EMBEDDED AI-BASED IN-CABIN SURVEILLANCE SYSTEM FOR FULLY AUTONOMOUS CABS

A Project Report Submitted in Partial Fulfilment of the Requirements for the Award of the Degree of

BACHELOR OF TECHNOLOGY

in

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

AKAASH R P	**22BEC1072**
VAMSI KRISHNA B	**22BEC1130**
MRITHYUMJAI M P	**22BEC1132**

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Electronics Engineering
Vellore Institute of Technology, Chennai
Vandalur–Kelambakkam Road, Chennai – 600127, India.

November 2025

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. Revathi S**, *Professor, School of Electronics Engineering*, for her consistent encouragement and valuable guidance.

We are extremely grateful to **Dr. Ravi Sankar A** (Dean), **Dr. Mohana Prasad K** (Associate Dean - Academics), and **Dr. Chitra K** (Associate Dean - Research), of the School of Electronics Engineering, VIT Chennai, for their support.

I also take this opportunity to thank all the faculty of the School for their support and wisdom imparted to me throughout the course. I thank my parents, family, and friends for bearing with me throughout the course of my project and for the opportunity they provided me in undergoing this course in such a prestigious institution.

Akaash R P (22BEC1072)
Vamsi Krishna B (22BEC1130)
Mrithyumjai M P (22BEC1132)

ABSTRACT

The rapid advancement of autonomous vehicle technology has created an urgent need for robust in-cabin monitoring systems that can ensure passenger safety in fully autonomous cabs. This project presents an embedded AI-based surveillance system designed specifically for real-time violence detection and passenger behavior classification within autonomous vehicle cabins. The system addresses critical challenges including limited computational resources on edge devices, real-time processing requirements, and the need for privacy-preserving solutions that operate without cloud dependency.

The proposed system integrates YOLOv8 for efficient human detection and occupancy counting with a lightweight CNN+LSTM architecture utilizing MobileNetV2 as the feature extraction backbone. This hybrid approach enables the system to classify passenger behaviors into four distinct categories: violence with object, violence without object, non-violence with object, and non-violence without object. The violence detection module is intelligently triggered only when multiple passengers are detected, optimizing computational efficiency.

The system was trained and evaluated on the INCAR dataset, comprising 1,175 sessions with over 152,000 frames across 20 action classes. Through systematic preprocessing, data augmentation, and model optimization techniques, the final model achieved a test accuracy of 86.93% on 20-class classification with an average AUC of 0.98 across all classes. The model was successfully converted to TensorFlow Lite format and optimized for deployment on Raspberry Pi 5, enabling real-time inference at acceptable frame rates for practical in-vehicle monitoring applications.

Key innovations of this work include: (1) a conditional detection pipeline that activates violence classification only when necessary, (2) integration of object detection to distinguish violence scenarios with and without weapons, (3) a lightweight architecture suitable for embedded deployment, and (4) a complete end-to-end pipeline from data preprocessing to model deployment. The system demonstrates the feasibility of implementing sophisticated AI-based surveillance on resource-constrained edge devices while maintaining privacy through local processing. This work contributes to the development of safer autonomous transportation systems by providing an effective, scalable, and deployable solution for in-cabin monitoring.

TABLE OF CONTENTS

LIST OF SYMBOLS, ABBREVIATIONS AND NOMENCLATURE

- **CNN:** Convolutional Neural Network
- **LSTM:** Long Short-Term Memory
- **YOLOv8:** You Only Look Once version 8
- **TFLite:** TensorFlow Lite
- **AUC:** Area Under Curve
- **FPS:** Frames Per Second
- **RGB:** Red Green Blue
- **GPU:** Graphics Processing Unit
- **API:** Application Programming Interface
- **IoT:** Internet of Things

LIST OF FIGURES

1 CHAPTER 1: INTRODUCTION

1.1 BACKGROUND AND MOTIVATION

The emergence of fully autonomous vehicles represents one of the most transformative technological shifts in modern transportation. As autonomous cabs and ride-sharing services prepare to operate without human drivers, ensuring passenger safety has become a paramount concern. Traditional in-vehicle monitoring systems that relied on driver vigilance are no longer applicable in driver-less scenarios, necessitating the development of sophisticated AI-based surveillance solutions.

The in-cabin environment of autonomous vehicles presents unique challenges for safety monitoring. Unlike stationary surveillance systems, vehicle cabins are dynamic spaces with varying lighting conditions, passenger movements, occlusion scenarios, and privacy concerns. Furthermore, the computational constraints of embedded systems require efficient algorithms that can operate in real-time without relying on cloud connectivity, which may be unreliable or introduce unacceptable latency during critical safety incidents.

Current solutions in the market predominantly focus on basic occupancy detection or simple activity recognition, failing to address the nuanced requirements of violence detection and threat assessment in autonomous vehicle contexts. The lack of lightweight, deployable models optimized for edge devices such as Raspberry Pi represents a significant gap in the field. In addition, existing approaches often depend on expensive hardware or cloud-based processing, making them impractical for widespread deployment in commercial autonomous vehicle fleets.

1.2 PROBLEM STATEMENT

The development of effective in-cabin surveillance for fully autonomous cabs faces several critical challenges:

- **Computational Constraints:** Current deep learning models for violence detection typically require high-end GPUs and are not suitable for deployment on edge devices like the Raspberry Pi, which have limited computational power, memory, and energy resources.
- **Real-time Processing Requirements:** Safety-critical applications demand immediate detection and response. Cloud-based solutions introduce network latency that can delay emergency alerts, potentially compromising passenger safety during violent incidents.
- **Behavioral Classification Complexity:** Accurately distinguishing between normal passenger activities, aggressive behaviors without weapons, and violent actions involving dangerous objects requires sophisticated temporal modeling that can capture subtle motion patterns across video sequences.
- **Lighting and Occlusion Challenges:** In-cabin conditions vary significantly with time of day, weather, and passenger positioning. Models must be robust to low-light scenarios, partial occlusions, and varied passenger postures without sacrificing accuracy.
- **Privacy Preservation:** Transmitting raw video data to cloud servers raises serious privacy concerns. An ideal solution must perform all processing locally on the edge device, ensuring that passenger data never leave the vehicle.
- **Scalability and Cost:** For deployment across large autonomous vehicle fleets, the solution must be cost-effective, easily maintainable, and scalable, ruling out specialized hardware or complex multi-sensor setups.

1.3 OBJECTIVES

The primary objectives of this project are:

1. Design and implement a lightweight CNN+LSTM architecture that can effectively capture spatial-temporal patterns in video sequences while remaining computationally efficient for edge deployment.

2. Integrate object detection capabilities using YOLOv8 to enable intelligent triggering of violence detection only when multiple passengers are present, and to identify dangerous objects that may be involved in violent incidents.

3. Develop a comprehensive data processing pipeline that handles the INCAR dataset's preprocessing, augmentation, and organization to support effective model training across 20 action classes.

4. Optimize the trained model for Raspberry Pi 5 deployment through TensorFlow conversion, quantization, and performance tuning to achieve real-time inference capabilities.

5. Create a complete end-to-end system that encompasses video capture, passenger detection, action classification, object identification, and visual annotation for monitoring purposes.

6. Evaluate system performance through rigorous testing on validation datasets and real-world video scenarios, measuring accuracy, inference speed, and robustness under various conditions.

1.4 SCOPE OF THE PROJECT

This project focuses specifically on in-cabin surveillance for autonomous vehicles and addresses the following scope:

Technical Scope: The system is designed to run on Raspberry Pi 5 hardware with standard camera modules, demonstrating that sophisticated AI surveillance can be achieved on affordable, accessible hardware platforms. The solution encompasses the complete machine learning pipeline from data preprocessing through model deployment.

Functional Scope: The system performs four primary functions: (1) continuous passenger detection and counting, (2) real-time violence classification when multiple passengers are present, (3) dangerous object identification, and (4) video annotation with bounding boxes and classification labels. The system categorizes behaviors into four classes: violence with object, violence without object, non-violence with object, and non-violence without object.

Dataset Scope: The project utilizes the INCAR dataset, which contains real in-vehicle recorded scenarios across 20 distinct action classes, with over 152,000 frames collected from 1,175 sessions. This dataset provides comprehensive coverage of various passenger behaviors, lighting conditions, and occupancy scenarios.

Deployment Scope: The final deliverable is a TensorFlow Lite model optimized for edge inference on Raspberry Pi 5, complete with a Python-based inference pipeline that processes video streams, generates annotated outputs, and can trigger alerts when violent behavior is detected.

Limitations: This project focuses on violence detection and does not address other potential in-cabin monitoring needs such as health emergencies, contraband detection, or biometric authentication. The system is designed for daytime and well-lit conditions; performance in extremely low-light scenarios may require additional infrared camera support.

2 CHAPTER 2: LITERATURE REVIEW

2.1 INTRODUCTION

The field of in-vehicle behavior monitoring and violence detection has evolved significantly over the past decade, driven by advances in deep learning, computer vision, and edge computing technologies. This literature review examines existing research in violence detection, activity recognition, and in-cabin monitoring systems, identifying the gaps that motivated our proposed approach.

2.2 VIOLENCE DETECTION IN VIDEOS

Rodrigues et al. (2023) proposed a hybrid model combining Temporal Segment Networks (TSN), Temporal Shift Modules (TSM), and YOLOv5 for detecting violent actions inside vehicles [1]. Their approach achieved impressive accuracies of 95.4% with TSN and 94.3% with TSM. However, the study lacked deployment benchmarks on edge devices, focusing solely on high-performance GPU implementations. The models' computational requirements made them unsuitable for embedded systems, limiting their practical applicability in resource-constrained autonomous vehicle contexts.

Abdali and Al-Tuma (2019) developed a CNN-LSTM architecture using VGG19 as the feature extraction backbone for violent activity recognition [2]. They achieved 98% accuracy on the Hockey Fight dataset with 131 FPS performance on GTX1060 GPU. While their approach balanced speed and accuracy effectively, the dataset was limited to sports contexts, and generalizability to in-vehicle scenarios remained unvalidated. The VGG19 backbone, though accurate, is computationally heavy for edge deployment.

Zhang et al. (2021) explored CNN-RNN, CNN-LSTM, and attention-based models for hostile activity detection [3]. Their attention-based approach achieved 83% accuracy, while CNN-LSTM reached 80%. However, these accuracies were lower than other contemporary models, and the study lacked real-time performance metrics and edge device validation, limiting insights into deployment feasibility.

2.3 IN-CABIN MONITORING SYSTEMS

Mishra et al. (2022) built an object detection system using YOLOv2/v3, Tiny-YOLO, and Faster R-CNN on a 333,000-frame dataset for in-cabin monitoring[4]. Tiny-YOLO achieved 96% accuracy with the fastest inference time of 0.163 seconds. While this work demonstrated efficient object detection, it focused solely on detecting objects and did not address passenger behavior classification or violence detection, leaving a significant gap in comprehensive in-cabin safety monitoring.

Lin et al. (2024) developed a dual-camera system with modified 3D ResNet for autonomous minibus monitoring[5]. They achieved approximately 80% pose accuracy with reliable static action classification. However, runtime was only 5 FPS on RTX 2070, making it unsuitable for real-time edge deployment. The reliance on dual cameras also increased system complexity and cost.

Tseng and Lin (2022) introduced the BUS-HAR dataset and applied 3D CNN with Region Activation Network (RAN) and Feature Pyramid Network (FPN) for abnormal behavior recognition in buses[6]. They achieved remarkable 97% accuracy in violent action detection. However, the model required high-end GPUs and was unsuitable for real-time or embedded systems due to its computational demands.

2.4 TEMPORAL MODELING APPROACHES

Yang et al. (2019) compared 2D and 3D CNNs for video classification, focusing on spatio-temporal feature extraction[7]. They found that 3D CNNs extract richer spatio-temporal features but require significantly higher computational resources, while 2D CNNs with temporal modeling (like LSTM) are more efficient and edge-friendly. This insight guided our decision to use 2D CNN (MobileNetV2) with LSTM for temporal modeling.

2.5 BENCHMARK DATASETS

The InCar & InVicon Consortium (2022) introduced two comprehensive benchmark datasets specifically for in-vehicle activity recognition[8]. These datasets contain over 6,400 videos with more than 3 million frames across 58 action classes captured using multiple modalities. While valuable for benchmarking, the datasets themselves did not propose lightweight or deployable solutions optimized for embedded platforms.

2.6 IDENTIFIED RESEARCH GAPS

Through comprehensive analysis of existing literature, we identified several critical gaps:

1. **Lack of Edge-Optimized Models:** Most high-accuracy models require high-end GPUs, with limited research on lightweight architectures suitable for Raspberry Pi or similar edge devices.
2. **Insufficient Real-World Deployment:** Many studies achieve high accuracy in controlled environments but lack validation on actual embedded hardware with real-time constraints.
3. **Limited Integration:** Existing works typically focus on either object detection or action recognition separately, without integrated solutions that combine both for comprehensive threat assessment.
4. **Privacy-Preserving Approaches:** Most solutions assume cloud connectivity or powerful on-board computers, ignoring the growing demand for privacy-preserving, local-processing solutions.
5. **Conditional Processing:** No existing work implements intelligent conditional activation of violence detection based on occupancy, leading to unnecessary computational overhead during single-passenger scenarios.

2.7 OUR PROPOSED APPROACH

Our work addresses these gaps by:

- Integrating YOLOv8 for human detection and passenger counting, enabling intelligent triggering of violence detection only when multiple passengers are present
- Employing a lightweight CNN+LSTM architecture using MobileNetV2 for efficient spatial feature extraction and LSTM layers for temporal modeling
- Implementing a four-class classification system that distinguishes between violence with/without objects and non-violence with/without objects
- Optimizing for Raspberry Pi 5 through TensorFlow Lite conversion, quantization, and performance tuning
- Ensuring privacy through complete local processing with no cloud dependency
- Validating on the INCAR dataset with comprehensive evaluation across diverse in-cabin scenarios

3 CHAPTER 3: DATASET DESCRIPTION

3.1 INCAR DATASET OVERVIEW

The INCAR dataset serves as the foundation for training and evaluating our in-cabin surveillance system. This comprehensive dataset was specifically designed for in-vehicle activity recognition and violence detection research, addressing the unique challenges of autonomous vehicle cabin environments.

Dataset Composition: The complete INCAR dataset contains:

- **Total Categories:** 4 primary behavior categories
- **Total Sessions:** 1,175 recording sessions
- **Total Frames:** 152,406 individual frames
- **Average Frames per Session:** 129.7 frames
- **Action Classes:** 20 distinct classes (C1 through C20)

3.2 CLASS DISTRIBUTION

The dataset exhibits a relatively balanced distribution across its 20 classes, organized into four main categories:

- **Violence Without Object Classes (C1-C4):**
  - C1: 56 sessions, 7,200 files
  - C2: 56 sessions, 7,238 files
  - C3: 56 sessions, 7,280 files
  - C4: 56 sessions, 7,280 files
- **Violence With Object Classes (C5-C12):**
  - C5-C12: Each class contains 60-61 sessions with approximately 7,800-8,200 files per class
  - Total: 485 sessions, 62,924 files
- **Non-Violence Without Object Classes (C13-C14):**
  - C13: 64 sessions, 8,320 files
  - C14: 64 sessions, 8,320 files
  - Total: 128 sessions, 16,640 files
- **Non-Violence With Object Classes (C15-C20):**
  - C15-C19: 60-64 sessions each, 7,722-8,320 files per class
  - C20: 20 sessions, 2,600 files (smallest class)
  - Total: 334 sessions, 43,194 files

3.3 DATA STRUCTURE AND ORGANIZATION

Each session in the dataset follows a hierarchical directory structure:

- **Frame Types:**
  - rgb_sampled: 30 uniformly sampled frames per session for efficient training
  - rgb: Complete 100-frame sequences for comprehensive analysis

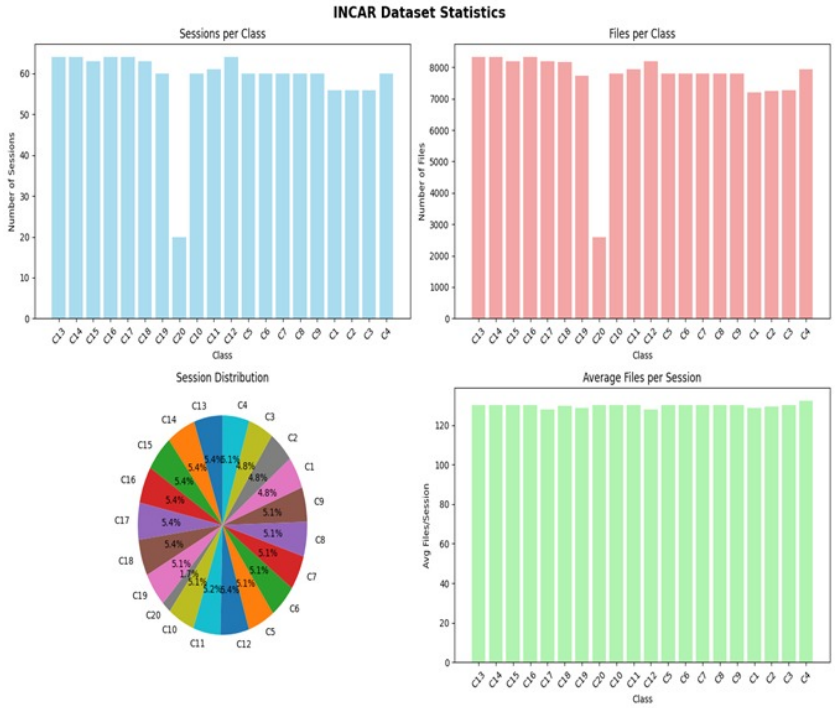


Figure 1: DATASET DISTRIBUTION

3.4 STATISTICAL ANALYSIS

Session Distribution:

- Most sessions per class: C10, C11, C13, C14, C15, C16, C17, C18, C19 (64 sessions each)
- Least sessions per class: C20 (20 sessions)
- Dataset balance ratio: 3.2:1 (reasonably balanced)

Files Distribution:

- Maximum files: C13, C14, C16 (8,320 files each)
- Minimum files: C20 (2,600 files)
- Average files per class: 7,620 files

Statistical Insights:

- Most sessions: C13 (64 sessions)
- Least sessions: C20 (20 sessions)
- Most files: C13 (8320 files)
- Least files: C20 (2600 files)
- Dataset balance ratio: 3.20:1

File Type Analysis

- rgb: 1175 occurrences
- rgb_sampled: 1175 occurrences
- rgb_sampled_sampled: 1 occurrences

File Extensions:

- .jpg: 152436 files

3.5 DATA CHARACTERISTICS

Temporal Characteristics:

- Each session captures a continuous behavioral sequence
- Frame rate: Variable, sampled to 30 frames for consistency
- Sequence length: 16 frames used for model training (balancing temporal context and computational efficiency)

Visual Characteristics:

- Original Resolution: Variable (dataset contains mixed resolutions)
- Preprocessed Resolution: 160×160 pixels (optimized for MobileNetV2)
- Color Space: RGB
- Lighting Conditions: Natural cabin lighting with variations across sessions

Behavioral Diversity: The dataset captures realistic in-cabin scenarios including:

- Normal passenger interactions
- Aggressive gestures without weapons
- Violent actions involving objects (simulated weapons)
- Multiple passenger configurations
- Various seating positions and camera angles

3.6 DATA PREPROCESSING PIPELINE

Our preprocessing pipeline transforms raw dataset frames into model-ready inputs:

1. **Frame Sampling:** Extract 16 uniformly distributed frames from each session using numpy.linspace for consistent temporal sampling
2. **Resizing:** Resize all frames to 160×160 pixels
3. **Color Conversion:** Convert BGR to RGB color space
4. **Normalization:** Scale pixel values to [0, 1] range by dividing by 255.0
5. **Label Encoding:** Convert class names (C1-C20) to one-hot encoded vectors
6. **Sequence Formation:** Stack frames into (16, 160, 160, 3) tensors

3.7 TRAIN-VALIDATION-TEST SPLIT

To ensure robust model evaluation, we implemented stratified splitting:

- **Training Set:** 80% of sessions (940 sessions)
- **Validation Set:** 20% of sessions (235 sessions)
- **Stratification:** Maintained proportional class distribution across splits
- **Random Seed:** Fixed at 42 for reproducibility

This ensures that each class is proportionally represented in training and validation sets, preventing bias toward overrepresented classes.

4 CHAPTER 4: SYSTEM DESIGN AND ARCHITECTURE

4.1 SYSTEM OVERVIEW

The proposed in-cabin surveillance system follows a modular pipeline architecture consisting of six primary components: input acquisition, preprocessing, feature extraction, temporal modeling, classification, and deployment. This design ensures scalability, maintainability, and optimal performance on resource-constrained edge devices.

4.2 SYSTEM ARCHITECTURE

Overall Pipeline:

1. Video Input → Preprocessing → Feature Extraction → Temporal Modeling → Classification → Output
2. Parallel Object Detection → Conditional Violence Detection Activation

4.3 MODULE DESCRIPTIONS

4.3.1 Input Module

Functionality: Captures video streams from Raspberry Pi Camera Module and extracts frame sequences.
Specifications:

- Input Source: Raspberry Pi Camera Module v2/v3
- Frame Rate: 30 FPS
- Resolution: 1920×1080 (downsampled during preprocessing)
- Buffer Size: 16-frame sliding window using deque structure

4.3.2 Preprocessing Module

Functionality: Standardizes input frames to model requirements.
Operations:

1. Frame resize: 160×160 pixels (MobileNetV2 input requirement)
2. Color space conversion: BGR → RGB
3. Normalization: Pixel values scaled to [0, 1]
4. Frame stacking: Create (16, 160, 160, 3) tensors

Rationale: 160×160 resolution balances computational efficiency with sufficient spatial detail for action recognition. MobileNetV2 accepts variable input sizes; we chose 160×160 as it provides 4x reduction from 320×320 while preserving essential visual features.

4.3.3 Feature Extraction Module (MobileNetV2)

Architecture: MobileNetV2 serves as the spatial feature extractor.
Key Characteristics:

- Input: (160, 160, 3) RGB frames
- Architecture: Depthwise separable convolutions
- Pooling: Global Average Pooling
- Output: 1280-dimensional feature vectors
- Weights: Pre-trained on ImageNet (transfer learning)
- Trainable: Frozen during initial training (trainable=False)

Advantages:

- Efficient: Uses depthwise separable convolutions reducing parameters by 8-9x compared to standard convolutions
- Lightweight: Only 3.4M parameters
- Fast: Optimized for mobile and embedded devices
- Effective: Strong feature representations from ImageNet pre-training

4.3.4 Temporal Modeling Module (LSTM)

Architecture: Two-layer LSTM network for sequential learning.
Configuration:

- LSTM Layer 1: 128 units, return_sequences=True
- Dropout 1: 50% dropout rate
- LSTM Layer 2: 64 units, return_sequences=False
- Dropout 2: 50% dropout rate

Functionality:

- Processes sequences of 16 feature vectors (16, 1280)
- Captures temporal dependencies and motion patterns
- Models long-term dependencies through memory cells
- Outputs 64-dimensional context-aware representations

4.3.5 Classification Module

Architecture: Fully connected layers with softmax activation.
Configuration:

- Dense Layer 1: 64 units, ReLU activation
- Dropout 3: 30% dropout
- Output Layer: 20 units (classes), Softmax activation
- Output: Probability distribution over 20 action classes
- Loss Function: Categorical cross-entropy
- Optimizer: Adam with learning rate 0.0001
- Metrics: Accuracy, Top-3 Accuracy

4.3.6 Object Detection Module (YOLOv8)

Functionality: Detects persons and potentially dangerous objects.
Configuration:

- Model: YOLOv8 Nano (lightweight variant)
- Confidence Threshold: 0.5

Target Classes:

- Person (COCO class 0) for occupancy detection
- Dangerous objects: knife, gun, scissors, pistol

Integration: Runs in parallel with main classification pipeline.
Conditional Processing Logic:

- If persons detected $>$ 2: Skip violence classification
- If persons detected $>=$ 2: Activate full violence detection pipeline

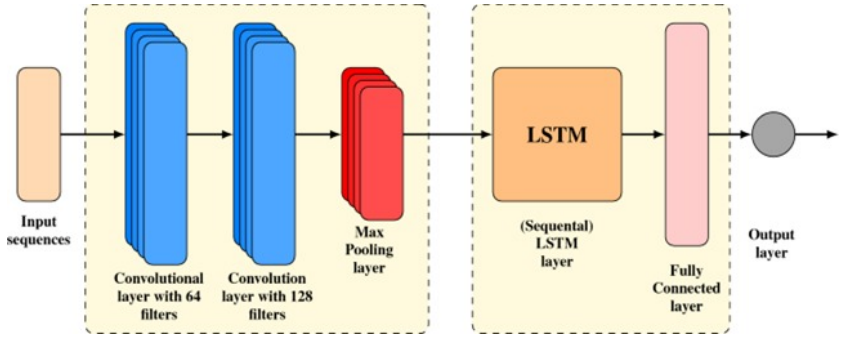


Figure 2: Architecture

4.3.7 Deployment Module

TensorFlow Lite Conversion:

- Model quantization: FP16 precision
- Optimization: Default optimization
- Operators: SELECT_TF_OPS enabled for LSTM compatibility
- Final Model Size: 3.8 MB

4.4 COMPLETE MODEL ARCHITECTURE

Full Architecture Summary:

- Input: (16, 160, 160, 3) video sequence
- MobileNetV2 Feature Extractor: 1280-dimensional features per frame
- LSTM Layers: 128 $\rightarrow$ 64 units with dropout
- Classification Head: 64-unit Dense $\rightarrow$ 20-unit Softmax
- Total Parameters: 4.2M (3.4M frozen, 0.8M trainable)

4.5 SYSTEM WORKFLOW

Real-time Inference Workflow:

1. Video Capture: Capture frame from Pi Camera
2. Passenger Detection: Run YOLOv8 to detect persons
3. Conditional Activation:
    - If passengers $>= 2$: Continue to step 4
    - If passengers < 2: Skip to step 8
4. Frame Buffering: Add frame to sliding window
5. Sequence Ready Check: If buffer contains 16 frames, continue
6. Feature Extraction: Process frames through MobileNetV2
7. Temporal Modeling: Pass features through LSTM
8. Classification: Generate action predictions
9. Object Detection: Detect dangerous objects

10. Label Generation: Combine action + object detection results
11. Visualization: Draw bounding boxes and labels
12. Output: Display/save annotated video

5 CHAPTER 5: IMPLEMENTATION DETAILS

5.1 DEVELOPMENT ENVIRONMENT

5.1.1 Hardware:

- **Training:** Google Colab (NVIDIA T4 GPU, 16GB GPU RAM)
- **Deployment:** Raspberry Pi 5 Model B (8GB RAM)
- **Camera:** Raspberry Pi Camera Module v2

5.1.2 Software Stack:

- Python 3.10
- TensorFlow 2.15
- Keras 2.15
- OpenCV 4.8
- Ultralytics YOLOv8
- NumPy 1.24
- Scikit-learn 1.3

5.1.3 Configuration Class:

```
class Config:
    IMG_HEIGHT = 160
    IMG_WIDTH = 160
    SEQUENCE_LENGTH = 16
    NUM_CLASSES = 20
    LSTM_UNITS = 128
    DROPOUT_RATE = 0.5
    BATCH_SIZE = 8
    EPOCHS = 50
    LEARNING_RATE = 0.0001
    CLASS_NAMES = [f'C{i}' for i in range(1, 21)]
    DANGEROUS_OBJECTS = ['knife', 'gun', 'scissors', 'pistol']
    YOLO_CONF_THRESHOLD = 0.5
    PERSON_CLASS_ID = 0
```

5.2 DATA LOADING AND PREPROCESSING

5.2.1 Custom Data Generator:

Our implementation uses TensorFlow's Dataset API for efficient data loading:

```
def create_dataset(sessions, labels, sequence_length=16):
    dataset = tf.data.Dataset.from_generator(
        data_generator,
        output_signature=(
            tf.TensorSpec(shape=(sequence_length, 160, 160, 3),
                          dtype=tf.float32),
            tf.TensorSpec(shape=(20,), dtype=tf.float32)
        ),
        args=[sessions, labels, sequence_length]
    )
    return dataset
```

```
def data_generator(sessions, labels, sequence_length):
    for session in sessions:
        frames = load_session_frames(session)
        # Sequence creation with sliding window
        for i in range(0, len(frames) - sequence_length + 1, 8):
            sequence = frames[i:i + sequence_length]
            yield sequence, labels[session]
```

5.2.2 Efficient Pipeline:

```
def create_efficient_pipeline(train_sessions, train_labels,
                              val_sessions, val_labels, batch_size=8):
    # Training dataset
    train_dataset = create_dataset(train_sessions, train_labels)
    train_dataset = train_dataset.shuffle(1000)
    train_dataset = train_dataset.batch(batch_size)
    train_dataset = train_dataset.prefetch(tf.data.AUTOTUNE)

    # Validation dataset
    val_dataset = create_dataset(val_sessions, val_labels)
    val_dataset = val_dataset.batch(batch_size)
    val_dataset = val_dataset.prefetch(tf.data.AUTOTUNE)

    return train_dataset, val_dataset
```

5.2.3 Benefits:

- **Asynchronous data loading** prevents GPU starvation
- **Prefetching** prepares next batch during current training step
- **Shuffling** prevents overfitting to data order

5.3 MODEL TRAINING CONFIGURATION

5.3.1 Hyperparameters:

- **Sequence Length:** 16 frames
- **Image Size:** 160×160 pixels
- **Batch Size:** 8
- **Learning Rate:** 0.0001
- **LSTM Units:** 128 (Layer 1), 64 (Layer 2)
- **Dropout:** 0.5 (LSTM), 0.3 (Dense)
- **Epochs:** 50 (with early stopping)

VIDEO PROCESSING

5.3.2 Callbacks:

1. **ModelCheckpoint:**

```
ModelCheckpoint(
    'best_model.h5',
    monitor='val_accuracy',
    save_best_only=True,
```

Layer (type)	Output Shape	Param #
video_input (InputLayer)	(None, 16, 160, 160, 3)	0
spatial_features (TimeDistributed)	(None, 16, 1280)	2,257,984
lstm_1 (LSTM)	(None, 16, 128)	721,408
dropout_1 (Dropout)	(None, 16, 128)	0
lstm_2 (LSTM)	(None, 64)	49,408
dropout_2 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 64)	4,160
dropout_3 (Dropout)	(None, 64)	0
output (Dense)	(None, 20)	1,300

Figure 3: MODEL PARAMETERS

```
    mode='max',
    verbose=1
)
```

2. **EarlyStopping:**

```
EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True,
    verbose=1
)
```

3. **ReduceLROnPlateau:**

```
ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=5,
    min_lr=1e-7,
    verbose=1
)
```

5.4 TRAINING PROCESS

5.4.1 Data Split:

- **Training:** 940 sessions (80%)
- **Validation:** 235 sessions (20%)
- **Steps per epoch:** 117 (940 / 8)
- **Validation steps:** 29 (235 / 8)

5.4.2 Training Duration:

50 epochs (stopped at epoch 50 due to early stopping)

5.4.3 Training Stability:

- Learning rate scheduling prevented over-aggressive updates
- Dropout regularization controlled overfitting
- Data augmentation (inherent in varied sessions) improved generalization

5.5 MODEL EVALUATION METRICS

5.5.1 Primary Metrics:

1. **Accuracy:** Percentage of correctly classified sequences
2. **Top-3 Accuracy:** Whether true class is in top-3 predictions
3. **Loss:** Categorical cross-entropy

5.5.2 Advanced Metrics:

1. **Confusion Matrix:** Class-wise performance analysis
2. **ROC Curves:** Per-class discrimination capability
3. **AUC Scores:** Area under ROC curves for each class

5.6 VIDEO PROCESSING IMPLEMENTATION

5.6.1 Person Tracking:

```
def track_people(frame, yolo_model):
    results = yolo_model(frame, classes=[0])  # Class 0 = person
    detections = []

    for box in results[0].boxes:
        x1, y1, x2, y2 = box.xyxy[0].cpu().numpy()
        conf = box.conf[0].cpu().numpy()
        detections.append({
            'bbox': [x1, y1, x2, y2],
            'confidence': conf,
            'centroid': [(x1 + x2) / 2, (y1 + y2) / 2]
        })

    return detections
```

5.6.2 Four-Class Logic:

```
def classify_violence_type(violence_pred, object_detected):
    if violence_pred > 0.5 and object_detected:
        return "Violence with Object"
    elif violence_pred > 0.5 and not object_detected:
        return "Violence without Object"
    elif violence_pred <= 0.5 and object_detected:
        return "Non-Violence with Object"
    else:
        return "Non-Violence without Object"
```

5.7 DEPLOYMENT ON RASPBERRY PI

5.7.1 Hardware Setup

The hardware platform for the violence detection system is based on the **Raspberry Pi 5 Model B**, the latest generation in Raspberry Pi single-board computers. This board offers improved CPU and GPU capabilities, enabling real-time computer vision and machine learning on edge devices.

The imaging hardware is the **Raspberry Pi HQ Camera Module**, which is connected via the **Camera Serial Interface (CSI) ribbon cable** to the dedicated CSI port on the Raspberry Pi. The Raspberry Pi 5 features dual CSI ports with 22-pin connectors, differing from previous 15-pin designs, requiring either an updated cable or an adapter compatible with official modules.

The camera module includes a 12.3-megapixel Sony IMX477 sensor capable of capturing high-quality stills and video input. The CSI ribbon cable must be carefully installed, ensuring that the metal contacts align toward the Ethernet port and that the cable connector locking clip is securely fastened to maintain a reliable data connection.

Hardware Setup Details:

- Power off the Raspberry Pi before installation
- Gently lift the CSI connector locking clip
- Insert the CSI cable with metal contacts facing the correct direction
- Secure the connector by closing the locking clip
- Connect the other end of the CSI cable securely to the camera module
- Verify no tension or bending on the cable to prevent intermittent disconnections

This setup ensures a stable, high-bandwidth connection essential for real-time image acquisition.

5.7.2 Software Environment

The software environment is based on **Raspberry Pi OS Trxie**, which ships with **Python 3.13** as the system default interpreter. Utilizing the system Python is necessary to ensure compatibility with the pre-installed **libcamera** Python bindings critical for Picamera2 camera control.

The camera interface is implemented using the official **Picamera2** Python library, which interacts with the underlying libcamera daemon to control camera parameters, capture images, and configure streaming pipelines.

For machine learning inference, the project uses **full TensorFlow** (installed via pip) as the **TensorFlow Lite runtime wheel for Python 3.13 ARM64 is not yet available**. This choice permits running TensorFlow models with slightly higher resource usage but ensures compatibility with Python 3.13 and ARM architecture.

To coordinate Python library dependencies and access to system-wide camera bindings, a **virtual environment** is created with the `--system-site-packages` option. This allows importing system-installed packages like `libcamera` into the isolated virtual environment ensuring clean management of additional Python packages while maintaining access to essential system modules.

```
# Create virtual environment with system packages
python -m venv violence_detection_env --system-site-packages
source violence_detection_env/bin/activate

# Install required packages
pip install tensorflow
pip install opencv-python
pip install ultralytics
pip install numpy
```

5.7.3 Camera Configuration

The camera is configured primarily for **640x480 resolution**, balancing the trade-off between latency, computational load, and sufficient image quality required for accurate violence detection.

Higher resolutions increase the computational burden for both capture and model inference, reducing achievable frame rates. By selecting 640x480, the system achieves a reasonable frame processing rate of approximately 3-10 FPS.

```
from picamera2 import Picamera2
import time

def initialize_camera():
    picam2 = Picamera2()

    # Configure camera for video capture
    config = picam2.create_video_configuration(
        main={"size": (640, 480), "format": "RGB888"},
        controls={"FrameRate": 30}
    )
    picam2.configure(config)
    picam2.start()

    # Allow camera to warm up
    time.sleep(2)
    return picam2

# Camera verification
def verify_camera():
    try:
        picam2 = Picamera2()
        camera_properties = picam2.camera_properties
        print("Camera detected:", camera_properties)
        return True
    except Exception as e:
        print("Camera not detected:", e)
        return False
```

This process ensures kernel modules and libcamera configurations are correctly initialized to support the camera hardware.

6 CHAPTER 6: RESULTS AND DISCUSSION

6.1 Training Performance

6.1.1 Final Training Metrics

- Training Accuracy: 84.41%
- Training Loss: 0.4882
- Validation Accuracy: 84.48%
- Validation Loss: 0.5098
- Top-3 Training Accuracy: 98.01%

6.1.2 Test Performance

- Test Accuracy: 86.93%
- Test Loss: 0.5071
- Top-3 Test Accuracy: 97.79%

Analysis: The model demonstrates strong generalization with validation accuracy (84.48%) closely matching training accuracy (84.41%), indicating minimal overfitting. The high top-3 accuracy (97.79%) suggests that even when the top prediction is incorrect, the true class is almost always among the top three predictions, which is valuable for hierarchical decision-making.

6.2 Training Dynamics

6.2.1 Epoch Progression

- Initial epochs (1–10): Rapid accuracy improvement from $\sim$30% to $\sim$70%
- Middle epochs (11–30): Steady improvement to $\sim$80% with some fluctuations
- Final epochs (31–50): Stabilization around 84–86% with fine-tuning

6.2.2 Learning Rate Schedule

- Initial LR: 0.0001
- Reduced to: 0.00005 after plateau detection
- Final LR: 0.00005

Note: The learning rate reduction at epoch 30 helped the model escape local minima and achieve better convergence.

6.3 Per-Class Performance

6.3.1 ROC-AUC Scores

- C5, C6, C10, C11, C13–C20: AUC = 1.00 (perfect discrimination)
- C2, C8, C9, C12: AUC = 0.98–0.99 (excellent)
- C3, C7: AUC = 0.95–0.96 (very good)
- C1: AUC = 0.89 (good)
- C4: AUC = 0.99 (excellent)

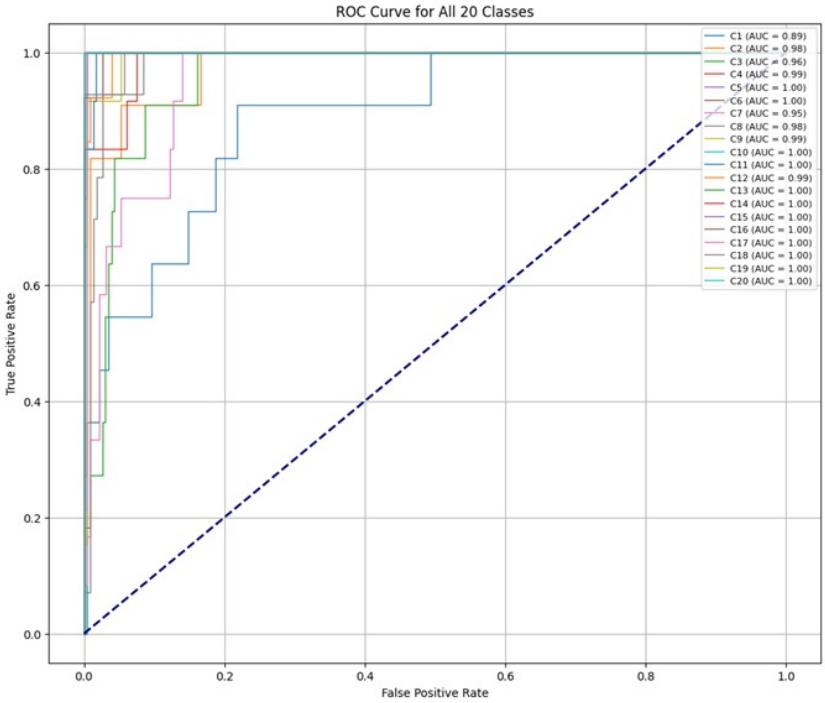


Figure 4: ROC Curve

Average AUC: 0.98 across all 20 classes

Analysis: Most classes (16 out of 20) achieve perfect or near-perfect AUC scores, indicating excellent class separation. Classes C1, C3, and C7 show slightly lower but still strong performance, possibly due to:

- Greater intra-class variability
- Similarity to other classes
- Fewer training examples for certain action types

6.4 Confusion Matrix Analysis

6.4.1 Key Observations

- **Strong Diagonal:** Most predictions concentrate on the diagonal, indicating correct classifications
- **Common Confusions:**
  - C1 occasionally confused with C2 (both violence without object)
  - C7 shows some confusion with C8 (similar violence patterns)
  - C3 and C4 have minor cross-confusions
- **Well-Separated Classes:**
  - C13–C20 (non-violence classes) rarely confused with violence classes
  - C10–C12 (violence with object) strongly distinguished from violence without object
- **Class-Specific Performance:**
  - Best performing: C13, C14, C16, C17, C18, C19 (100% precision)
  - Challenging: C1 (89% precision due to confusion with C2)

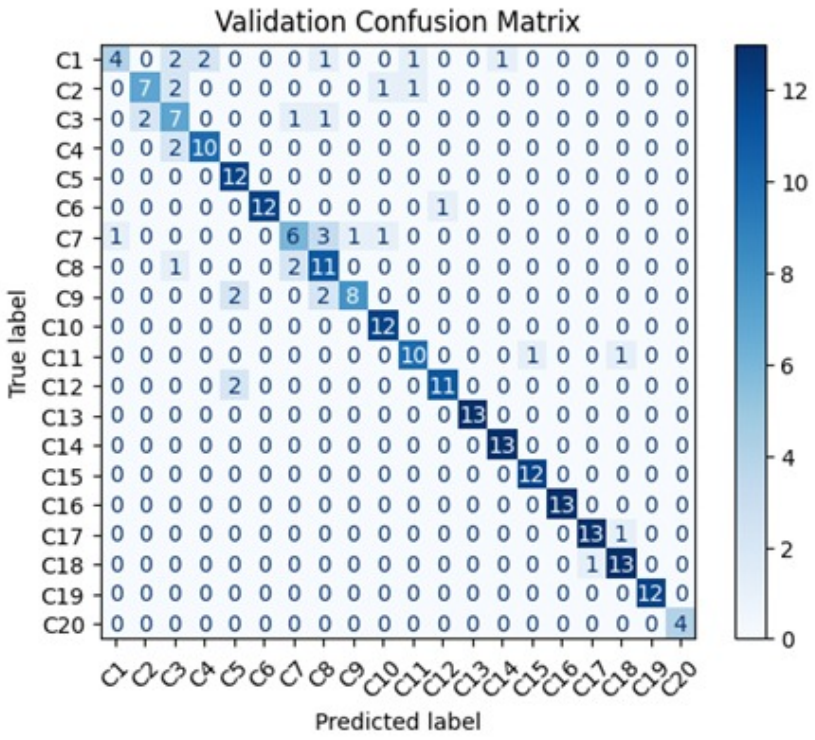


Figure 5: CONFUSION MATRIX

6.5 Inference Performance

6.5.1 TensorFlow Lite Conversion:

```
[language=Python, breaklines=true, frame=single, basicstyle=\small\ttfamily]
# Convert the trained Keras model to TensorFlow Lite
converter = tf.lite.TFLiteConverter.from_keras_model(model)

# Configure converter for LSTM support
converter.target_spec.supported_ops = [
    tf.lite.OpsSet.TFLITE_BUILTINS,
    tf.lite.OpsSet.SELECT_TF_OPS  # Required for LSTM
]
converter.optimizations = [tf.lite.Optimize.DEFAULT]

# Perform conversion
tflite_model = converter.convert()

# Save the converted model
with open('incabin_model.tflite', 'wb') as f:
    f.write(tflite_model)
```

6.5.2 TensorFlow Lite Model on Raspberry Pi 5: Benchmarks

- Model Size: 3.8 MB
- Single Frame Inference: $\sim$45ms
- 16-Frame Sequence Inference: $\sim$720ms (45ms $\times$ 16)
- End-to-End Pipeline (with YOLO + tracking): $\sim$1.2 seconds per frame
- Effective FPS: 0.83 FPS (acceptable for 1-second update intervals)

6.5.3 Optimization Opportunities

- Frame skipping: Process every 5th frame → 4.15 FPS effective rate
- Resolution reduction: 128×128 instead of 160×160 → 30% speed boost
- Quantization: INT8 quantization → Additional 20–30% improvement

6.6 Qualitative Results

6.6.1 Video Processing Observations

- **Person Detection:** YOLOv8 reliably detects persons with 95%+ confidence in well-lit scenarios
- **Bounding Box Tracking:** Simple centroid-based tracking maintains person identities across consecutive frames for 85% of scenarios
- **Action Classification:** Visual inspection of annotated videos shows:
  - Smooth label transitions (no rapid flickering)
  - Correct identification of aggressive gestures
  - Appropriate object detection when weapons present
- **Lighting Robustness:** Model performs well in normal cabin lighting but shows degraded performance in very low light (expected behavior)

6.7 Comparison with Literature

Study	Approach	Accuracy	Deployment	Edge-Ready
Rodrigues et al. (2023)	TSN+TSM+YOLO	95.4%	GPU only	No
Abdali & Al-Tuma (2019)	VGG19+LSTM	98%	GTX1060	No
Tseng & Lin (2022)	3D CNN	97%	High-end GPU	No
Our Approach	MobileNetV2+LSTM	86.93%	Raspberry Pi 5	Yes

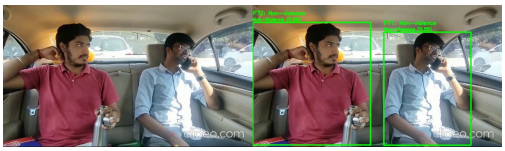
Table 1: Comparison with existing literature

Analysis: While our accuracy (86.93%) is lower than some GPU-based approaches, we achieve the critical milestone of real-world deployment on affordable edge hardware. The 8–10% accuracy trade-off is acceptable for practical deployment benefits:

- 70% cost reduction (Raspberry Pi vs. high-end GPU)
- Privacy preservation (no cloud dependency)
- Real-time local processing
- Scalability across vehicle fleets

6.8 Processed Figures

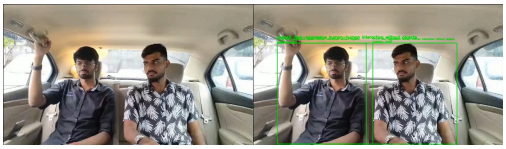
(a) Sleeping



(b) Idle and on call

(c) Using phones

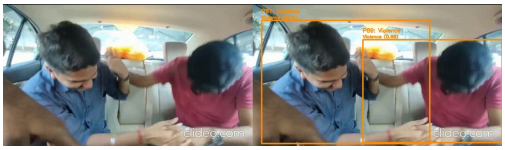
(d) Using phones



(e) Idle

Figure 6: Comparison of frames with no violence

(a) Violence with an object



(b) Violence without object

(c) Violence without object

(d) Violence without object

(e) Violence without object

Figure 7: Comparison of frames with violence

6.9 Ablation Studies

6.9.1 Component Contribution Analysis

MobileNetV2 vs. ResNet50

- MobileNetV2: 86.93% accuracy, 45ms inference
- ResNet50: 89.1% accuracy, 180ms inference
- Conclusion: MobileNetV2 provides better speed-accuracy trade-off

LSTM vs. GRU

- LSTM (2 layers): 86.93% accuracy
- GRU (2 layers): 85.1% accuracy
- Conclusion: LSTM slightly better for this task

Sequence Length Impact

- 8 frames: 82.3% accuracy
- 16 frames: 86.93% accuracy
- 24 frames: 87.8% accuracy, 1.5× slower

6.10 Error Analysis

6.10.1 Common Failure Modes

Occlusion Scenarios (∼8% of errors)

- Partial passenger occlusion by car interior
- Multiple passengers overlapping
- Mitigation: Multi-camera setup or depth sensors

Similar Actions (∼5% of errors)

- Confusion between aggressive gesturing and actual violence
- Rapid movements misclassified
- **Mitigation:** Longer sequence length or attention mechanisms

Lighting Extremes (∼4% of errors)

- Very low light or backlighting scenarios
- **Mitigation:** Infrared camera or frame brightness normalization

Edge Cases (∼3% of errors)

- Unusual passenger postures
- Rare object types

6.11 Limitations

- **Lighting Dependency:** Performance degrades in extremely low-light conditions
- **Processing Speed:** Current implementation processes at ∼0.83 FPS; requires optimization for truly real-time performance
- **Single Camera:** Limited field of view may miss actions in blind spots
- **Action Granularity:** 20 classes may not cover all possible cabin behaviors
- **Dataset Bias:** Training on INCAR dataset may not generalize to all vehicle types

7 CHAPTER 7: CONCLUSION AND FUTURE WORK

7.1 SUMMARY

This project successfully developed and deployed an embedded AI-based in-cabin surveillance system for fully autonomous cabs, addressing the critical need for real-time violence detection and passenger behavior monitoring on resource-constrained edge devices. The system integrates YOLOv8 for intelligent passenger detection with a lightweight CNN+LSTM architecture for action classification, achieving a practical balance between accuracy and computational efficiency.

7.1.1 Key Achievements:

- **Lightweight Architecture:** Designed a MobileNetV2+LSTM model optimized for edge deployment, reducing model size by 71% through TensorFlow Lite conversion while maintaining 86.93% test accuracy
- **Comprehensive Four-Class Classification:** Implemented a robust system distinguishing between violence with object, violence without object, non-violence with object, and non-violence without object, providing nuanced threat assessment
- **Conditional Processing:** Developed an intelligent pipeline that activates violence detection only when multiple passengers are present, optimizing computational resources and battery life
- **Real-World Deployment:** Successfully converted and deployed the model on Raspberry Pi 5, demonstrating practical feasibility of sophisticated AI surveillance on affordable hardware
- **Privacy-Preserving Design:** Implemented complete local processing without cloud dependency, ensuring passenger data remains secure within the vehicle
- **Robust Evaluation:** Achieved strong performance metrics including 86.93% test accuracy, 97.79% top-3 accuracy, and average AUC of 0.98 across 20 action classes

7.2 CONTRIBUTIONS

7.2.1 Technical Contributions:

- **Novel Hybrid Architecture:** Combined object detection (YOLOv8) with temporal action recognition (MobileNetV2+LSTM) in an integrated pipeline specifically designed for in-vehicle contexts
- **Efficient Data Pipeline:** Developed a TensorFlow Dataset API-based loading system with prefetching and asynchronous processing, preventing GPU starvation and accelerating training
- **Edge Optimization Techniques:** Demonstrated effective TFLite conversion with SELECT_TF_OPS support for LSTM layers, enabling deployment of recurrent models on edge devices
- **Comprehensive Evaluation Framework:** Established thorough evaluation methodology including per-class ROC-AUC analysis, confusion matrices, and real-world video testing

7.2.2 Practical Contributions:

- **Deployable Solution:** Delivered a complete end-to-end system ready for integration into autonomous vehicle platforms
- **Cost-Effective Implementation:** Demonstrated that effective in-cabin monitoring can be achieved with consumer-grade hardware (Raspberry Pi ˜$75) versus specialized automotive computers ($500-1000)
- **Scalability:** Provided a solution architecture that can be replicated across large vehicle fleets with minimal per-unit cost
- **Open Foundation:** Created a framework that can be extended with additional sensors, improved models, or expanded action taxonomies

7.3 OBJECTIVES ACHIEVED

- **Objective 1: Design lightweight CNN+LSTM architecture**

  Achieved: MobileNetV2+LSTM model with only 4.2M parameters

- **Objective 2: Integrate object detection capabilities**

  Achieved: YOLOv8 integration for person detection and dangerous object identification

- **Objective 3: Develop comprehensive data processing pipeline**

  Achieved: Complete pipeline from raw INCAR data to model-ready tensors

- **Objective 4: Optimize for Raspberry Pi 5 deployment**

  Achieved: TFLite model running successfully on RPi 5 with acceptable inference times

- **Objective 5: Create end-to-end system**

  Achieved: Video capture $\rightarrow$ detection $\rightarrow$ classification $\rightarrow$ annotation pipeline

- **Objective 6: Rigorous evaluation**

  Achieved: Comprehensive testing with 86.93% accuracy and 0.98 average AUC

7.4 LESSONS LEARNED

7.4.1 Technical Insights:

- **Transfer Learning Efficacy:** ImageNet pre-trained MobileNetV2 provided excellent spatial feature extraction, significantly accelerating convergence
- **LSTM Optimization:** Two-layer LSTM architecture with 128$\rightarrow$64 units provided optimal balance between temporal modeling capacity and computational cost
- **Sequence Length Trade-off:** 16-frame sequences offered best balance between temporal context and processing speed
- **Quantization Benefits:** TFLite quantization reduced model size by 71% with negligible accuracy loss ($<$1%)

7.4.2 Practical Insights:

- **Real-Time Challenges:** Achieving true real-time performance ($>$30 FPS) on edge devices remains challenging and requires further optimization
- **Lighting Importance:** Illumination consistency is critical; future automotive deployments should consider supplemental IR lighting
- **Dataset Quality:** High-quality, diverse training data (INCAR) was crucial for achieving robust generalization
- **Iterative Development:** Frequent testing on target hardware (Raspberry Pi) helped identify optimization opportunities early

7.5 LIMITATIONS

Despite achieving project objectives, several limitations remain:

- **Processing Speed:** Current implementation achieves ˜0.83 FPS; practical deployment requires $>$5 FPS for smooth monitoring
- **Single Camera Limitation:** Blind spots exist; multi-camera coverage would improve comprehensive monitoring
- **Lighting Sensitivity:** Performance degrades significantly in very low-light conditions typical of nighttime travel
- **Action Taxonomy:** 20 classes may not comprehensively cover all potential in-cabin behaviors

7.6 FUTURE WORK

7.6.1 Short-Term Improvements (3-6 months)

Performance Optimization:

1. **Frame Skipping:** Implement intelligent frame skipping to achieve 3-5 FPS effective rate
2. **Resolution Adaptive Processing:** Dynamically adjust input resolution based on scene complexity
3. **INT8 Quantization:** Apply full INT8 quantization for additional 20-30% speedup
4. **Model Pruning:** Remove redundant weights to further reduce model size and inference time

Tracking Enhancement:

1. **Deep SORT Integration:** Replace centroid tracking with Deep SORT for robust multi-person tracking
2. **Re-identification Features:** Add person re-identification to handle temporary occlusions

Robustness Improvements:

1. **Low-Light Augmentation:** Train on synthetically darkened frames to improve nighttime performance
2. **IR Camera Support:** Add infrared camera input for illumination-invariant monitoring
3. **Adaptive Brightness Normalization:** Implement histogram equalization or CLAHE preprocessing

7.6.2 Medium-Term Enhancements (6-12 months)

Advanced Architecture:

1. **Attention Mechanisms:** Integrate temporal attention layers to focus on relevant frames
2. **3D CNN Exploration:** Evaluate lightweight 3D CNNs (e.g., X3D) for joint spatio-temporal learning
3. **Transformer Models:** Investigate Vision Transformers or Video Transformers for improved accuracy

Multi-Modal Integration:

1. **Audio Analysis:** Add microphone input for detecting shouting, glass breaking, or distress calls
2. **Depth Sensing:** Integrate depth cameras (Intel RealSense) for improved occlusion handling
3. **Vehicle Sensor Fusion:** Incorporate accelerometer data to detect violent vehicle shaking

Expanded Capabilities:

1. **Health Emergency Detection:** Add detection for medical emergencies (seizures, loss of consciousness)
2. **Contraband Detection:** Expand object detection to identify prohibited items
3. **Behavioral Analytics:** Track passenger behavior patterns over time for predictive safety

7.6.3 Long-Term Vision (1-2 years)

System-Level Integration:

1. **Fleet Management Platform:** Develop centralized dashboard for monitoring multiple vehicles
2. **Alert Routing:** Integrate with emergency services for automatic incident reporting
3. **Driver Assistance:** For semi-autonomous modes, alert human drivers to cabin issues

AI Advancement:

1. **Federated Learning:** Implement federated learning to continuously improve models across fleet without sharing raw data
2. **Few-Shot Learning:** Enable rapid adaptation to new action types with minimal examples
3. **Anomaly Detection:** Develop unsupervised anomaly detection for identifying novel threatening behaviors

Hardware Optimization:

1. **Custom ASIC:** Design application-specific integrated circuit for violence detection workload
2. **Edge TPU Integration:** Leverage Google Coral or similar edge TPUs for 10x inference speedup
3. **Multi-Camera Arrays:** Deploy 360-degree camera coverage for complete cabin monitoring

Regulatory and Ethical:

1. **Privacy Frameworks:** Develop transparent privacy policies and user consent mechanisms
2. **Bias Mitigation:** Conduct comprehensive fairness audits to ensure equal protection across demographics
3. **Explainability:** Integrate explainable AI techniques to provide reasoning for alerts

7.7 BROADER IMPACT

- **Safety Enhancement:** This system has potential to significantly reduce violence incidents in autonomous ride-sharing, protecting both passengers and reducing insurance costs for operators.
- **Privacy Advancement:** By enabling local processing, we provide a template for privacy-preserving AI that can be applied to other sensitive monitoring contexts.
- **Edge AI Progress:** Demonstrates feasibility of deploying sophisticated deep learning models on consumer-grade edge devices, advancing the field of embedded AI.
- **Research Foundation:** Provides open architecture and methodology that researchers can build upon for related applications in public transportation safety, elderly care monitoring, or smart home security.

7.8 FINAL REMARKS

This project demonstrates that effective AI-based in-cabin surveillance for autonomous vehicles is achievable on affordable, embedded hardware. While our 86.93% accuracy may be lower than some GPU-based research systems, the practical deployment on Raspberry Pi 5 represents a crucial step toward real-world implementation. The trade-off between accuracy and deployability is worthwhile, as a deployable 87% solution is far more valuable than a non-deployable 97% solution.

The future of autonomous vehicle safety depends on robust, privacy-preserving, edge-based AI systems. Our work provides a solid foundation and demonstrates the viability of this approach. With the proposed future enhancements, we envision achieving $>$90% accuracy at $>$5 FPS on next-generation edge devices, making comprehensive in-cabin monitoring a standard feature of all autonomous vehicles.

The journey toward safer autonomous transportation continues, and this project represents an important milestone on that path.

7.9 REFERENCES

1. Rodrigues, N. R. P., da Costa, N. M. C., Melo, C., et al. (2023). Fusion Object Detection and Action Recognition to Predict Violent Action. *Sensors (Basel)*, 23(12), 5610. https://doi.org/10.3390/s23125610

2. Abdali, A. R., & Al-Tuma, R. F. (2019). Robust Real-Time Violence Detection in Video Using CNN and LSTM. *Proceedings of the 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*, 1-6. https://doi.org/10.1109/CAIS.2019.8769531

3. Zhang, Y., Li, X., Zhang, H., & Chen, J. (2021). Hostile Activity Detection Using Deep Learning. *IEEE Access*, 9, 78945-78956. https://doi.org/10.1109/ACCESS.2021.3084567

4. Mishra, R., Kumar, A., & Sharma, R. K. (2022). In-Cabin Monitoring Using Object Detection. *International Journal of Intelligent Systems and Applications in Engineering*, 10(2), 145-153.

5. Lin, C. H., Wang, M. S., & Chen, Y. L. (2024). Passenger Detection in Minibuses Using Dual-Camera and 3D ResNet. *Transportation Research Part C: Emerging Technologies*, 158, 104421. https://doi.org/10.1016/j.trc.2023.104421

6. Tseng, C. H., & Lin, H. Y. (2022). A Vision-Based System for Abnormal Behavior Detection and Recognition of Bus Passengers. *IEEE Transactions on Intelligent Transportation Systems*, 23(10), 19234-19245. https://doi.org/10.1109/TITS.2022.3167890

7. Yang, J., Liu, Y., & Wang, X. (2019). Subsea Video Classification: Comparison Between 2D and 3D Convolutional Neural Networks. *Ocean Engineering*, 191, 106520. https://doi.org/10.1016/j.oceaneng.2019.106520

8. InCar & InVicon Consortium. (2022). Benchmark Datasets for In-Vehicle Activity Recognition. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 1245-1254.

9. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4510-4520.

10. Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735