Session Handling**: Flask manages user sessions to ensure secure access to the web application features.

Live Monitoring and Detection

- **Frame Capture**: The system captures frames from connected cameras.
- **API Calls**: For each frame, the system sequentially calls the dress code, violence, and Smoking detection APIs.
- **Threshold Verification**: If any detection surpasses a 75% confidence level, the system initiates incident handling.
- **Incident Handling**:
  - **Video Capture**: Upon detection, the system saves a 30-second video of the incident in Firebase Storage.

- **Snapshot Creation**: A snapshot of the incident is extracted and sent to the face recognition API using FastAPI.
- **Face Recognition**: The API identifies students in the snapshot using DeepFace.
- **Student Identification**: Returns the student ID if a match is found in the database.
- **Report Generation**: Creates an incident report with the following details:
  * **Student Info**: Includes student's personal information.
  * **Student Personal Photo**: Photo of the student involved in the incident.
  * **Date and Time**: Timestamp of when the incident occurred.
  * **Snapshot of the Incident**: Image capturing the moment of the incident.
  * **Description of the Incident**: Provides a brief description or classification of the incident type.
- **Email Notification**: Calls the email API to send the incident report to the student's email.

User Management

- **Security Guards**: Access Live and Reports tabs to monitor real-time detections and manage reports.
- **Admins**: Full access to all tabs, including user management functionalities such as adding, deleting, and editing user information.

Recordings and Reports

- **Recordings Tab**: Users can view and select specific detection recordings.
- **Reports Tab**: Security guards can review reports, send incident emails, and download detailed reports.

4.3.2 Frontend Development

The frontend of our surveillance system's web application is designed to provide a user-friendly interface for interacting with the system's features. It is developed using a combination of HTML, CSS, JavaScript, and Bootstrap to ensure a responsive and interactive experience.

HTML

HTML (HyperText Markup Language) is used to structure the content of the web pages. It defines the layout of the application, including elements like headings, paragraphs, buttons, and forms. Each page within the application, such as the Home, About Us, Demo, Contact Us, and Documentation tabs, is constructed using HTML to create a cohesive and navigable structure.

CSS

CSS (Cascading Style Sheets) is used to style the HTML elements, providing a visually appealing design. It handles the layout, colors, fonts, and overall aesthetics of the web application. By utilizing CSS, we ensure that the application is not only functional but also attractive and consistent across different devices and screen sizes.

JavaScript

JavaScript is employed to add interactivity and dynamic behavior to the web application. It enables functionalities such as form validation, asynchronous requests to the backend APIs, and real-time updates to the user interface. For instance, the detection logs in the Live tab are dynamically updated using JavaScript to reflect the latest events captured by the system.

Bootstrap

Bootstrap is a popular CSS framework that helps create responsive and mobile-first web pages. It provides pre-designed components and a grid system that simplifies the development of a flexible layout. By leveraging Bootstrap, we ensure that the web application adapts to various screen sizes, from desktops to mobile devices, enhancing the user experience.

Features and Navigation

The web application features a navigation bar with five main tabs:

1. **Home**: This tab provides an overview of the project's features, frameworks, and benefits. It serves as the landing page for users to understand the purpose and capabilities of the system.
2. **About Us**: This section includes information about the team, our mission, vision, and goal. It introduces the people behind the project and our objectives.
3. **Demo**: In this tab, users can log in with a verified email account and password, followed by face recognition verification. Once logged in, users can access different functionalities of the system.

4. **Contact Us**: This tab allows users to request an account to try the demo system. It includes a form for users to provide their contact information and reason for requesting access.
5. **Documentation**: Here, users can download the full documentation of the project and watch a demonstration video. It serves as a resource for understanding the technical details and implementation of the system.

Interaction with Backend

The frontend interacts with the backend APIs to perform various actions. For example:

- **Live Feed**: The Live tab displays a camera feed with detection logs. Users can visualize detections with buttons for each module, and the logs are updated in real-time using JavaScript.
- **Recordings**: The Recordings tab allows users to view and select specific detections from the logs. The recordings include a few seconds before and after each detection, providing context for the events.
- **Reports**: In the Reports tab, security guards can review student reports for detections, send incident emails with additional information, and download reports. This tab facilitates communication and record-keeping for security personnel.
- **User Management**: The Users tab displays all security guards' information, allowing for adding, deleting, and editing user info. The Admin tab lets admins modify their info, delete accounts, or change passwords.

By combining these technologies and features, the frontend of our web application ensures an intuitive and efficient user experience, enabling users to effectively interact with the surveillance system.

4.3.3 Features and Navigation

Our surveillance system's web application is designed with a user-friendly interface to allow easy access to its various features. The navigation bar includes the following tabs: Home, About Us, Demo, Contact Us, and Documentation. Once logged in, users can access additional functionalities through the tabs: Live, Recordings, Reports, Users, and Admin.

Home

The Home tab provides an overview of the project's features, frameworks, and benefits. It serves as the landing page for users to understand the purpose and capabilities of the system. The main sections of the Home tab include:

- **Project Overview**: A brief description of the surveillance system, its applications, and key features.
- **Technologies Used**: An outline of the frameworks and technologies employed in developing the system, such as DeepFace, YOLOv10, Flask, FastAPI, and Firebase.
- **Benefits**: A summary of the advantages offered by the system, including enhanced security, real-time monitoring, and automated incident reporting.

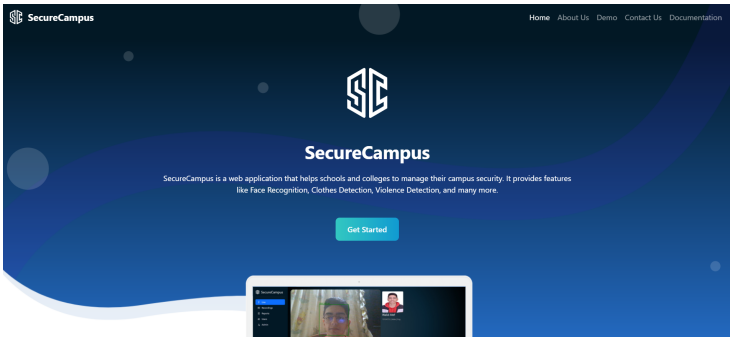


Figure 4.1: Home Page

About Us

The About Us tab includes information about the development team, our mission, vision, and goals. It introduces the people behind the project and our objectives. This section is divided into:

- **Team Information**: Details about the team members, their roles, and contributions to the project.
- **Mission and Vision**: Statements defining the purpose and long-term aspirations of the project.
- **Goals**: Specific objectives we aim to achieve with the surveillance system.

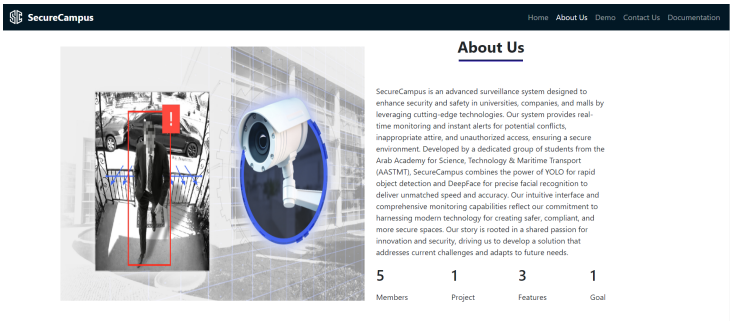


Figure 4.2: About Us Page

Demo

The Demo tab allows users to log in with a verified email account and password, followed by face recognition verification. Once logged in, users can access the core functionalities of the system. The demo includes:

- **Login**: A secure login page requiring email and password authentication.
- **Face Recognition Verification**: Additional security using face recognition to verify user identity.
- **Feature Access**: Once authenticated, users can explore the Live feed, view Recordings, generate Reports, manage Users, and access Admin settings.

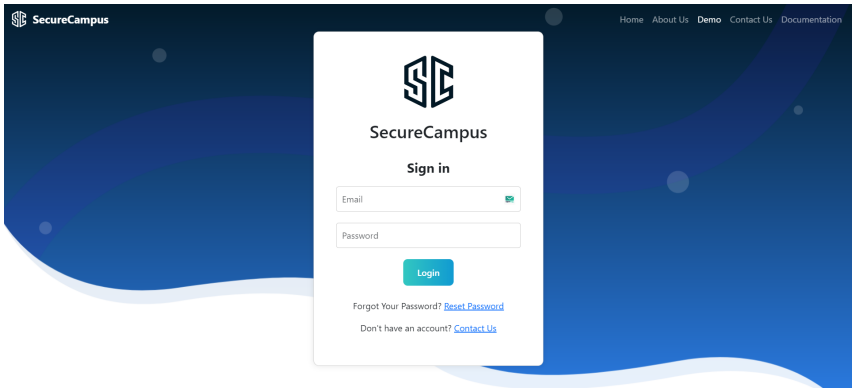


Figure 4.3: Login Page

Contact Us

The Contact Us tab allows users to request an account to try the demo system. It includes a form for users to provide their contact information and reason for requesting access. This tab ensures that only authorized users can access the demo features.

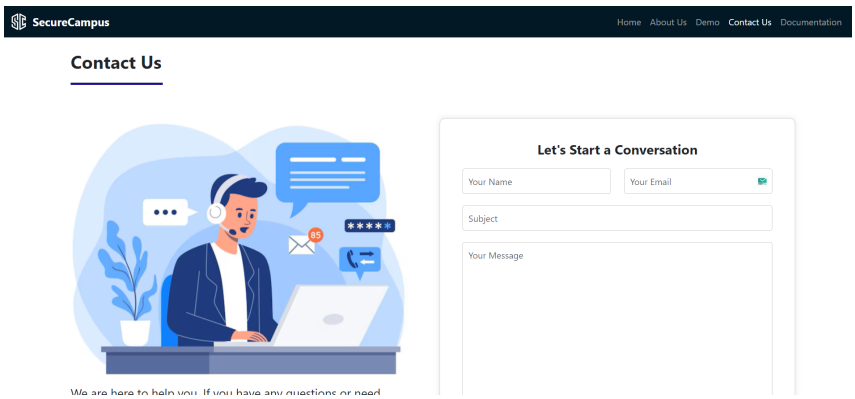


Figure 4.4: Contact Us Page

Documentation

The Documentation tab offers downloadable documentation and a demonstration video. This section provides comprehensive resources for understanding the technical details and implementation of the system. It includes:

- **Project Documentation**: Detailed documents covering various aspects of the system, including setup, usage, and development.
- **Demonstration Video**: A video walkthrough showcasing the system's features and functionalities.

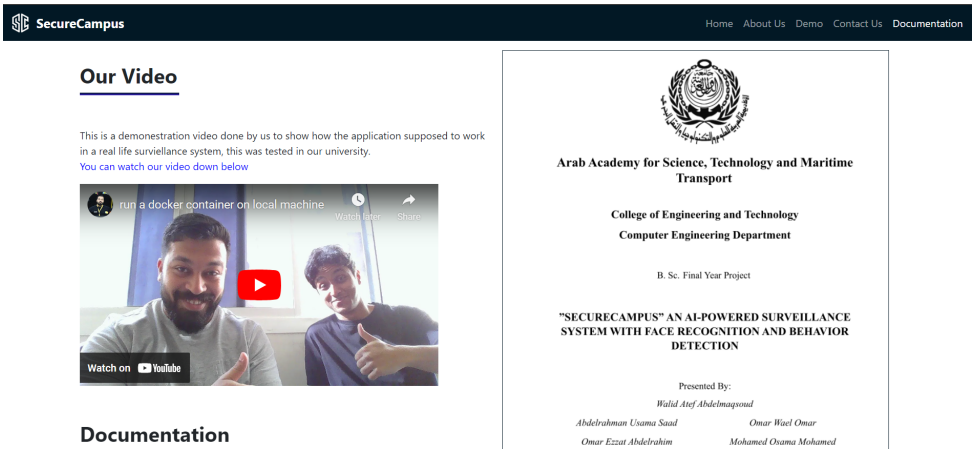


Figure 4.5: Documentation Page

Live

The Live tab displays a camera feed with detection logs. Users can visualize detections with buttons for each module. The logs, updated in real-time, show the latest events captured by the system. Key features include:

- **Camera Feed**: Live video feed from connected cameras.
- **Detection Logs**: Real-time logs of detected events, searchable by users.
- **Module Buttons**: Options to filter detections based on specific modules such as violence, dress code, and Smoking detection.

Figure 4.6: Live Page

Recordings

The Recordings tab allows users to view recordings of the latest detections and select specific ones from the logs. Recordings include a few seconds before and after each detection, providing context for the events. Features include:

- **Latest Detections**: A list of recent detections with corresponding recordings.
- **Detailed Recordings**: Access to specific recordings, showing events before and after detections.
- **Log Search**: Functionality to search and filter detection logs to find specific events.

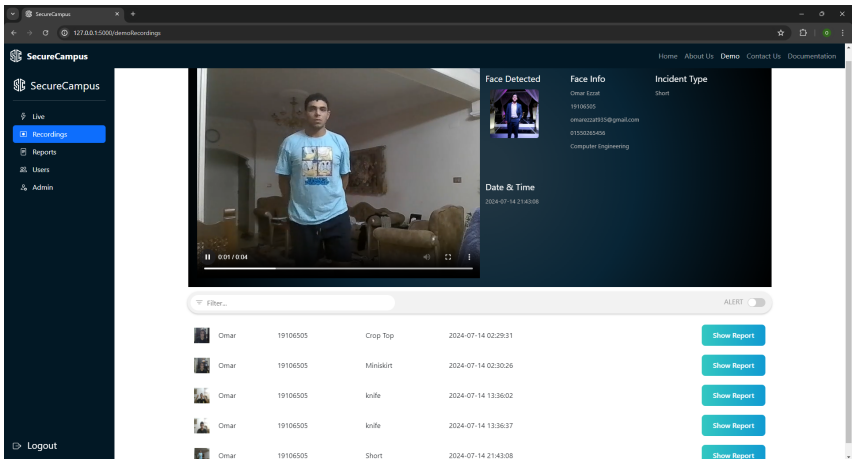


Figure 4.7: Recordings Page

Reports

The Reports tab enables security guards to review student reports for detections, send incident emails with additional information, and download reports. This tab facilitates communication and record-keeping for security personnel. Features include:

- **Incident Review**: Detailed review of incidents, including screenshots and student information.
- **Email Notifications**: Ability to send incident emails to students with additional details.
- **Report Downloads**: Option to download comprehensive incident reports for documentation and analysis.

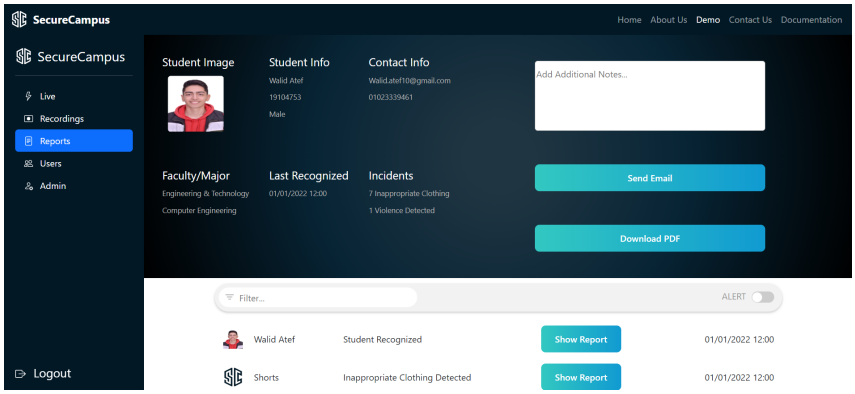


Figure 4.8: Reports Page

Users

The Users tab displays all security guards' information, allowing for adding, deleting, and editing user info. This section ensures proper user management within the system. Features

include:

- **User List**: A comprehensive list of all registered security guards.
- **User Management**: Options to add, edit, and delete user information.
- **Access Control**: Management of user access levels and permissions.

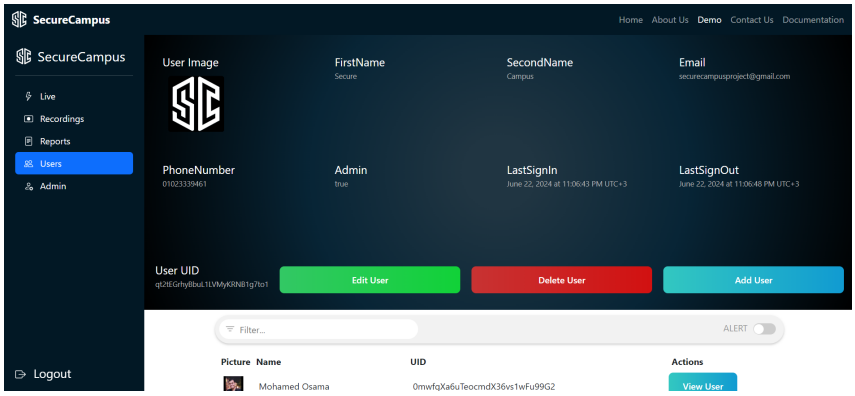


Figure 4.9: Users Page

Admin

The Admin tab lets admins modify their info, delete accounts, or change passwords. Admins have full access to all tabs, while normal security guards only have access to the Live and Reports tabs. Features include:

- **Admin Settings**: Options to modify admin information and account settings.
- **Account Management**: Ability to delete accounts and reset passwords.
- **Full Access**: Comprehensive access to all features and settings within the web application.

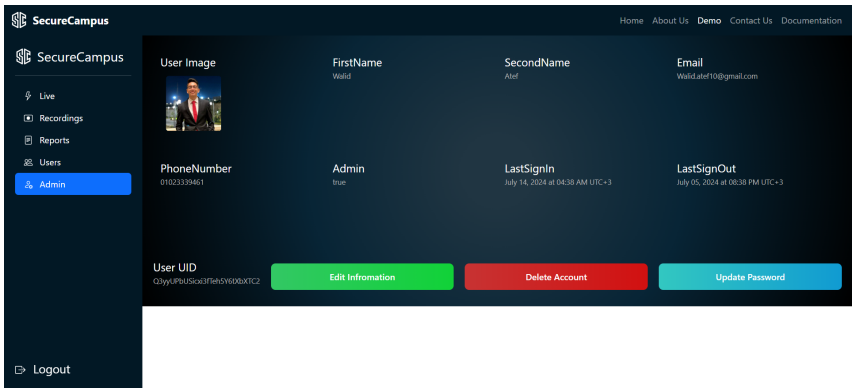


Figure 4.10: Admin Page

4.4 DATABASE AND STORAGE

4.4.1 Firestore and Realtime Database

In our surveillance system, we utilized Firebase's Firestore and Realtime Database to handle different aspects of data storage and management effectively.

Firestore

In our surveillance system, Firestore is used to store user and student databases. The user database includes information about security guards and admins, while the student database contains student profiles, including their IDs, photos, and incident history. Firestore's structured data storage and real-time synchronization ensure that our system can manage and access user and student information efficiently and reliably.

Realtime Database

In our system, the Realtime Database is used to store and manage incident detection logs. Whenever one of our models detects a violation (violence, inappropriate clothing, or Smoking), the detection information, including the timestamp and confidence level, is immediately saved in the Realtime Database. This ensures that security guards have instant access to the latest incident reports and can respond promptly.

Integration and Workflow

Our system leverages both Firestore and Realtime Database to provide a comprehensive and efficient solution for managing and storing different types of data:

- **Firestore** handles user and student profiles, ensuring scalable and reliable storage with advanced querying capabilities.
- **Realtime Database** manages incident detection logs, providing low-latency, real-time data synchronization.

By combining the strengths of Firestore and Realtime Database, our surveillance system achieves optimal performance, scalability, and real-time data handling. This integrated approach ensures that security guards have timely access to critical information, enhancing the overall effectiveness and responsiveness of our surveillance solution.

4.4.2 Storage for Photos and Recordings

In our surveillance system, efficient storage and management of photos and recordings are critical for accurate monitoring and incident reporting. We utilized Firebase Storage for handling these tasks due to its robustness and scalability.

Integration and Workflow

Our system captures frames from the connected cameras and processes them to detect incidents. When an incident is detected, the relevant frames and recordings are stored in Firebase Storage. The workflow for handling photos and recordings is as follows:

- **Incident Detection:** When one of our models (violence detection, inappropriate clothing detection, or Smoking detection) identifies a violation, it triggers the storage process.
- **Frame Capture:** The system captures the frame in which the incident is detected along with a few seconds of video before and after the detection.
- **Storage:** The captured frame and recording are uploaded to Firebase Storage. Each file is stored with a unique identifier and metadata, including the timestamp and type of incident.
- **Access and Retrieval:** The stored files are accessible through the web application. Security guards can view live detections, search logs for specific incidents, and retrieve relevant photos and recordings. The system ensures that all files are easily retrievable and viewable within the application.

By leveraging Firebase Storage, our surveillance system ensures that all photos and recordings are securely stored, easily accessible, and efficiently managed, enhancing the overall functionality and reliability of our monitoring solution.

4.5 DEPLOYMENT

4.5.1 Microservices Implementation Details

In the implementation phase of our surveillance system, we detailed how each microservice was designed, developed, and integrated to achieve our system's goals. Here's a breakdown of the implementation details for key microservices:

Face Detection Service

- **Technologies Used:** DeepFace and face_recognition libraries for facial recognition tasks.
- **Architecture:** Deployed as a standalone microservice with RESTful API endpoints for user authentication and incident identification.
- **Integration:** Interacts with other microservices such as the main application backend and database service to manage user identities and incident data.

Violence Detection Service

- **Technologies Used:** Machine learning models for real-time violence detection in video streams.
- **Architecture:** Implemented as an independent microservice, providing APIs for detecting violent activities and alerting security personnel.
- **Integration:** Integrated with the main application backend and frontend to display alerts and manage security incidents.

Inappropriate Clothing Detection Service

- **Technologies Used:** YOLOv10 for detecting prohibited attire such as shorts, crop tops, skirts, and ripped jeans.
- **Architecture:** Deployed as a microservice with specialized APIs for analyzing video feeds for dress code violations.
- **Integration:** Communicates with the main application backend to report violations and trigger notifications to security personnel.

Email Notification Service

- **Technologies Used:** SMTP protocols and email automation libraries for sending incident notifications.
- **Architecture:** Implemented as a standalone microservice, providing reliable communication channels for alerting students and administrators.
- **Integration:** Integrated with detection services to trigger notifications based on detected incidents, ensuring timely communication.

Database Service

- **Technologies Used:** Firebase for managing user, student, and incident data.
- **Architecture:** Integrated with Firebase Firestore and Realtime Database, offering APIs for storing and retrieving data across microservices.
- **Integration:** Interacts with all microservices to maintain data consistency and provide real-time updates on incidents and user information.

Main Application Service (Backend and Frontend)

- **Technologies Used:** Flask and FastAPI for backend development, HTML, CSS, JavaScript, and Bootstrap for frontend development.

- **Architecture:** Acts as the central service for managing user interfaces, data flow, and interactions with other microservices.
- **Integration:** Exposes RESTful APIs for frontend interactions and communicates with backend microservices (face detection, violence detection, etc.) to fetch data and process user requests.

By detailing the implementation of each microservice, including the integrated main application service, we ensure clarity on how technology choices were applied to achieve specific system functionalities. This approach enhances understanding of the system's architecture, integration points, and deployment strategies.

4.5.2 Implementing APIs Containerization with Docker

Steps to Create Docker Images for Each API

To containerize each API as Docker images, follow these steps:

- **Define Dockerfiles:** Create Dockerfiles for each API to specify the environment setup, dependencies installation, and application deployment.
- **Build Docker Images:** Use Docker build commands to build reproducible images encapsulating each API and its dependencies.
- **Run Docker Containers:** Deploy Docker containers from the built images using Docker run commands to ensure consistent behavior across different deployment environments.

Benefits of Containerizing the APIs

Containerizing the APIs with Docker offers several advantages for our surveillance system:

- **Consistency:** Docker ensures consistent API performance across various environments, eliminating issues related to dependencies and configuration mismatches.
- **Isolation:** Each Docker container isolates an API and its dependencies, ensuring independent operation without interference from other services.
- **Scalability:** Docker's ability to horizontally scale containers allows our system to handle increased traffic and workload demands efficiently.
- **Efficiency:** Containerized APIs are lightweight and share resources, optimizing resource utilization and enabling fast startup times.
- **Portability:** Docker's standardized format facilitates easy deployment and migration of containerized APIs across different infrastructure environments.

- **DevOps Integration:** Docker integrates seamlessly with CI/CD pipelines, supporting automated software builds, tests, and deployments for enhanced development efficiency.

By implementing Docker containerization for our APIs, we enhance the agility, scalability, and reliability of our surveillance system, ensuring optimal performance and efficient resource utilization in diverse deployment scenarios.

4.5.3 Implementing Kubernetes

Setting Up a Local Kubernetes Cluster

We opted to deploy our Kubernetes cluster locally rather than on the cloud for reasons related to latency, real-time performance, and cost management.

Steps to Set Up a Local Kubernetes Cluster Using Kubeadm

Prerequisites Before setting up the Kubernetes cluster with Kubeadm, ensure the following prerequisites are met:

- Minimum hardware requirements for master and worker nodes, including CPU, RAM, and disk space.
- A network connection between the nodes.
- Installation of Kubernetes components (kubeadm, kubelet, and kubectl) and Docker on all nodes.

Using Vagrant for Infrastructure as Code (IaC) Vagrant was used to provision and manage virtual machines, ensuring consistency across development environments and simplifying cluster setup.

- **Define Vagrantfile:** Configure virtual machines with the required specifications, including base image, network settings, and provisioning scripts.
- **Provisioning:** Use Vagrant to automate the installation of Kubernetes components on master and worker nodes.

Master Node Setup

1. **Initialize the Cluster:** Use Kubeadm to initialize the Kubernetes cluster on the master node, specifying the pod network CIDR.
2. **Configure Kubectl:** Set up kubectl to manage the cluster using the admin.conf file generated during cluster initialization.

3. **Deploy Pod Network Add-On:** Install a pod network add-on (e.g., Calico or Flannel) to enable communication between Pods.

Worker Node Setup

1. **Join Nodes to Cluster:** On each worker node, join them to the Kubernetes cluster using the join command provided by Kubeadm during cluster initialization.
2. **Verify Cluster Status:** Confirm the successful addition of worker nodes to the Kubernetes cluster using kubectl commands from the master node.

Deploying Containers in Kubernetes

Creating Kubernetes Manifests

Each containerized application in our surveillance system is defined in Kubernetes manifests, specifying Pods, Services, and Deployments.

Example Deployment Process To deploy a containerized application (e.g., face detection API) in Kubernetes:

1. **Define Deployment Manifest:** Create a Deployment YAML file specifying container image, replicas, and other configurations.
2. **Define Service Manifest:** Create a Service YAML file to expose the Deployment, defining ports and load balancing policies.
3. **Apply Manifests:** Use kubectl apply to deploy Pods, Services, and Deployments defined in the YAML files to the Kubernetes cluster.

Managing Kubernetes Objects

Pods Pods in Kubernetes represent the smallest unit of deployment, encapsulating one or more containers. Each API in our system runs within its own Pod to ensure isolation and scalability.

Services Services provide a stable endpoint for accessing Pods and enable load balancing across multiple instances of an application. They facilitate communication between components in our surveillance system.

Deployments Deployments manage the lifecycle of Pods, ensuring a specified number of replicas are running at all times. They support rolling updates and allow us to scale our applications seamlessly.

4.5.4 Benefits of Kubernetes for our Surveillance System

Load Balancing

Kubernetes provides built-in load balancing to evenly distribute traffic across multiple Pods running our surveillance system's APIs. This enhances performance and ensures efficient resource utilization.

Scalability

Kubernetes supports horizontal scaling of applications by increasing or decreasing the number of Pod replicas based on demand. This capability allows us to handle fluctuations in traffic and workload effectively.

Conclusion

By implementing Kubernetes for container orchestration in our surveillance system, we achieve improved scalability, reliability, and operational efficiency. Kubernetes simplifies the deployment and management of containerized applications, supports automated scaling and self-healing, and ensures high availability of our critical services. The detailed setup process using Kubeadm for local deployment and the configuration of Kubernetes objects illustrate our approach to leveraging this powerful orchestration platform.

Chapter Five

5 RESULTS AND FINDINGS

5.1 OBJECT DETECTION

5.1.1 Approach

To fortify the object detection capabilities within SecureCampus, we adopted a systematic approach centered around fine-tuning the YOLOv8 large model. The training process was conducted on a custom dataset meticulously curated in Roboflow. Our initial efforts yielded a respectable 74% accuracy, prompting a thorough investigation into dataset modifications.

Recognizing the pivotal role of dataset quality in model performance, we undertook a series of refinements. These adjustments, coupled with meticulous dataset augmentation techniques, propelled the model's accuracy to an impressive 85%. Eager to enhance performance further, an exhaustive augmentation strategy was employed, leading to a remarkable accuracy of 94.3% with 0.5742 Loss.

In an exploratory phase, the model was trained for an extended 150 epochs, resulting in an exceptional accuracy of 97% with 0.3802 loss. However, considering the potential risk of overfitting, particularly in the 'abandoned bag' class, a deliberate decision was made to favor the model with 94.3% accuracy. This choice was rooted in the model's superior generalization during testing, affirming its reliability in real-world scenarios.

5.1.2 Results

The performance of the object detection model is vividly portrayed through various visualizations and metrics:

Fig 5.1 illustrates the confusion matrix, providing a detailed breakdown of the model's predictions across different classes.

Fig 5.2 showcases the accuracy and loss curves over the training epochs, offering insights into the model's learning trajectory.

Fig 5.3 presents notable results on the test set, showcasing the model's efficacy in real-world scenarios.

Table 5.1 tabulates the accuracies for each class, offering a comprehensive overview of the model's performance across diverse object categories.

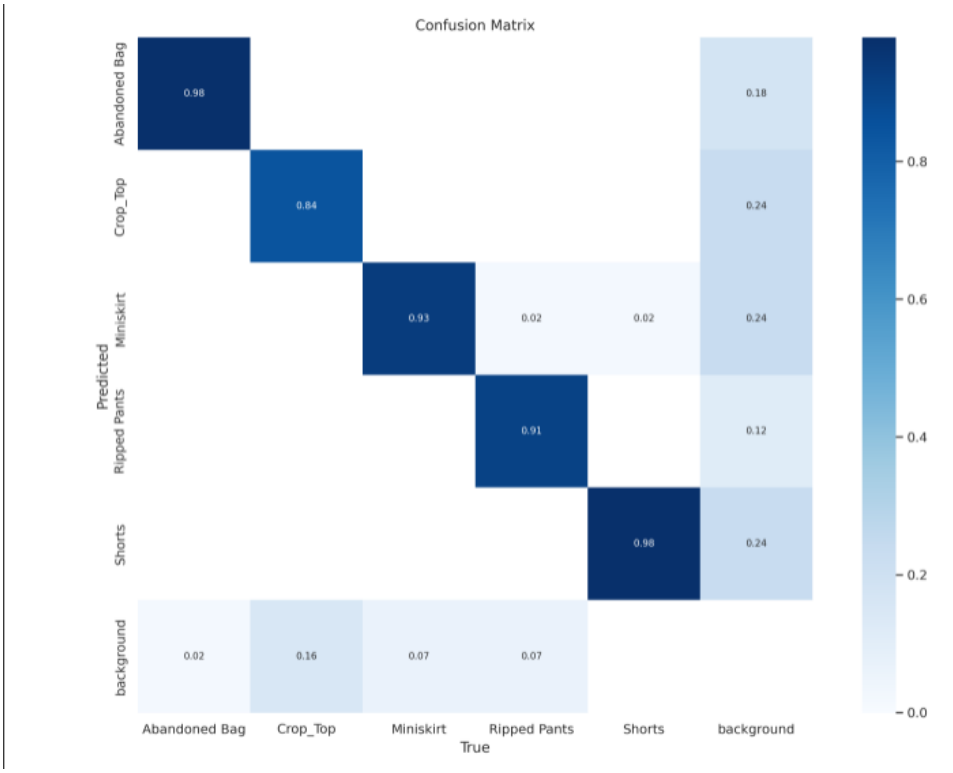


Figure 5.1: Object Detection Confusion Matrix

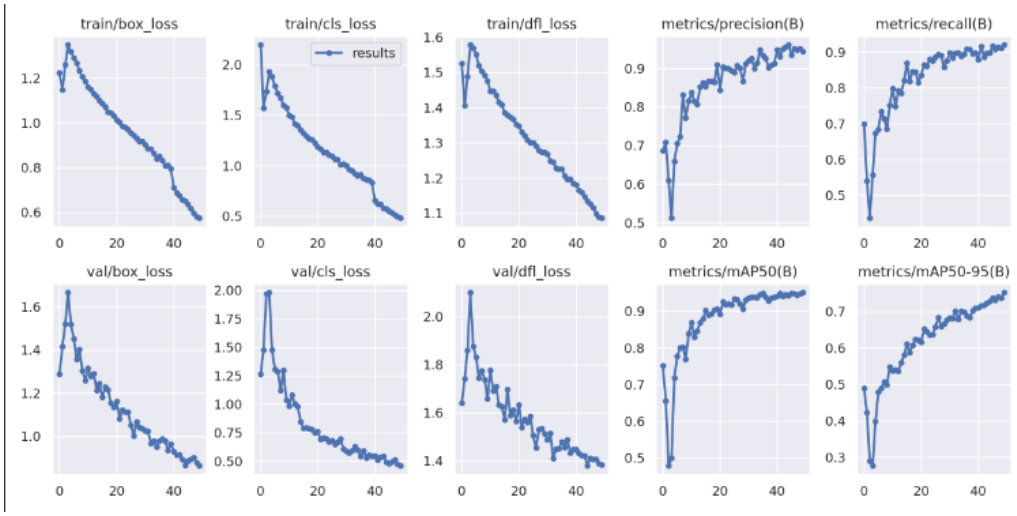


Figure 5.2: Object Detection Acc and Loss Curves

Through these comprehensive visualizations and metrics, the results section provides a nuanced understanding of the object detection model's performance, emphasizing both its overall accuracy and its efficacy across specific classes

Figure 5.3: Object Detection Test Set

Table 5.1: Object Detection Results.

Class	Box Percision	Recall	mAP-50
All	0.943	0.921	0.951
Abandoned Bag	0.96	0.977	0.994
Crop_Top	0.916	0.845	0.881
MiniSkirt	0.97	0.945	0.973
RippedPants	0.947	0.913	0.932
Shorts	0.922	0.924	0.975

5.2 FACE RECOGNITION

5.2.1 Approach

face_recognition Approach:

Recognized for its simplicity and ease of use, face_recognition provides a high-level API that abstracts away many complexities, ensuring user-friendly interactions. Leveraging the dlib library, this approach employs a traditional method based on Histogram of Oriented Gradients (HOG) features and a linear Support Vector Machine (SVM) classifier. While suitable for straightforward scenarios, it may exhibit limitations in complex environments. With minimal dependencies, primarily relying on dlib, installation and setup are straightforward. However, customization options are limited, making it ideal for users prioritizing simplicity and ease of use. Benefitting from a decent-sized community and well-documented resources, face_recognition ensures support and troubleshooting accessibility.

deepface with VGG Approach:

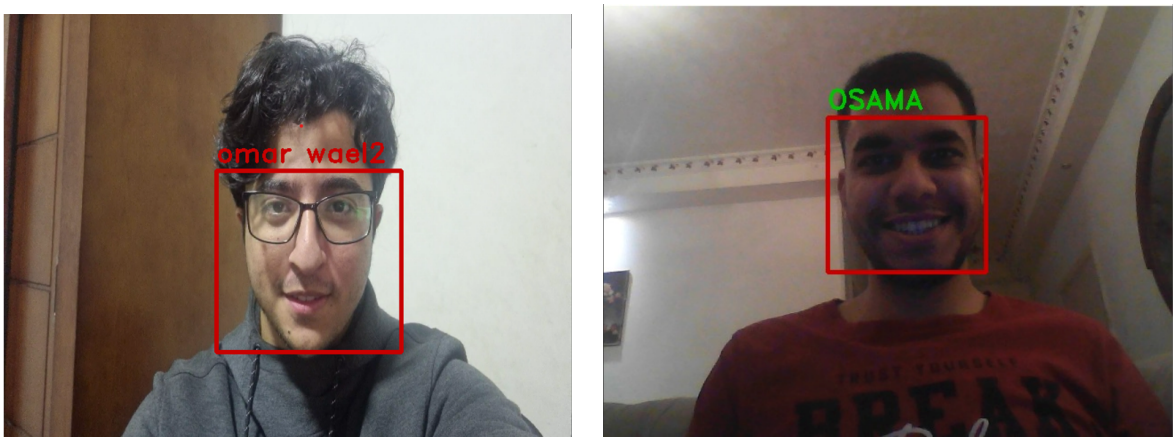
While aiming to simplify deep learning-based face recognition, deepface introduces complexity, especially for beginners, by utilizing the VGG-Face model, a deep convolutional neural network (CNN). Capable of handling complex scenarios and diverse datasets, it provides superior performance in challenging conditions. However, its dependencies include deep learning frameworks like TensorFlow or Keras like in [23] in the references due to VGG-Face usage, potentially requiring more involved installation and setup. Offering greater flexibility for customization, deepface enables fine-tuning or the use of alternative deep learning models for specific requirements. Although the community might be smaller, and documentation might not be as extensive, deepface with VGG is a robust option for tasks demanding higher accuracy and intricate face recognition within SecureCampus.

5.2.2 Results

In the face recognition results section, visual representations showcase the efficacy of both approaches. Two compelling photos, each representing one of the distinctive methods, provide a tangible glimpse into the performance and capabilities of face_recognition and deepface with VGG in recognizing faces within the SecureCampus surveillance system. These visual representations aim to highlight the nuanced differences between the two approaches, aiding in the selection of the most suitable face recognition methodology for specific project requirements.

face_recognition Results:

Fig 5.4 presents notable results on the face_recognition library, showcasing the model's efficacy in real-world scenarios.



Omar Wael face_recognition Mohamed Osama face_recognition

Figure 5.4: face_recognition results

deepface Results:

Fig 5.5 presents notable results on the deepface library, showcasing the model's efficacy in real-world scenarios.

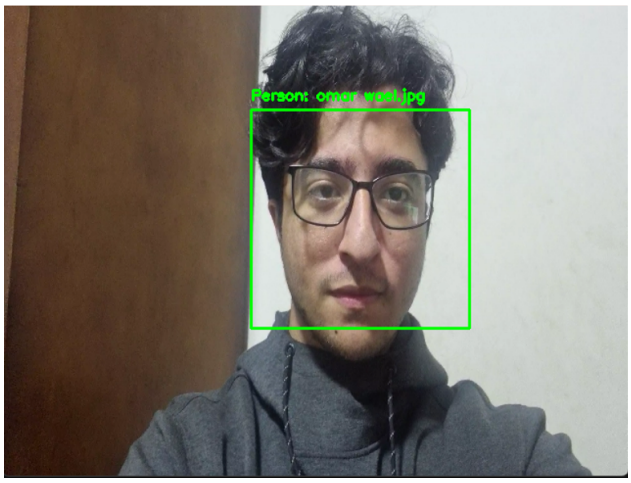


Figure 5.5: deepface results

5.3 VIOLENCE DETECTION

5.3.1 Approach

In tandem with the approach employed for object detection, the violence detection aspect of SecureCampus hinged on fine-tuning the YOLOv8 large model. Leveraging the same model architecture and training for 50 epochs, mirroring the object detection methodology, the process incorporated a custom dataset generated through Roboflow. Augmentations were systematically applied to enrich the dataset, fostering a nuanced understanding of diverse behaviors.

This parallel training trajectory resulted in a commendable performance, showcasing a behavior detection model with a notable accuracy of 93.4% and a corresponding loss of 0.7275. The meticulous application of dataset augmentations contributed to refining the model's ability to discern and categorize behaviors accurately.

5.3.2 Results

The performance of the behavior detection model is succinctly depicted through key visualizations and metrics:

Fig 5.6 illustrates the confusion matrix, offering a detailed breakdown of behavior predictions across different classes.

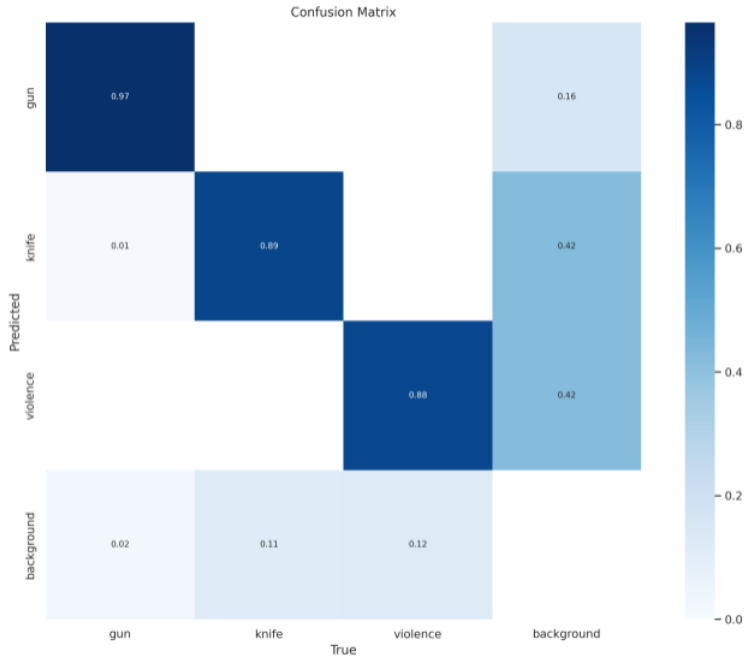
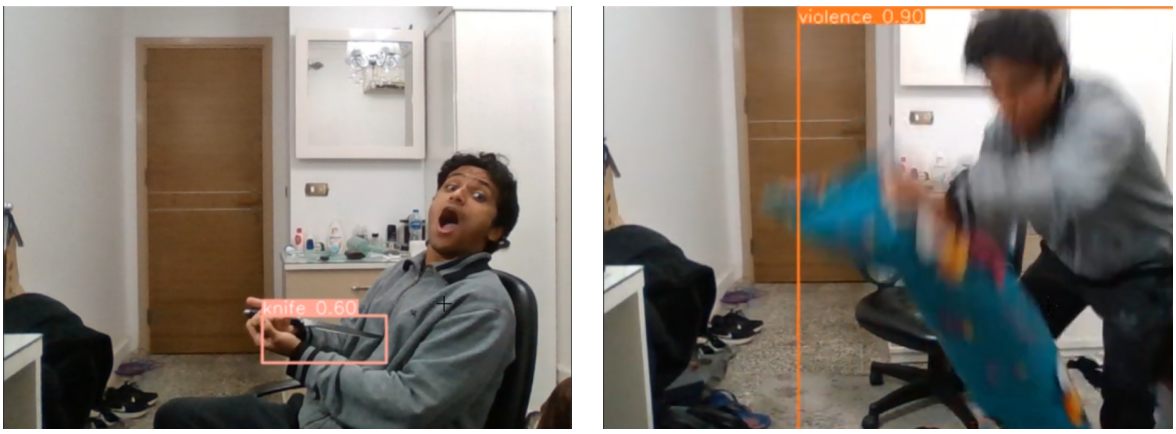


Figure 5.6: Violence Confusion Matrix

Fig 5.7 presents notable results on the webcam, showcasing the model's efficacy in real-world scenarios.



Abdelrahman Usama Knife Abdelrahman Usama Violence

Figure 5.7: Violence detection results

Table 5.2 tabulates the accuracies for each violence class, providing a comprehensive overview of the model's performance.

Table 5.2: Violence Results.

Class	Box Percision	Recall	mAP-50
All	0.934	0.881	0.948
gun	0.979	0.96	0.988
knife	0.909	0.849	0.921
violence	0.913	0.833	0.936

Through these succinct visualizations and metrics, the results section sheds light on the performance of the violence detection model, emphasizing its accuracy and proficiency in discerning and categorizing various behaviors.

5.4 ADDITIONAL FEATURES AND INTEGRATION

5.4.1 Face Detection and Recognition

SecureCampus integrates **face_recognition** for user logins and **DeepFace with VGG** for incident identification. The **face_recognition** library, used for authentication, ensures straightforward and reliable login processes. **DeepFace with VGG**, employed for identifying students during incidents, provides high accuracy and robust performance under varying conditions.

5.4.2 Smoking Detection

The system includes a model for detecting Smoking :

Smoking Detection: A YOLOv10 model fine-tuned with 2,800 images reached an accuracy of 86%, effectively identifying Smoking activities.

Table 5.3 summarizes the results for the Smoking detection model.

Table 5.3: Smoking Detection Results.

Model	Dataset Size	Accuracy
Smoking Detection	2,800 images	86%

5.4.3 Web Application and Workflow

The SecureCampus web application, developed using Flask and FastAPI for the backend and HTML, CSS, JavaScript, and Bootstrap for the frontend, includes various tabs for user interaction:

- **Home Tab:** Features the project's capabilities, frameworks, and benefits.
- **About Us Tab:** Displays team information, mission, vision, and goals.
- **Contact Us Tab:** Provides access to request demo accounts.
- **Documentation Tab:** Contains downloadable documentation and a demo video.

In the demo section:

- **Live Tab:** Shows live camera feeds with detection logs and visualizations.
- **Recordings Tab:** Accesses recordings of recent detections with pre- and post-detection footage.
- **Reports Tab:** Allows security personnel to manage and review incident reports, send emails, and download reports.

- **Users Tab:** Manages security guard information with options to add, delete, or edit details.
- **Admin Tab:** Provides admins with capabilities to modify info, delete accounts, or change passwords.

5.4.4 Workflow

The system workflow integrates several components:

- **Frame Capture:** Cameras capture frames, which are processed by APIs for clothing, violence, and Smoking detection models.
- **Violation Detection:** Frames with detected violations (confidence level ≥ 75%) are saved, and session recordings are created.
- **Face Recognition:** The face recognition API verifies the student's identity and retrieves their information from the database.
- **Incident Notification:** An email is sent to the student with incident details. The system supports student appeals via email.

The comprehensive integration of these features ensures a robust surveillance solution with effective detection capabilities and user-friendly management tools.

Chapter Six

6 CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

The SecureCampus project represents a significant advancement in surveillance and security systems, offering a comprehensive solution for various environments such as universities, companies, and malls. By integrating advanced technologies, the system delivers robust performance in real-time monitoring and incident management.

The face recognition component leverages the `face_recognition` library for user authentication and DeepFace with the VGG-Face model for detailed post-incident identification, ensuring reliable recognition of individuals involved in incidents. The violence detection feature, powered by YOLOv8, achieves a high accuracy of 94.3%,

effectively identifying potential altercations and enabling timely intervention by security personnel. Additionally, the system addresses dress code compliance and Smoking regulations through models fine-tuned with YOLOv10, achieving 97% accuracy for inappropriate clothing and 86% accuracy for Smoking detection.

The object detection model, initially achieving 74% accuracy, was refined through dataset modifications and augmentation to reach an impressive 97% accuracy, with a decision made to avoid potential overfitting by opting for the model with 97% accuracy. The integration of these models within a scalable microservices architecture,

using Docker and Kubernetes, ensures efficient deployment and load balancing. The web application, developed with Flask, FastAPI, and modern frontend technologies, provides a user-friendly interface for accessing live feeds, managing recordings, and generating reports. With Firebase for data storage and a well-defined workflow for capturing,

analyzing, and responding to incidents, SecureCampus stands out as a versatile and effective surveillance solution. The project not only addresses immediate security and compliance needs but also adheres to ethical AI practices, safeguarding privacy and ensuring responsible technology use. In conclusion, SecureCampus exemplifies our commitment to advancing security technologies while upholding high standards of privacy and operational integrity.

6.2 FUTURE WORK

While the SecureCampus project has achieved considerable success, there are several areas where further development and enhancement could improve its functionality and impact. Future work includes:

1. Real-Time Face Recognition Optimization:
Currently, face recognition is performed post-incident due to the computational demands of real-time processing. Future work will focus on optimizing algorithms and leveraging more efficient hardware or distributed processing techniques to enable real-time face recognition without compromising camera feed quality.

2. Expanding Dataset for Testing:
Given the limitations in accessing real university datasets for testing, future efforts will aim to acquire or simulate more diverse and representative datasets. This will enhance model robustness and improve the system's performance in real-world scenarios.

3. Enhanced Model Integration:
Further integration and optimization of the various models (face recognition, behavior detection, object detection) are planned. Improved synergy among these models will provide more accurate and contextually aware security responses.

4. Advanced User Interface Enhancements:
Future updates will refine the user interface to incorporate interactive visualizations and real-time analytics. Enhancements will focus on user experience, making it easier for security personnel to navigate and respond to incidents.

6. Strengthening Security Measures:
Ongoing improvements in security measures will be pursued, including advanced encryption methods and regular security assessments to protect against emerging threats and ensure data integrity.

7. Comprehensive User Training and Support:
As the system evolves, investing in enhanced training programs and support resources will be essential. This will ensure that users are well-equipped to utilize the system's full capabilities effectively.

By addressing these areas, SecureCampus will continue to evolve as a cutting-edge solution in AI-powered surveillance, adapting to new challenges and opportunities in the field of security technology.

REFERENCES

[1] Dat Tran, "Real-Time Object Detection with YOLO, YOLOv2 and now YOLOv3," *PyImageSearch*, pp. 1–10, 2018.

[2] Muhammad Waqas, "Real-Time Surveillance Through Face Recognition Using HOG and Feedforward Neural Networks," *King Abdulaziz University*, pp. 1–9, 2019, DOI: 10.1007/s00542-019-05192-8.

[3] Joonki Paik, "Real-Time Surveillance System for Analyzing Abnormal Behavior of Pedestrians," *Applied Sciences*, pp. 1–16, 2021, DOI: 10.3390/app11146325.

[4] Jiahong Wei, "Face Detection in Security Monitoring Based on Artificial Intelligence Video Retrieval Technology," *IEEE Access*, pp. 1–13, 2020, DOI: 10.1109/ACCESS.2020.3018213.

[5] René Ranftl et al., "YOLOv4: Optimal Speed and Accuracy of Object Detection," *arXiv preprint arXiv:2011.08036*, pp. 1–12, 2020.

[6] Virender Singh, "Real-Time Anomaly Recognition Through CCTV Using Neural Networks," *Pattern Recognition*, pp. 1–10, 2020, DOI: 10.1016/j.patcog.2020.107247.

[7] Guodong Guo et al., "Violence Detection in Surveillance Video Using a Hybrid Model," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 1–10, 2020, DOI: 10.1109/TCSVT.2020.3032823.

[8] K. A. Raut et al., "Real-Time Video Violence Detection Using CNN," *International Journal for Research in Applied Science and Engineering Technology*, pp. 1–8, 2023, DOI: 10.22214/ijraset.2023.4894.

[9] Y. Zhang et al., "Improved Real-Time Violence Detection System Using Deep Learning," *IEEE Access*, pp. 1–12, 2022, DOI: 10.1109/ACCESS.2022.3146215.

[10] R. S. Sharma et al., "YOLO-based Real-Time Violence Detection System for Smart Surveillance," *Journal of Visual Communication and Image Representation*, pp. 1–10, 2022, DOI: 10.1016/j.jvcir.2022.103034.

[11] M. Kim J. Lee K. Hwang, "Real-Time Smart Surveillance System for Crowd Monitoring and Violence Detection," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–12, 2023, DOI: 10.1109/TSMC.2023.3148985.

[12] Khalid Housni, "Deep Learning-Based Methods for Anomaly Detection in Video Surveillance," *ResearchGate*, pp. 1–15, 2022, DOI: 10.13140/RG.2.2.17860.21129.

[13] Mingzhe Zhou, "Research Advances in Deep Learning Object Detection," *Neurocomputing*, pp. 1–10, 2022, DOI: 10.1016/j.neucom.2022.03.018.

[14] P. Devaki, "Real-Time Object Detection using Deep Learning and OpenCV," *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, pp. 1–4, 2019, DOI: 10.35940/ijitee.K7846.0891220.

[15] Yung-Yao Chen, “Distributed Real-Time Object Detection Based on Edge-Cloud Collaboration for Smart Video Surveillance Applications,” *IEEE Access*, pp. 1–15, 2022, DOI: 10.1109/ACCESS.2022.3170678.

[16] Shrikant Jagannath Patro, “YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection,” *Ultralytics*, pp. 1–12, 2023, DOI: 10.1016/j.compind.2023.103411.

[17] Khoirun Nisa, “Implementation of Personal Protective Equipment Detection Using Django and Yolo Web at Paiton Steam Power Plant (PLTU),” *Journal of Information Technology Education: Innovations in Practice (JITEKI)*, pp. 1–15, 2023, DOI: 10.28945/4893.

[18] Sefik Ilkin Serengil, “Visualization of VGG Model Architecture,” *VGG Model Visualization*, Aug. 2018.

[19] Jiatu Wu, “Complexity and accuracy analysis of common artificial neural networks on pedestrian detection,” *MATEC Web of Conferences* , vol. 232, p. 01003, Jan. 2018, DOI: 10.1051/matecconf/201823201003.

[20] Kamil Figura, *Benefits of Microservices*, https://pretius.com/blog/benefits-of-microservices/, Accessed: December 25, 2023.

[21] Muhammad Raza et al., *Containers vs Virtual Machines*, https://www.bmc.com/blogs/containers-vs-virtual-machines/, Accessed: January 10, 2024.

[22] James Walker, *Kubernetes Architecture*, https://spacelift.io/blog/kubernetes-architecture/, Accessed: January 20, 2024.

[23] François Chollet, “Keras: The Python Deep Learning library,” *GitHub Repository*, pp. 1–10, 2015.