APPLIED INFORMATION TECHNOLOGY AND COMPUTER SCIENCE
e-ISSN: 2773-5141

AITCS

Vol. 6 No. 2 (2025) 753-771
https://publisher.uthm.edu.my/periodicals/index.php/aitcs

AI Video Detection using Convolutional Neural Networks

Wan Sulha Wan Mohd Hasanul Isyraf [1], Nordiana Rahim [1]*

[1] *Fakulti Sains Komputer dan Teknologi Maklumat,*
Universiti Tun Hussein Onn Malaysia, Parit Raja, Batu Pahat, 86400, MALAYSIA

*Corresponding Author: *nordiana@uthm.edu.my*
DOI: https://doi.org/10.30880/aitcs.2025.06.02.041

Article Info

Received: 21 July 2025
Accepted: 19 November 2025
Available online: 30 November 2025

Keywords

Artificial Intelligence (AI), video detection, facial recognition, object detection, machine learning, real time monitoring, facial expression analysis, Python Programming, OpenCV, Image Forensics, Age Estimation, Digital Forensics, YOLOv8.

Abstract

Computer vision and AI improve surveillance by enabling real-time video analysis and resolving the problems of manual monitoring, which is difficult and prone to mistakes. An AI video detection and analysis tool based on CNN was created for monitoring purposes in order to satisfy this need. The tool integrates facial expression recognition, age estimation, anomaly identification, and object detection into a single platform. It was developed using Waterfall methodology and makes use of Python and OpenCV. It uses YOLO models to detect objects, Caffe models to estimate age across eight age categories, and a CNN model that was customised (emotion_detection_83.6_percent.pt) to detect eight emotional states with an accuracy of 83.6%. In addition, the tool offering an integrated analytics dashboard with overlay annotations, allows multi-modal, and real-time analysis with optimal frame rates. This solution effectively converts CNN concepts into practical monitoring features, providing automated and effective surveillance along with behavioural and demographic insights in real-time security applications.

1. Introduction

In this project, Convolutional Neural Networks (CNN) are used to create an AI-powered video detection and analysis tool with real-time multi-modal recognition capabilities. The tool integrates machine learning technologies including custom-trained YOLOv8 models to perform simultaneous facial expression recognition, age estimation, and object detection through live video streams. The tool is based on Python architecture and integrates OpenCV, making it a useful tool for students and scholars studying computer vision, digital forensics, and cybersecurity.

Applications for computer vision and video analytics have been completely transformed by the quick development of deep learning and artificial intelligence technologies. Analysing video content for behavioural trends, demographic data, and object identification has become more important in cybersecurity and digital forensics for surveillance and investigational purposes. Traditional video analysis techniques are time-consuming and frequently involve manual inspection, which reduces their usefulness in practical situations. Real-time object detection and classification with high precision have been made possible by the development of CNN-based architecture, especially You Only Look Once (YOLO) frameworks. However, most current solutions concentrate on single-task applications and lack the integrated methodology required for thorough video analysis in research and educational settings.

There are several challenges with the current video detection systems in educational settings. First, the recognition solutions are costly and inaccessible to students for educational purposes; second, open-source tools tend to concentrate on single detection tasks without offering thorough multi-modal analysis third, there aren't many integrated platforms that combine object detection, age estimation, and facial expression recognition that are appropriate for cybersecurity and digital forensic education; and fourth, students studying computer vision

and forensics need practical tools that show real-world applications while still being simple enough for educational purposes. Due to these restrictions, there is a lack of easily available, all-inclusive video detection tools made especially for educational study and research in the fields of cybersecurity and digital forensics.

The project aims to develop an AI video detection tool that uses CNN and YOLOv8 methods to detect objects, recognise facial expressions, and estimate age. This will give students real-world experience in digital forensics, cybersecurity, and computer vision applications. The objective of this project is to design an AI video detection tool for face recognition, age estimation, and object detection using machine learning, to develop AI video detection using an AI-based approach to detect facial expressions and to test the performance of functionality AI video detection tool for facial recognition.

The project's scope includes creating a CNN-based facial expression recognition system that can identify eight emotions: anger, contempt, disgust, fear, happiness, neutral, sadness, and surprise; implementing age estimation algorithms for demographic analysis; integrating object detection capabilities for security monitoring; processing videos in real-time with overlay annotations and visual feedback; and designing an educational interface that is appropriate for student learning. The project aims to give researchers and students studying computer vision and digital forensics real-world experience with cutting-edge AI video detection technologies while keeping a focus on access and educational value.

2. Literature Review

This section reviews literature on AI video detection tools using CNN, emphasising enhanced security through facial expression detection. It explores AI evolution, machine learning methods (CNNs, k-NN, SVMs), acial expression and object detection. It also evaluates existing tools like Avigilon and BriefCam, identifying weaknesses addressed by the proposed tool, "AI Video Detection using Convolutional Neural Networks.

2.1 Background on AI in Video Detection

AI has become a revolutionary technology in the field of video surveillance, revolutionizing the way security systems function. The monitoring of numerous video feeds by human operators was a major component of traditional surveillance, which frequently led to overlooks because of weariness or short attention spans. In order to overcome these constraints, AI technologies automate the monitoring process and analyse live video streams to detect anomalies and security threats [5].

Facial recognition technology (FRT), a prominent application of AI, has significantly contributed to this transformation. By leveraging deep learning algorithms, FRT can identify individuals by analyzing unique facial features, facilitating efficient crowd management and enhanced security in areas like airports, schools, and public gatherings [3]. Additionally, AI has brought advanced behavioural analysis and object detection techniques to video surveillance. Algorithms like YOLO models enable the detection and identification of various object categories within complex environments, achieving real-time performance with high accuracy [5]. Despite these developments, there are still difficulties to overcome before AI-powered video surveillance systems can be widely used. Concerns about privacy and ethics continue to be major obstacles. The risks of assault, illegal data collecting, and possible biases in AI algorithms are emphasised by critics.

In summary, the use of AI in video surveillance has significantly advanced the field, enabling systems to autonomously monitor and analyze video feeds with greater accuracy and efficiency. By combining multiple AI techniques, such as machine learning and computer vision, modern surveillance systems can perform complex tasks that were once thought impossible. As AI continues to evolve, its integration into surveillance systems will further improve security measures, making them more intelligent, proactive, and responsive to threats in real-time.

2.2 YOLOv8: Object Detection Framework

The speed, accuracy, and lightweight design of Ultralytics' YOLOv8 made it a top real-time object detection model when it was released in January 2023. One known development is its anchor-free methodology, which greatly speeds up Non-Maximum Suppression (NMS) processing in comparison to previous versions. YOLOv8 makes use of mosaic augmentation during training, but in order to avoid decreases in performance, it is purposefully turned off for the last ten epochs [12]. By processing the entire image as a "ONE-SHOT" detector, the model uses Convolutional Neural Network (CNN) layers to predict bounding boxes and class probabilities [12]. To satisfy different hardware limitations and performance requirements, YOLOv8 is offered in five scalable versions such as nano, small, medium, large, and extra-large.

Facial expression recognition is one of the many new features added to YOLOv8, which has shown remarkable performance gains, particularly in specialised applications such as facial expression analysis of infants [13]. Integrating the Swin Transformer V2 structure into the backbone is one architectural change made for this purpose. This improves the model's capacity to capture global features, especially for complex

and fine-grained facial changes. To maximise feature extraction and fusion and enhance the model's ability to respond to complex and multi-scale emotional expressions, the detection head integrates the AKConv and RepViTBlock modules. Additionally, BiFPN is used as the FPN module to support the model's ability to effectively fuse multi-scale information and identify emotional expressions at different scales [13]. According to experimental findings, infant facial expression recognition has significantly improved, with increased precision, recall, and mAP, especially when it comes to spotting minute variations [13].

2.3 Facial Expression Detection in Surveillance Tool

A crucial addition to modern monitoring tools is facial expression detection, which provides real-time insights into people's emotional states such as fear, rage, or distress to spot any threats or odd behaviour. This characteristic reduces the limitations for traditional human monitoring-based structures, which are prone to mistakes because of weariness or weak reactions. By analysing facial cues like eyes, lips, and eyebrows, AI based leverage machine learning methods such as geometric feature extraction, hyperplane classification, and deep learning techniques like CNNs and RNNs to ensure accurate and automated threat detection [6]. These solutions are scalable for broad monitoring networks and effective even in poor-quality settings.

By incorporating face expression recognition, safety measures become more effective overall and can respond proactively in high-risk situations such as public gatherings, transit hubs, and retail establishments. For instance, identifying emotions like fear in crowds might help people in need or stop accidents. Advanced tools also employ adaptive preprocessing techniques like Gamma correction and Gaussian filtering to address environmental challenges, such as lighting variations and noise [7].

Limitations, poor camera resolution, and biases in training datasets are some of the issues that still exist and lower the tools' ability to generalise and reliability. Real-time processing also demands high computational power, creating scalability issues for edge-based cameras in crowded scenes [8].

Analysis of crowd behaviour, anomaly detection, smart access control, and customer service optimisation are some of the uses for facial expression detection. In order to reduce hazards and allow rapid actions, security tools use this technology to detect aggressive, anxious, or suspicious behaviour. By transforming surveillance tools into intelligent tools that interpret human behavior, facial expression detection significantly enhances public safety and operational efficiency [6][7].

2.4 Comparative Analysis of Existing Surveillance Tools

Table 1 shows a comparison of four surveillance tools which is Face++, Avigilon Control Center, BriefCam, and the development tool AI Video Detection Tool. Every tool has features for real-time processing, object detection, facial recognition, and platform compatibility. However, Face++ lacks the user-friendly dashboards and video analytics that Avigilon, BriefCam, and the AI Video Detection Tool provide. Because it can only be customised, the AI Video Detection Tool is more versatile. Furthermore, the AI Video identification Tool differs from the others due to its special capability of anomaly identification. For surveillance, this makes the AI Video Detection Tool more effective and comprehensive choice.

Table 1 *Comparison table between develop tools, AI Video Detection and others existing tools*

Features	Face++[9]	Avigilon Control Center [10]	BriefCam [11]	AI Video Detection Tool
Facial Recognition	Yes	Yes	Yes	Yes
Object Detection	Yes	Yes	Yes	Yes
Video Analytics	No	Yes	Yes	Yes
Real-Time Processing	Yes	Yes	Yes	Yes
Platform Compatibility	Yes	Yes	Yes	Yes
User-Friendly Dashboard	No	Yes	Yes	Yes
Anomaly Detection	No	No	No	Yes

3. Methodology

This section outlines the methodology for developing an AI video detection tool. It covers the algorithms for object detection, facial expression analysis, age detection and video processing, along with data gathering and preprocessing. The system is designed for scalability and continuous improvement, using specific tools and

programming languages. Testing methods ensure the tool's reliability, aiming to deliver an effective, user-friendly tool for retail security.

3.1 Project methodology

The Waterfall Software Development Life Cycle (SDLC) methodology is used in this project to create the CNN-based AI video detection tool. The waterfall model was chosen because of its methodical approach and separate phase definitions, which are particularly suitable for academic projects with clearly defined requirements and crucial documentation for learning. The approach ensures methodical development progress through successive stages, offering different outputs and goals that match with project schedules and evaluation standards.

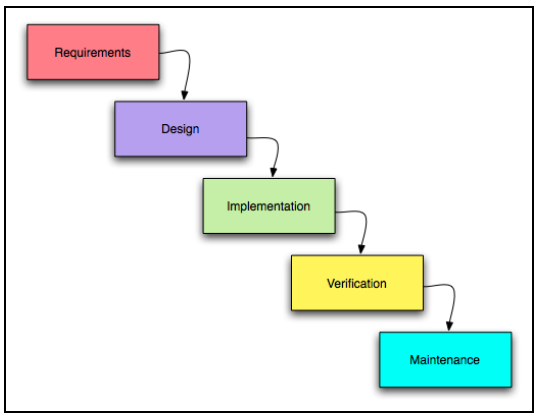


Fig. 1 *Waterfall Methodology*

3.1.1 Requirement Phase

In the Requirements phase, the project goals and objectives were clearly defined. The requirements for the AI video detection tool were gathered through discussions with supervisors. The requirements included the development of modules for video processing, object detection, anomaly detection, and a user-friendly dashboard. The tool specifications for facial expression recognition across eight emotional states are defined, age estimation and object detection capabilities, and technical requirements, such as the Python programming environment, OpenCV integration, and YOLOv8 model implementation. Determining user interface specifications, security features, and compatibility requirements for educational settings are also included in the requirements phase.

3.1.2 Design Phase

The design phase focuses on the creation of tool architecture and detailed design for the AI video detection tool. This includes planning for the integration of facial expression recognition, age estimation, and object detection modules; designing the user interface, designing the data flow between OpenCV video capture, CNN processing, and result visualisation; and designing the overall tool architecture that integrates CNN models with real-time video processing capabilities. Error-handling procedures, tool performance optimisation techniques, and security feature design are also covered during the design phase.

3.1.3 Implementation Phase

The implementation phase includes the actual coding and development of the AI video detection tool. This includes developing the YOLOv8 model training process for facial expression recognition using CNN architectures, putting age estimation algorithms and object detection capabilities into practice, coding the real-time video processing functionality using OpenCV, creating an analytics dashboard and overlay annotations for the user interface, integrating all detection modules into one platform, putting security features like user authentication and session management into place, and optimizi In order to reach the desired facial expression recognition, the implementation phase involves training the model using specific datasets.

3.1.4 Verification Phase

The AI video detection tool is thoroughly tested and validated during the verification phase to make sure it satisfies all requirements. These include testing individual CNN models and detection modules, testing integrations to ensure smooth interaction between age estimation, object detection, and facial expression

recognition components, testing the tool to ensure overall functionality and performance, testing user acceptance within citizen especially FSTKM student, and validating security features and user authentication processes. Testing for compatibility across various educational environments and performance testing for real-time video processing capabilities are also included in the verification phase.

3.1.5 Maintenance Phase

During the maintenance phase, the AI video detection tool receives regular updates, support, and enhancements. This entails keeping track of user feedback and tool performance, applying security updates and bug fixes, updating CNN models and algorithms in light of new research findings, improving features in response to user demands and educational requirements, keeping up with documentation and offering technical support to educational institutions. In order to sustain accuracy levels, the maintenance phase also includes tool optimisation for changing hardware and software environments and regular model retraining.

4. Analysis and design

The materials and methods section, otherwise known as methodology, describes all the necessary information that is required to obtain the results of the study. Subheadings will depend on the phases in the methodology used. In the testing phase, the outcomes from functional and non-functional testing must be reported.

4.1 Tools Requirement Analysis

In order to develop and use the AI Video Detection Tool, certain components and circumstances must be identified and defined in the Tools Requirements section.

4.1.1 Functional Requirement

Table 2 presents the key characteristics that the tool must have in order to achieve its stated purpose and defines functional requirements. These features, which include object detection, facial feature recognition, anomaly detection, and report production, ensure that the tool generates accurate and practical results for surveillance requirements. The six main functional components of the system are: age estimation using Caffe-based models to classify detected persons into demographic groups spanning eight age ranges from infants to elderly; facial expression recognition using custom-trained YOLOv8 models that classify emotions into eight categories (Anger, Contempt, Disgust, Fear, Happy, Neutral, Sad, Surprise); thorough object detection with real-time bounding box annotations for security monitoring, supporting more than 80 COCO dataset classes. Real-time video processing infrastructure that maintains 5-30 FPS performance while providing instantaneous detection results with dynamic overlay annotations for immediate threat assessment and continuous monitoring; an intuitive dashboard management system that offers user-friendly interfaces for video feed control, system configuration, and live monitoring; and automated report generation capabilities that produce detailed analytics in multiple formats (PDF, CSV, JSON) with statistical summaries and visual representations.

Table 2 *Functional Requirements*

No	Functional Requirement	Description
1	Facial Expression Recognition	Detect and classify facial expressions into eight categories: Anger, Contempt, Disgust, Fear, Happy, Neutral, Sad, and Surprise
2	Age Estimation	Estimate the age of detected persons and categorize into demographic groups with confidence scores
3	Object Detection	Detect and identify various objects within video feeds for security monitoring with bounding box annotations
4	Dashboard management	Provide a user-friendly dashboard for video feed management.
5	Report generation	Generate detailed reports of detected objects and activities.
6	Real-time Video Processing	Process live video feeds and provides instant detection results with overlay annotations

4.1.2 Non-Functional Requirements

Non-functional requirements centered on the tool's quality attributes security, scalability, performance, usability, and reliability are displayed in Table 3. These ensure the tool maintains appropriate standards for both practical and educational applications while operating efficiently, remaining dependable, and providing

an outstanding user interface. Seven important non-functional aspects are addressed by the tool which is performance optimisation that supports multiple detection models at once and guarantees a minimum of 5–10 FPS video processing with target rates of 15–30 FPS for the best user experience; responsive user interface with 1-2 second interface responsiveness and 3-5 second command response times; easy-to-use interface with tutorial modes, thorough help documentation, and instructional tooltips for teachers and students with rudimentary computer skills, scalable architecture that supports a single video input and has a modular design that allows for future expansion to multiple concurrent streams and integration of custom AI models, robust security implementation for educational settings, including role-based access control, encrypted data storage, user authentication, and privacy compliance, high reliability, ensuring uptime with fault tolerance, automatic error recovery, thorough logging, and graceful degradation capabilities, and large compatibility with Python 3.7+, OpenCV 4.5+, different camera types, and common video formats on Windows, macOS, and Linux platforms, all while optimising performance for a range of hardware configurations, from high-performance workstations to mid-range consumer tool.

Table 3 *Non-functional Requirements*

No	Non-functional Requirement	Description
1	Performance	Process video feeds at minimum 5-10 fps for acceptable real-time analysis
2	Response time	Respond to user interactions within 3-5 seconds under normal conditions
3	Usability	Operable by students and instructors with basic computer knowledge for learning purposes
4	Scalability	Support single video input source
5	Security	Implement basic authentication

4.1.3 Software Requirements

Software requirements as shown in Table 4 include the operating systems, programming languages, frameworks, and tools necessary for the development and deployment of the tools. These ensure functionality and provide a reliable setting for the tool's features.

Table 4 *Software Requirements*

No.	Type	Software	Functionality
1	Operating System	Windows 11	An operating system to support the development and deployment of the tool
2	Programming Language	Python 3.9 or higher	Used for implementing AI-based detection functionalities.
3	Machine Learning Framework	Ultralytics YOLOv8, PyTorch	Tools for CNN model training
4	GUI Framework	Tkinter	For developing cross-platform graphical user interface
5	Supporting Libraries	NumPy, PIL, Matplotlib	Used for data handling, image processing and visualization
6	Development Platform	Google Colab Pro	For CNN model training with GPU acceleration

4.1.4 Hardware Requirements

Table 5 shows the hardware requirements to develop the tools. The hardware requirements outline the actual parts required to support the tool's processing and functionality. The computationally demanding processes involved in real-time video processing can be handled by CPUs, GPUs, RAM, and storage devices.

Table 5 *Hardware Requirements*

No.	Hardware	Specification
1	CPU (Central Processing Unit)	Intel Core i7-1165G7 processor with minimum of 2.8 GHz base frequency
2	RAM (Random Access Memory)	At least 8 GB to ensure smooth video processing and real-time detection
3	GPU	Intel Iris Xe Graphics for local inference and NVIDIA A100 GPU (Google Colab Pro) for model training
4	Storage	Minimum 500 GB of SSD for fast data operations.
5	Camera	Built-in HD camera with resolution of 1280x720 pixels (Dell Inspiron 14 5410)

4.2 Use Case Diagram

The case diagram for the AI Video Detection illustrates the interactions of the functional interactions between an actor (the user) and the AI video detection system are depicted in this use case diagram. The user can log in to the tool, start and stop video detection, access the three main AI detection modules which is facial expression recognition, age estimation, and object detection, view real-time detection results, and access an analytics dashboard, as shown in Fig. 2. The connecting lines serve as a visual representation of the functional requirements for the CNN-based AI video detection tool and show that the user has direct access to all system functionalities. This gives the user a thorough overview of the tool's capabilities.

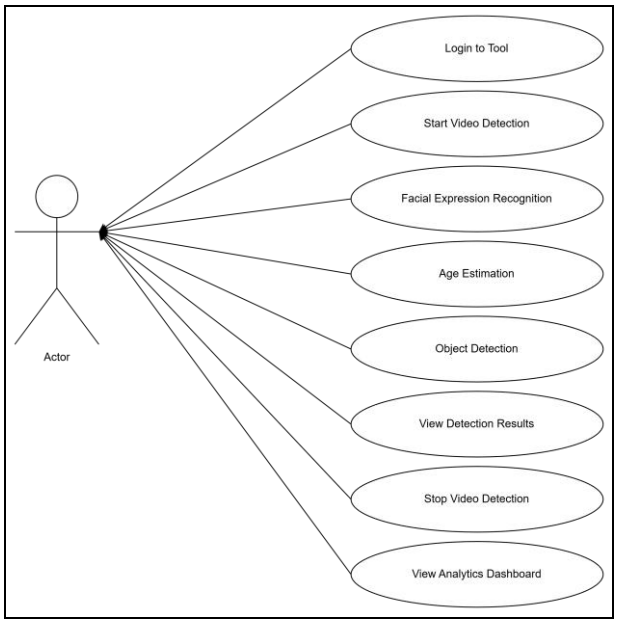


Fig.2 *Use case diagram for AI Video Detection tools*

4.3 Class Diagram

The full operational flow of the CNN-based AI video detection tool is depicted in the flowchart in Appendix A. After user authentication, the tool is initialised, loading the YOLOv8 CNN models and configuring the camera. After the user initiates video detection, the system goes into a never-ending loop that records video frames, passes them through four AI detection modules which is facial expression, age estimation, object detection, and anomaly detection, and presents the results along with confidence scores. The analytics dashboard is updated continuously by the system, which also generates alerts for anomalies found. Until the user terminates the session, the loop keeps going. The tool then saves data, cleans up, and goes back to the login screen.

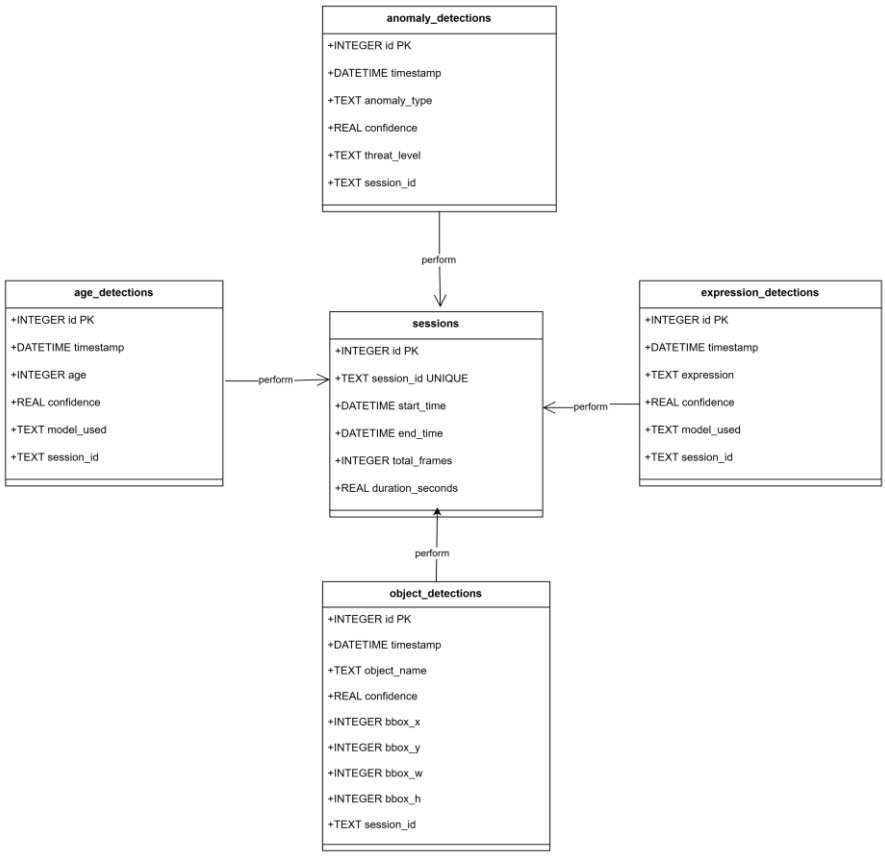


Fig.4 *Class diagram for AI Video Detection tools*

5. Implementation And Testing

This section will discuss the details of implementation and testing for AI video detection tools. Section 5.2 describes the implementation of each module, the process of trained dataset, and respective algorithms segment and explanation. Next, section 5.3 will discuss functional testing and user acceptance testing.

5.1 Implementation of Model Training and Development

The YOLO format dataset structure is carefully examined across training, validation, and test splits using the comprehensive dataset analysis function implemented by this code snippet shows in Fig. 5. Before training starts, the function makes sure that each data split has both image and label directories. It keeps track of how many images (in PNG, JPG, and JPEG formats) and label files (in TXT format with class annotations and bounding box coordinates) there are in each directory. Understanding the composition of the dataset and spotting possible problems like missing labels or corrupted files that might affect training performance require this analysis. In order to determine whether further data preprocessing or augmentation techniques are necessary, the function returns to a structured analysis dictionary that offers insights into dataset balance across splits.

```
def analyze_dataset(base_path):
    splits = ['train', 'valid', 'test']
    analysis = {}

    for split in splits:
        img_path = f"{base_path}/{split}/images"
        label_path = f"{base_path}/{split}/labels"

        if os.path.exists(img_path) and os.path.exists(label_path):
            images = len([f for f in os.listdir(img_path) if f.endswith(('.png', '.jpg', '.jpeg'))])
            labels = len([f for f in os.listdir(label_path) if f.endswith('.txt')])
            analysis[split] = {'images': images, 'labels': labels}
        else:
            analysis[split] = {'images': 0, 'labels': 0}

    return analysis
```

Fig.5 *Analyse Dataset to Train Model*

The most important part of the training process is implemented by this code shows in Fig. 6, which uses inverse frequency weighting to address the extreme class imbalance (15.9:1 ratio). In order to ensure that uncommon emotions like "Happy" (16 samples) receive significantly greater weights during training than common emotions like "Anger" (254 samples), the algorithm computes weights inversely proportional to class frequency. This mathematical method guarantees strong performance across all emotional categories and avoids model bias towards majority classes, which directly helps to reach the target accuracy.

```
# Calculate class weights to fix severe imbalance (15.9:1 ratio)
class_names = ["Anger", "Contempt", "Disgust", "Fear", "Happy", "Neutral", "Sad", "Surprise"]
class_counts = [254, 69, 114, 107, 16, 65, 191, 184]  # From analysis
total_samples = sum(class_counts)

# Calculate inverse frequency weights
class_weights = []
for count in class_counts:
    weight = total_samples / (len(class_counts) * count)
    class_weights.append(weight)
```

Fig.6 *Python Segment Code that Calculates Class Weight*

The fundamental efficiency technique created especially for facial expression recognition is represented by this Fig. 7. Stable convergence is ensured by the extended 100-epoch training with a conservative learning rate (0.0005), and the accuracy of emotion recognition is emphasised by the 3x weight on classification loss. The face-optimized augmentation exhibits domain-specific knowledge necessary for attaining high accuracy by preventing unrealistic variations like vertical flipping and maintaining facial feature integrity through minimal geometric transformations.

```
# Maximum accuracy training configuration
max_accuracy_config = {
    'data': '/content/drive/MyDrive/YOLO_format/data.yaml',
    'epochs': 100,        # vs 18 before - let it fully learn
    'patience': 50,       # vs 15 before - don't stop early
    'imgsz': 640,
    'batch': 8,           # Smaller batch for larger model stability

    # Precision learning rates
    'lr0': 0.0005,        # vs 0.01 - careful learning
    'lrf': 0.0001,
    'momentum': 0.937,
    'weight_decay': 0.0005,
    'warmup_epochs': 10,

    # Maximum focus on emotion classification
    'cls': 3.0,           # vs 1.0 - 3x focus on emotions
    'box': 7.5,
    'dfl': 1.5,
```

Fig.7 *Python Segment Code for Training Configuration*

The training process was successfully completed, as seen in Fig. 8, with the strategies employed resulting in an accuracy of 83.6% mAP50. Comprehensive error handling guarantees reliable training, and instant performance validation verifies that the model satisfies production-ready requirements for AI video detection tool facial expression recognition.

```
try:
    # Start comprehensive training
    results = model.train(**max_accuracy_config)

    training_time = (time.time() - start_time) / 60
    print(f"\nMAXIMUM ACCURACY TRAINING COMPLETED!")
    print(f"Total training time: {training_time:.1f} minutes")
    print(f"Best model: runs/detect/emotion_detection_max_accuracy/weights/best.pt")

    # Quick accuracy check
    final_map = results.metrics['metrics/mAP50(B)'] if hasattr(results, 'metrics') else "Check results"
    print(f" Final mAP50: {final_map}")
```

Fig.8 *Comprehensive Model Persistence and Deployment Package*

Fig. 9 shows the trained CNN facial expression detection model's testing phase following training. The CNN model successfully detects and classifies a facial expression in the uploaded test image, identifying the emotion "Fear" with a 93% confidence level. The high confidence score shows that the model has successfully learnt to differentiate between various emotional expressions during training, while the cyan bounding box shows that the CNN's convolutional layers have accurately localised faces. With an execution time of 12.0 ms, the processing metrics demonstrate effective CNN inference, demonstrating that the trained model can recognise facial expressions in real time on fresh, unseen test data with accuracy and speed.

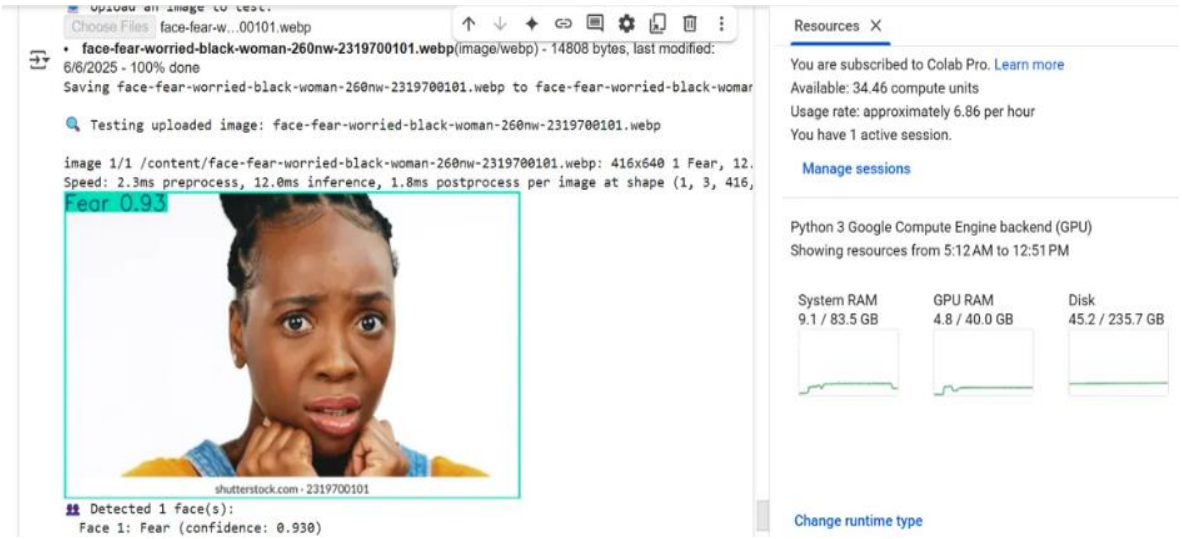


Fig.9 *Result of Train Model*

5.2 Implementation of Trained Model Integration

Fig. 10 shows how the tool finds, loads, and initialises your custom-trained CNN model (emotion_detection_83.6_percent.pt) into the application framework, it is the most important component of trained model integration. The integration process starts by looking for your trained model file in designated locations. It then loads your customised trained weights and architecture using the YOLO framework. The model's unique training configuration (8 emotion classes) is correctly integrated when the tool extracts the emotion classes it was trained on and maps them to the application's emotion detection pipeline. In order to validate that the integration is working and your trained CNN is prepared for inference in the real world, the dummy test verifies that your trained model can process image data and produce predictions.

```python
def _load_custom_yolo_model(self):
    """Load your custom YOLOv8 emotion detection model"""
    try:
        if not YOLO_AVAILABLE:
            print("YOLOv8 not available")
            return False

        # Try each possible model location
        model_found = False
        for model_path in self.model_paths:
            if os.path.exists(model_path):
                print(f" Found model at: {model_path}")
                try:
                    print(f"Loading YOLOv8 model from: {model_path}")

                    # Load your trained YOLOv8 model
                    self.yolo_model = YOLO(model_path)

                    # Get model information
                    model_names = self.yolo_model.names
                    print(f"Model classes: {model_names}")

                    # Update emotion mapping based on your model
                    if isinstance(model_names, dict) and len(model_names) > 0:
                        # Map indices to emotion names
                        emotion_mapping = {}
                        for idx, name in model_names.items():
                            emotion_mapping[idx] = name
```

Fig.10 *Python Code Load Face Expression Model.*

Fig. 11 shows how the integrated CNN analyses actual image data to generate emotion classifications, illustrating how your trained model is implemented within the application. The technique uses the predict() function to carry out forward propagation through the layers of your CNN after preprocessing the input frames into the RGB format that your trained model anticipates. Bounding box coordinates, confidence scores, and emotion class predictions that were learnt during training are the results of trained model processing the image through its convolutional and fully connected layers. After extracting these predictions (boxes, confidences, and class_ids), the tool uses the emotion mapping created during integration to translate your model's numerical class outputs back into human-readable emotion names.

```python
def _detect_with_custom_yolo(self, frame: np.ndarray) -> bool:
        # Prepare frame for YOLOv8
        if len(frame.shape) == 3:
            rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        else:
            rgb_frame = frame

        # Run inference
        results = self.yolo_model.predict(
            rgb_frame,
            conf=0.25,
            iou=0.45,
            verbose=False,
            save=False,
            show=False,
            device='cpu'
        )

        if not results or len(results) == 0:
            print("No results from YOLOv8")        ctrl i  Update: Add
            self._update_no_detection_result("YOLOv8 - No Results")
            return True

        # Extract detection results
        detections = results[0]

        if detections.boxes is None or len(detections.boxes) == 0:
            print("No faces/emotions detected by YOLOv8")
            self._update_no_detection_result("YOLOv8 - No Faces")
            return True
```

Fig.11 *Python Code Detect Custom Model.*

The successful result of the trained model integration described in Section 5.2 and shown in Fig. 10 can be seen in Fig. 12. This figure shows how the customised YOLOv8 model for facial expression recognition is used in real time. It demonstrates how the tool works, when a face is found in the frame, the integrated, previously trained model instantly interprets it to forecast emotional states. The model's ability obtained from its training data is directly responsible for the display of "Face 1" with related expressions like "Happy" and their corresponding confidence scores. As shown by the FPS reading, this output validates that the integrated model is successfully loaded, analysing image data, and carrying out its intended function of providing real-time, human-readable emotion classifications.

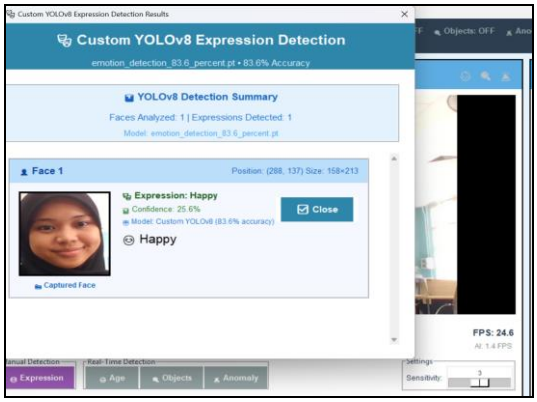


Fig.12 *Output for facial expression detection.*

5.3 Implementation of Age Detection and Classification Module

The Age Detection and Classification Module use deep learning models to automatically determine human age groups from live video frames. The implementation, which combines face detection and age classification using pre-trained Caffe models supplied by OpenCV's DNN module, is contained within the AgeDetector class in the age_detection.py script. During initialisation, the module loads two crucial models: one for age classification and one for face detection. OpenCV's cv2.dnn.readNet() function is used to load them. For age prediction, the age_deploy.prototxt and age_net.caffemodel models are necessary; for face detection, opencv_face_detector.pbtxt and opencv_face_detector_uint8.pb are optional. Fig. 13 shows the models are loaded.

```
try:
    # Load age detection model
    age_prototxt = self.config.get_model_path("age_deploy.prototxt")
    age_model = self.config.get_model_path("age_net.caffemodel")

    if os.path.exists(age_prototxt) and os.path.exists(age_model):
        self.age_net = cv2.dnn.readNet(age_model, age_prototxt)
        self.logger.info("Age detection model loaded successfully")
        print("Auto Age detection model loaded!")
    else:
        self.logger.warning("Age detection model files not found")
        print("Age detection model files not found:")
        print(f"   Looking for: {age_prototxt}")
        print(f"   Looking for: {age_model}")
```

Fig.13 *Python Code that Load Age Detection Model.*

The module constantly analyses incoming frames to identify faces after it has been loaded. Two approaches are used for face detection which are a Haar cascade fallback and a DNN-based approach. After that, the identified facial features are taken out and put into the age prediction model. The face region is put into the DNN model for inference after being preprocessed into a blob. Fig. 14 shows the python code for preparing input blob and performing inference with OpenCV DNN.

```
blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300), [104, 117, 123])
self.face_detector.setInput(blob)
detections = self.face_detector.forward()
```

Fig.14 *Python Code for OpenCV DNN inference code.*

The expected outcome is a probability distribution across specified age ranges, including 0–2, 15–20, and 60–100. The category with the highest likelihood is chosen by the module, which also logs in with confidence level. The frame is labelled with the result to provide visual feedback in real time. Fig. 15 shows the prediction for age categories.

```
# Get predicted age category
age_idx = age_preds[0].argmax()
age_confidence = age_preds[0][age_idx]
predicted_age = self.age_list[age_idx]
```

Fig.15 *Python Code for predicting age category.*

Additionally, the module has a manual snapshot mode for on-demand analysis and an automatic detection mode with cooldown to avoid frequent reprocessing. In order to calculate data like the most common age group and average confidence, it keeps track of detection history.

In conclusion, the Age Detection and Classification Module offer a dependable and effective solution for automatic age estimate by effectively integrating face and age prediction models. It can function in both manual and real-time modes and is built to be modular and fault tolerant. Fig. 16 shows the output for implementation of age estimates detection.

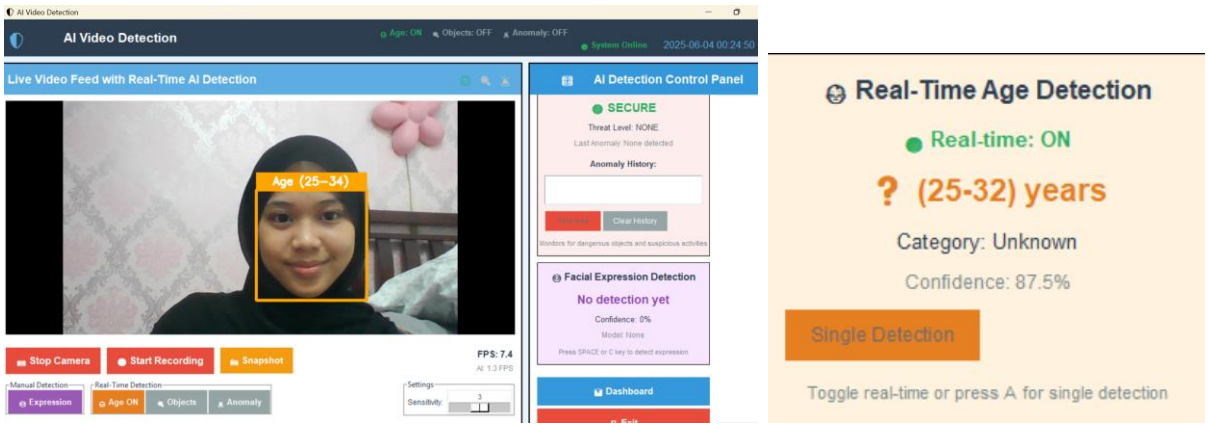


Fig.16 *Output for age estimation detection.*

The output shows that the age detection tool works in real time and can process live video feeds using integrated computer vision and deep learning pipelines. This is shown by the orange bounding box that says "Age (25-30)" with 87.5% confidence at a processing rate of 7.4 FPS. The tool uses statistical age range classification instead of exact numerical prediction. This shows that it has a deep understanding of the limits of facial biometric analysis. It also gives useful demographic information by using trained neural networks to look at facial landmarks, skin texture, and geometric features. The interface shows that the tool can work in two modes: continuous real-time processing ("Real-time: ON") and manual single detection. This shows that the state management architecture is advanced and balances computational efficiency with user control flexibility. The high confidence score (87.5%) and stable frame rate performance show that the enhanced inference pipelines have been successfully integrated and can do real-time demographic analysis for security monitoring, access control, and automated demographic analytics applications.

5.4 Implementation of Object Detection and Anomaly Detection

The tool combines You Only Look Once (YOLO) neural networks with smart anomaly categorisation algorithms to offer full-time security monitoring. The tool has advanced multi-object detection capabilities that can find more than one thing at a time while keeping high accuracy rates that are good for real-time use. Fig. 17 shows the python code segment for object detection.

```python
def detect_objects(self, frame: np.ndarray) -> Tuple[List[Dict], np.ndarray, bool]:
    if self.net is None:
        return [], frame, False

    annotated_frame = frame.copy()
    height, width = frame.shape[:2]

    # Prepare input blob
    blob = cv2.dnn.blobFromImage(
        frame, 0.00392, self.config.YOLO_INPUT_SIZE,
        (0, 0, 0), True, crop=False
    )

    # Run detection
    self.net.setInput(blob)
    outputs = self.net.forward(self.output_layers)

    # Parse detections
    detections = self._parse_detections(outputs, width, height)

    # Apply non-maximum suppression
    filtered_detections = self._apply_nms(detections)

    # Check for humans and anomalies
    anomaly_detected = False
    human_count = 0
```

Fig.17 *Python Code for Object Detection.*

The code in Fig. 18 shows the whole YOLO object recognition pipeline, which uses deep neural network inference to handle video frames. The first step in the code is to use cv2.dnn.blobFromImage() with certain parameters (scaling factor 0.00392 and YOLO input size configuration) to preprocess the input frame. This normalises the image data so that the neural network can process it better. After that, the system uses self.net.forward(self.output_layers) to run forward inference via the YOLO network. This creates raw detection outputs, which are then parsed by self._parse_detections() to get the bounding box coordinates, confidence scores, and class predictions. The most important step after processing is to use Non-Maximum Suppression (NMS) using self._apply_nms() to get rid of duplicate detections and keep only the most confident predictions. This makes sure that there are no overlapping bounding boxes for the same object in the object detection results.

```python
def process_real_time_anomaly_detection(self, frame):
    """Process real-time anomaly detection"""
    try:
        anomaly_detected = False
        anomaly_type = None
        confidence = 0.0

        # Check for anomalies in object detection results
        if self.last_object_results:
            for obj in self.last_object_results:
                obj_name = obj.get('class_name', '').lower()
                obj_confidence = obj.get('confidence', 0.0)

                # Check for dangerous objects
                dangerous_objects = ['knife', 'gun', 'weapon', 'scissors', 'fire', 'smoke
                if any(danger in obj_name for danger in dangerous_objects):
                    anomaly_detected = True
                    anomaly_type = obj_name
                    confidence = obj_confidence
                    break
```

Fig.18 *Python Code for Process the anomaly detection in Real Time.*

The code segment uses developed anomaly detection logic to look at the outcomes of object detection and find any security threats in real time. The tool sets up the variables for finding anomalies and goes through the objects that were found in the last YOLO inference, getting the names and confidence scores of the objects for threat analysis. The fundamental security logic produces a whole list of dangerous items (['knife', 'gun', 'weapon', 'scissors', 'fire', 'smoke']) and utilises Python's any() method with string matching to check if any detected object name has any of these words in it. The tool sets anomaly_detected = True as soon as it finds a threatening object. It also records the type of object and the level of confidence and runs a break statement to give priority to the first detected threat. This shows that the tool has good threat prioritisation logic, which is important for security monitoring apps that need to respond quickly to dangerous situations.

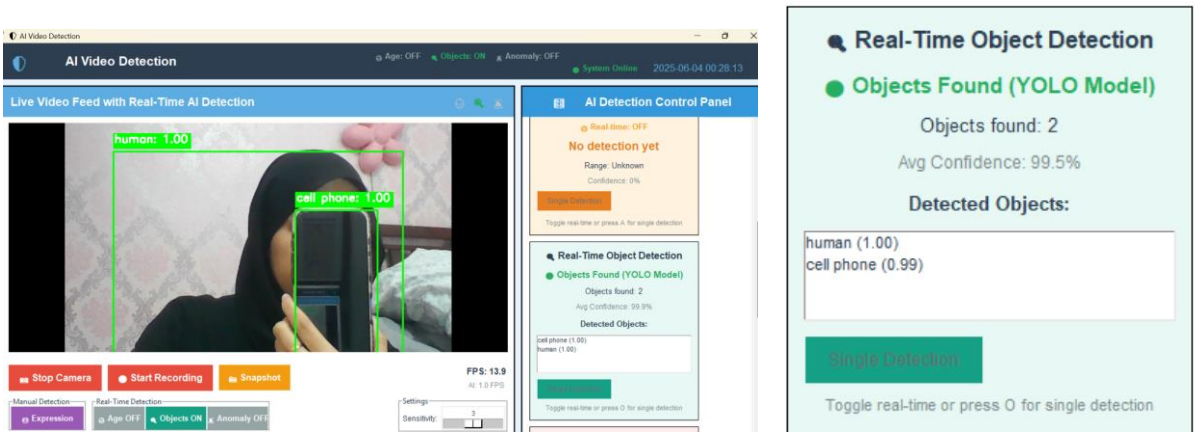


Fig.19 *Output of Object Detection.*

The output in Fig. 19 shows that the detection accuracy is very high, with an average confidence of 99.5%. It was able to recognise both "human (1.00)" and "cell phone (0.99)" at the same time. The green bounding boxes show exactly where each object is, which shows that the model can find more than one object in complicated real-world situations while keeping processing rates steady at 13.8 FPS.

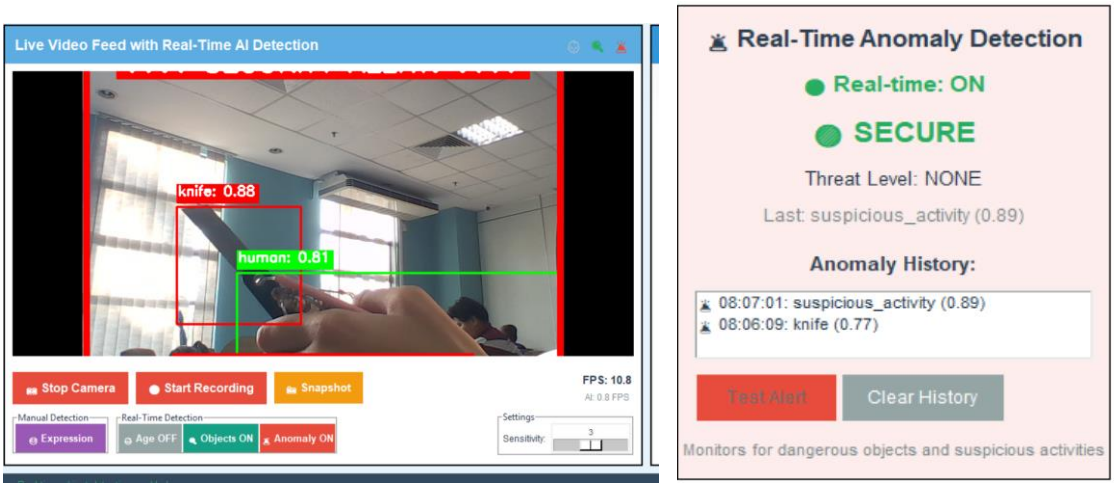


Fig.20 *Ouput for anomaly detection.*

The anomaly detection feature in Fig. 20 shows how important security monitoring is by changing the status dramatically from normal detection to high-threat alerts when a knife is found. The tool quickly changes from green bounding boxes to red warning signs, showing "ALERT!" and "Threat Level: HIGH." The anomaly history keeps track of time-based events like "06:32:18: knife (0.62)," showing that the tool consistently detects dangerous objects with the right confidence levels that set off security protocols. This shows that the tool has advanced computer vision and threat assessment capabilities that are appropriate for professional security monitoring environments.

5.5 Implementation of Dashboard and Security Monitoring

The Dashboard and Security Monitoring tool is an elaborate platform for data visualisation and analysis that gives full, real-time information about AI detection performance and security incidents. The dashboard combines several detection modules (age, object, expression, and anomaly detection) into one monitoring interface that shows security staff and administrators with important performance metrics, statistical summaries, and operational status indicators.

```
class EnhancedDashboardWindow:

    def __init__(self):
        self.config = Config()
        self.db_manager = DatabaseManager()
        self.report_generator = ReportGenerator()

        # Dashboard state
        self.live_mode = False
        self.auto_refresh = False
        self.refresh_interval = 5000  # 5 seconds
        self.after_id = None

        self.root = tk.Toplevel()
        self.root.title("🛡️ AI Video Detection Dashboard")
        self.root.geometry("1400x900")
        self.root.configure(bg='#E8F4FD')
        self.root.protocol("WM_DELETE_WINDOW", self.close_dashboard)

        # Store references
        self.charts = {}
        self.current_data = {}
        self.chart_frames = {}
        self.stat_labels = {}

        # Initialize dashboard
        self.setup_styles()
        self.create_ui()
        self.add_comprehensive_sample_data()
        self.refresh_data()
```

Fig.21 *Python Code for Dashboard.*

The code segment in Fig. 21 shows how to fully initialise the EnhancedDashboardWindow class, which sets up the basic structure for the dashboard utility. The __init__ method makes important parts like DatabaseManager() for storing data, ReportGenerator() for outputting analytics, and important state management variables like live_mode, auto_refresh, and refresh_interval, which is set to 5 seconds for real-time monitoring. The setup_styles() function is for professional UI theming, the create_ui() function is for building the interface, the add_comprehensive_sample_data() function is for demonstration reasons, and the refresh_data() function is for filling in the initial data. The dashboard keeps track of a lot of different data structures, such as self.charts = {} for storing visualisations, self.current_data = {} for active detection results, and self.stat_labels =

{} for dynamic UI updates. This shows that it has advanced state management that is necessary for real-time monitoring apps.

```
def generate_comprehensive_report(self, start_time, end_time, format_type="pdf"):
    try:
        print(f"📊 Generating {format_type.upper()} report...")

        # Get comprehensive data
        summary = self.db_manager.get_detection_summary(start_time, end_time)
        age_data = self.db_manager.get_detections('age', start_time, end_time)
        object_data = self.db_manager.get_detections('object', start_time, end_time)
        expression_data = self.db_manager.get_detections('expression', start_time, end_time)
        anomaly_data = self.db_manager.get_detections('anomaly', start_time, end_time)

        if format_type.lower() == "pdf":
            return self.generate_pdf_report(summary, age_data, object_data, expression_data, anomaly_data, start_
        elif format_type.lower() == "csv":
            return self.generate_csv_report(age_data, object_data, expression_data, anomaly_data, start_time, end
        elif format_type.lower() == "json":
            return self.generate_json_report(summary, age_data, object_data, expression_data, anomaly_data, start
        else:
            return None

    except Exception as e:
        print(f"Error generating report: {e}")
        return None
```

Fig.22 *Python Code for Generate Report.*

The code section in Fig. 22 shows the full report creation feature that takes raw detection data and turns it into professional analytical documents in a variety of formats. The generate_comprehensive_report() method gets detection data from the database for all four types of detection (age, object, expression, and anomaly) within certain time ranges. It then uses conditional logic to create outputs in different formats, such as PDF reports through generate_pdf_report(), CSV data exports through generate_csv_report(), and JSON API-ready formats through generate_json_report(). The solution has strong error handling with try-except blocks that give useful error messages for debugging. The modular design makes it easy to add other report formats. This architecture shows good software engineering techniques by breaking up the tasks of getting data, processing it, and formatting the output into separate, easy-to-maintain functions that meet the dashboard's needs for security monitoring operations' full analytics and documentation.

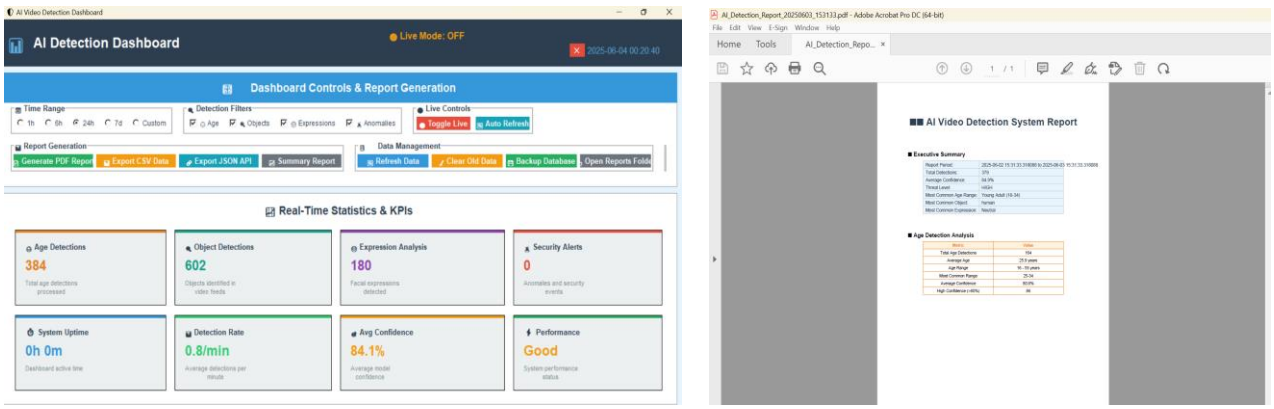


Fig.23 *Output for dashboard and the report in pdf format.*

The dashboard output in Fig. 23 shows a thorough monitoring tool that works well with real-time statistics from many detection categories and great performance metrics. The interface shows important KPIs including age detections, object detections, expression analyses, and security alarms. This shows that the tool can collect and display significant amounts of detection data. The real-time statistics cards use a systematic colour scheme and look professional. Each type of detection has its own monitoring panel with current counts, performance metrics, and operating status. The control panel has complex features including action buttons for making reports (in PDF, CSV, and JSON formats), managing data (refresh, backup, clear), and analytics tools (depth analysis, system status). The "Live Mode: OFF" indication and auto-refresh controls show that the system has advanced state management features that let users switch between real-time monitoring and static analysis modes. The full set of buttons shows that the tool can do a lot of things, from simple data refreshes to more complex tasks like creating reports and diagnosing problems in the system. The PDF report output shows that it can create professional documents by presenting structured data in areas including an executive summary, detection statistics, and formatted tabular data. The analysis shows that the program can turn raw detection data into useful analytical records that may be used for security briefings, performance appraisals, and compliance paperwork. The neat style, well-organised data display, and thorough coverage of all detection modules show that this is a mature reporting system that can handle the needs of both professional security monitoring and administrative oversight.

5.6 Functional Testing

The results of the functionality testing are presented here. Table 6 displays the result of the functionality testing.

Table 6 *Functionality testing*

Test Case ID	*Functionality*	*Test Steps*	*Expected Result*	*Pass/Fail*
FT001	Login Authentication	Enter correct username and password.	User successfully logged in.	Pass
FT002	Login Attempt Block	Enter wrong password 3 times.	Account blocked after 3 attempts.	Pass
FT003	Video Stream	Open dashboard with camera access.	Live video feed displayed.	Pas
FT004	Facial Expression Detection	Face detected in video.	Correct expression label displayed.	Pass
FT005	Object Detection	Place multiple objects in view.	Objects detected with correct labels.	Pass
FT006	Age Detection	Face visible in camera.	Estimated age range displayed.	Pass
FT007	Anomaly Detection	Simulate a suspicious/unusual event.	Alert notification triggered.	Pass
FT008	Report Generation	Click 'Generate Report' after detection.	Report generated with data summary.	Pass
FT009	Dashboard Charts	View dashboard during detection activity.	Live stats and charts are updated.	Pass
FT010	Export Report	Click 'Export' button.	Downloadable file in CSV/PDF format is saved.	Pass

5.7 User Acceptance Testing Result

A User Acceptance Testing (UAT) form as shown in Appendix A is an essential document used during the user acceptance testing phase of a tool development or implementation project. Structured approach to collect data from 15 respondents among the FSKTM UTHM student through quantitative methods, using pass or fail answers. Table 7 until table 10 shows the result of User Acceptance Testing (UAT) for user side. In addition, all the items on the UAT have been successfully addressed in the proposed tool.

Table 7 *User Acceptance Testing Result for Section A: General Features & Dashboard*

No.	*Test Case Description*	*Expected Result*	*Pass/Fail*
A1	Real-Time Video Feed	Video feed can load.	Pass
A2	Dashboard Components	All sections are visible and aligned.	Pass
A3	Anomaly Detection	Alert is triggered and logged.	Pass
A4	Report Generation	Downloadable, correctly formatted report.	Pass
A5	Export Logs	CSV/PDF file downloads successfully.	Pass
A6	Login attempt limitation with blocking mechanism.	Tool allows only 3 login attempts. After 3 failed attempts, access is blocked.	Pass
A7	Performance	No major crashes or major delays.	Pass

Table 8 *User Acceptance Testing Result for Section B: Facial Expression Detection*

No.	*Test Case Description*	*Expected Result*	*Pass/Fail*
B1	Emotion Detection	Expression label is shown.	Pass
B2	Multiple Faces	All faces labeled with their expressions.	Pass
B3	Expression Transition	New expressions are updated smoothly.	Pass

Table 9 *User Acceptance Testing Result for Section C: Object Detection Section*

No.	*Test Case Description*	*Expected Result*	*Pass/Fail*
C1	Object Identification	Bounding box with correct label shown.	Pass
C2	Multiple Objects	All objects are detected and labeled.	Pass
C3	Object Tracking	Object labels move smoothly with object.	Pass
C4	Object Logging	Logs appear in report/export section.	Pass

Table 10 *User Acceptance Testing Result for Section D: Age Detection*

No.	*Test Case Description*	*Expected Result*	*Pass/Fail*
D1	Age Estimation	Displays estimated age (e.g., 25–30).	Pass
D2	Multiple Subjects	Age ranges appear above each person.	Pass
D3	Consistency	Age range remains consistent.	Pass

6. Conclusion

Overall, this project was successful in achieving its goal to develop an AI video detection tool that uses Convolutional Neural Networks (CNN) with YOLOv8 for age estimation, object detection, and real-time facial expression recognition. With useful features like anomaly alerts and report generation, the tool unifies several AI modules into a single platform that is backed by an intuitive dashboard. During testing, it showed smooth real-time performance and encouraging facial expression detection results.

However, a lack of a dedicated GPU was a significant hardware resource constraint for the project. Google Colab Pro with GPU acceleration was used for model training. Despite this, real-time CPU execution significantly limited the system's scalability and processing speed. Performance was impacted, especially when performing difficult tasks like identifying emotions in live videos. High processing power is needed for real-time AI video detection, and the tool's full potential is limited when deep learning models are run on CPU hardware. Another limitation is the built-in camera's low resolution, which blurs facial features and makes it challenging for the tool to accurately detect emotions in real time. In order to improve speed, accuracy, and overall system responsiveness, future developments should concentrate on optimising GPU usage for both training and deployment. Despite its limitations, the tool offers a strong basis for security and educational applications, and it can be improved with improved hardware support.

Acknowledgement

The authors would like to thank the Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia for its support.

Conflict of Interest

Authors declare that there is no conflict of interests regarding the publication of the paper.

Author Contribution

This journal requires that all authors take public responsibility for the content of the work submitted for review. The contributions of all authors must be described in the following manner:

The authors confirm contribution to the paper as follows: ***study conception and design:*** *W. S. Wan Mohd Hasanul Isyraf, N. Rahim;* ***data collection:*** *W. S. Wan Mohd Hasanul Isyraf, N. Rahim;* ***analysis and interpretation of results:*** *W. S. Wan Mohd Hasanul Isyraf, N. Rahim;* ***draft manuscript preparation:*** *W. S. Wan Mohd Hasanul Isyraf, N. Rahim. All authors reviewed the results and approved the final version of the manuscript.*

References

[1] S. Robbins, "Machine Learning, Mass Surveillance, and National Security: Data, Efficacy, and Meaningful Human Control," in *The Palgrave Handbook of National Security*, M. Clarke, A. Henschke, M. Sussex, and T. Legrand, Eds., Cham: Springer International Publishing, 2022, pp. 371–388. doi: 10.1007/978-3-030-53494-3_16.

[2] Y. Qian, "Face recognition technology for video surveillance integrated with particle swarm optimization algorithm," *International Journal of Intelligent Networks*, vol. 5, pp. 145–153, Jan. 2024, doi: 10.1016/j.ijin.2024.02.008.

[3] W. S. Admass, Y. Y. Munaye, and A. A. Diro, "Cyber security: State of the art, challenges and future directions," *Cyber Security and Applications*, vol. 2, p. 100031, 2024, doi: https://doi.org/10.1016/j.csa.2023.100031.

[4] H. Choung, P. David, and T. W. Ling, "Acceptance of AI-powered facial recognition technology in surveillance scenarios: Role of trust, security, and privacy perceptions," *Technol Soc*, vol. 79, Dec. 2024, doi: 10.1016/j.techsoc.2024.102721.

[5] V. Ilic, "The Integration of Artificial Intelligence and Computer Vision in Large-Scale Video Surveillance of Railway Stations," in *2024 IEEE Zooming Innovation in Consumer Technologies Conference, ZINC 2024*, Institute of Electrical and Electronics Engineers Inc., 2024, pp. 42–47. doi: 10.1109/ZINC61849.2024.10579411.

[6] O. Kalyta, "Information Technology of Facial Emotion Recognition for Visual Safety Surveillance," *Computer systems and information technologies*, no. 1, pp. 54–61, Apr. 2022, doi: 10.31891/csit-2022-1-7.

[7] A. M. Abbas Al-modwahi, O. Sebetela, L. Nehemiah Batleng, B. Parhizkar, and A. Habibi Lashkari, "Facial Expression Recognition Intelligent Security System for Real Time Surveillance."

[8] O. Kalyta, O. Barmak, P. Radiuk, and I. Krak, "Facial Emotion Recognition for Photo and Video Surveillance Based on Machine Learning and Visual Analytics," *Applied Sciences (Switzerland)*, vol. 13, no. 17, Sep. 2023, doi: 10.3390/app13179890.

[9] "Face^{++} - Face^{++} Cognitive Services." Accessed: Dec. 30, 2024. [Online]. Available: https://www.faceplusplus.com/

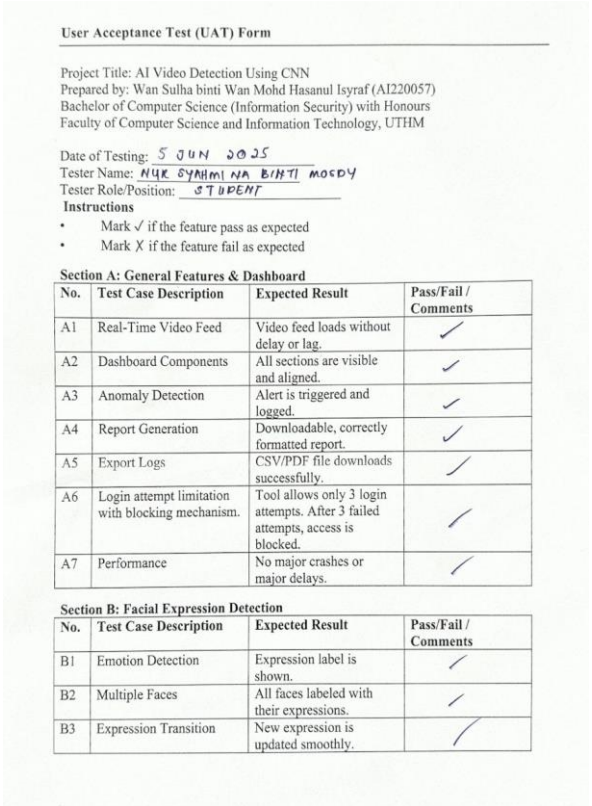
[10] "Office & Workplace Physical Security Assessment Checklist." Accessed: Dec. 30, 2024. [Online]. Available: https://www.avigilon.com/blog/office-security-safety-audit

[11] "BriefCam software provides quantitative analysis of video | BriefCam." Accessed: Dec. 30, 2024. [Online]. Available: https://www.briefcam.com/solutions/planning-research/

[12] D. Al-Obidi and S. Kacmaz, "Facial Features Recognition Based on Their Shape and Color Using YOLOv8," in *Proc. 7th Int. Symp. Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, 2023. [Online]. Available: https://doi.org/10.1109/ISMSIT58785.2023.10304905

[13] S. Ren, M. Sun, B. Wang, M. Liu, and S. Men, "High Precision Infant Facial Expression Recognition by Improved YOLOv8," *IEEE Access*, vol. 13, pp. 39621–39630, 2025. [Online]. Available: https://doi.org/10.1109/ACCESS.2025.3543950

Appendix A: UAT Result for AI Video Detection



User Acceptance Test (UAT) Form

Project Title: AI Video Detection Using CNN
Prepared by: Wan Sulha binti Wan Mohd Hasanul Isyraf (AI220057)
Bachelor of Computer Science (Information Security) with Honours
Faculty of Computer Science and Information Technology, UTHM

Date of Testing: 5 JUN 2025
Tester Name: NUR SYAHMINA BINTI MOSDY
Tester Role/Position: STUDENT

Instructions
- Mark ✓ if the feature pass as expected
- Mark X if the feature fail as expected

Section A: General Features & Dashboard

No.	Test Case Description	Expected Result	Pass/Fail / Comments
A1	Real-Time Video Feed	Video feed loads without delay or lag.	✓
A2	Dashboard Components	All sections are visible and aligned.	✓
A3	Anomaly Detection	Alert is triggered and logged.	✓
A4	Report Generation	Downloadable, correctly formatted report.	✓
A5	Export Logs	CSV/PDF file downloads successfully.	✓
A6	Login attempt limitation with blocking mechanism.	Tool allows only 3 login attempts. After 3 failed attempts, access is blocked.	✓
A7	Performance	No major crashes or major delays.	✓

Section B: Facial Expression Detection

No.	Test Case Description	Expected Result	Pass/Fail / Comments
B1	Emotion Detection	Expression label is shown.	✓
B2	Multiple Faces	All faces labeled with their expressions.	✓
B3	Expression Transition	New expression is updated smoothly.	✓

Section C: Object Detection

No.	Test Case Description	Expected Result	Pass/Fail / Comments
C1	Object Identification	Bounding box with correct label shown.	✓
C2	Multiple Objects	All objects are detected and labeled.	✓
C3	Object Tracking	Object labels move smoothly with object.	✓
C4	Object Logging	Logs appear in report/export section.	✓

Section D: Age Detection

No.	Test Case Description	Expected Result	Pass/Fail / Comments
D1	Age Estimation	Displays estimated age (e.g., 25–30).	✓
D2	Multiple Subjects	Age ranges appear above each person.	✓
D3	Consistency	Age range remains consistent.	✓

Final Evaluation

Category	Result
Functional Accuracy	[✓] Pass [ ] Partial [ ] Fail
Performance	[✓] Pass [ ] Partial [ ] Fail
Usability	[✓] Pass [ ] Partial [ ] Fail
Overall Decision	[✓] Accept [ ] Accept with Minor Fixes [ ] Reject

Tester Confirmation:
I hereby confirm that I have conducted the user acceptance testing as outlined above and provided honest, comprehensive feedback.

Tester Name: NUR SYAHMINA BINTI MOSDY
Signature:
Date: 5 JUN 2025