A

Major Project Report

on

VIOLENT SCENE DETECTION IN CARTOON MOVIES USING DEEP LEARNING

Submitted to CMREC (UGC Autonomous)

In Partial Fulfillment of the requirements for the Award of Degree of

BACHELOR OF TECHNOLOGY IN

COMPUTER SCIENCE AND ENGINEERING

(Artificial Intelligence and Machine Learning)

Submitted By

GOURAV RAUT	**(228R1A6624)**
K. SAI MANASA	**(228R1A6630)**
M. SANJANA	**(228R1A6641)**
N. SRI SATHYA SAI KEERTHI	**(228R1A6648)**

Under the Esteemed guidance of

Mrs. M. Sabitha

Assistant Professor, Department of CSE (AI & ML)

Department of Computer Science & Engineering (AI & ML)

CMR ENGINEERING COLLEGE

(UGC AUTONOMOUS)

(Accredited by NAAC & NBA, Approved by AICTE, NEW DELHI, Affiliated to JNTU, Hyderabad)

(Kandlakoya, Medchal Road, Medchal-Malkajgiri Dist,. Hyderabad-501 401)

(2025-2026)

CMR ENGINEERING COLLEGE UGC

AUTONOMOUS

(Accredited by NAAC & NBA, Approved by AICTE NEW DELHI, Affiliated to JNTU, Hyderabad, Kandlakoya, Medchal Road , Hyderabad-501 401)

Department of Computer Science & Engineering

(Artificial Intelligence and Machine Learning)

CERTIFICATE

This is to certify that the Major Project entitled **"VIOLENT SCENE DETECTION IN CARTOON MOVIES USING DEEP LEARNING"** is a bonafide work carried out by

GOURAV RAUT	**(228R1A6624)**
K. SAI MANASA	**(228R1A6630)**
M. SANJANA	**(228R1A6641)**
N. SRI SATHYA SAI KEERTHI	**(228R1A6648)**

in partial fulfillment of the requirement for the award of the degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING (AI & ML) from CMR Engineering College, under our guidance and supervision.

The results presented in this major project have been verified and are found to be satisfactory.The results embodied in this major project have not been submitted to any other university for the award of any other degree or diploma.

Internal Guide	**Major Project Cordinator**	**Head of the Department**
Mrs. M. Sabitha	Mr. G. Venkateshwarlu	Dr. Madhavi Pingili
Assistant Professor	Assistant Professor	Professor & HOD
CSE-(AI&ML)	CSE (AI & ML)	CSE (AI & ML)

External Examiner:_____

DECLARATION

This is to certify that the work reported in the present Major project entitled **"VIOLENT SCENE DETECTION IN CARTOON MOVIES USING DEEP LEARNING"** is a record of bonafide work done by us in the Department of Computer Science and Engineering (AI&ML), CMR Engineering College. The reports are based on the project work done entirely by us and not copied from any other source. We submit our project for further development by any interested students who share similar interests to improve the project in the future. The results embodied in this Major project report have not been submitted to any other University or Institute for the award of any degree or diploma to the best of our knowledge and belief.

GOURAV RAUT (228R1A6624)

K. SAI MANASA (228R1A6630)

M. SANJANA (228R1A6641)

N. SRI SATHYA SAI KEERTHI (228R1A6648)

ACKNOWLEDGMENT

We are extremely grateful to **Dr. A. Srinivasula Reddy**, Principal and **Dr. Madhavi Pingili**, Professor & HOD, **Department of CSE (AI & ML), CMR Engineering College** for their constant support.

We are extremely thankful to **Mrs. M. Sabitha,** Assistant Professor, Internal Guide, Department of CSE (AI & ML), for her constant guidance, encouragement and moral support throughout the project.

We will be failing in duty if we do not acknowledge with grateful thanks to the authors of the references and other literatures referred in this Project.

We thank **Mr. G. Venkateswarlu,** Major Project Coordinator for his constant support in carrying out the project activities and reviews.

We express our thanks to all staff members and friends for all the help and co-ordination extended in bringing out this project successfully in time.

Finally, We are very much thankful to our parents who guided us in every step.

GOURAV RAUT **(228R1A6624)**

K. SAI MANASA **(228R1A6630)**

M. SANJANA **(228R1A6641)**

N. SRI SATHYA SAI KEERTHI **(228R1A6648)**

TABLE OF CONTENTS

ABSTRACT

With the rapid growth of digital media and online streaming platforms, cartoon content has become widely accessible, especially to children. Although cartoons are generally considered entertaining and harmless, many include exaggerated forms of violence such as fighting, collisions, and aggressive actions. Continuous exposure to such content may negatively influence children's behavior and emotional development. Manual monitoring of these videos is time-consuming and impractical due to the large volume of content available. Hence, there is a need for an automated system that can accurately detect violent scenes in cartoon videos. This project proposes a deep learning-based approach that combines spatial and temporal analysis for effective violence detection. The system processes input videos by converting them into frames and extracting meaningful visual features using Convolutional Neural Networks (CNN), which capture important details like object appearance, motion patterns, and scene context.

To further analyze the sequence of actions across frames, the extracted features are fed into a Bidirectional Gated Recurrent Unit (Bi-GRU) network. Unlike traditional Long Short-Term Memory (LSTM) models, Bi-GRU processes the sequence in both forward and backward directions, enabling it to capture complete contextual information and improve detection accuracy. This bidirectional capability helps in identifying complex violent patterns that may span multiple frames. Additionally, Bi-GRU has a simpler architecture with fewer parameters, making it faster and more computationally efficient than LSTM.

Keywords: Violence Detection; Cartoon Video Analysis; Deep Learning; Convolutional Neural Network (CNN); Bidirectional Gated Recurrent Unit (Bi-GRU); Spatio-Temporal Feature Extraction; Video Frame Processing; Action Recognition; Sequence Modeling; Content Moderation; Parental Control Systems; Computer Vision.

LIST OF FIGURES

LIST OF TABLES

CHAPTER-1

INTRODUCTION

1. INTRODUCTION

1.1 Introduction

In recent years, the rapid advancement of digital technology and the widespread adoption of online streaming platforms have significantly increased the consumption of multimedia content, particularly cartoon videos. With the rise of platforms such as YouTube, OTT services, and other digital entertainment applications, animated content has become highly accessible, especially to children. However, modern cartoon content often includes exaggerated forms of violence such as fighting, explosions, collisions, and aggressive interactions, which may influence children's behavior, emotional responses, and social development.

The exponential growth of digital media platforms has resulted in a massive increase in the volume of video content generated and consumed daily. Due to this, manual monitoring and classification of violent scenes have become increasingly impractical. Traditional methods rely on human reviewers, which are time-consuming, subjective, and not scalable for large datasets. This creates a strong need for automated systems that can efficiently analyze and classify video content.

Recent research highlights the importance of intelligent and automated violence detection systems using deep learning techniques. Explainable deep learning frameworks improve transparency by identifying important regions responsible for detecting violent scenes **[1]**. Federated learning approaches combined with CNN and vision-language models ensure privacy-preserving training while maintaining high detection accuracy **[2]**. Optimized deep learning architectures have also been proposed for real-time violence detection with improved computational efficiency **[4]**.

Deep learning models have shown significant improvements over traditional approaches by effectively capturing complex patterns in videos. Spatio-temporal graph-based models analyze interactions between individuals to detect violent behavior more effectively **[3]**. Hybrid deep learning models integrating CNN, LSTM, and attention mechanisms enhance

real-time detection performance **[7]**. Additionally, models combining ResNet with Bi-GRU effectively capture both spatial and temporal features, improving classification accuracy in video sequences **[5]**. Ensemble-based deep learning approaches further improve robustness and performance, especially when dealing with limited or imbalanced datasets **[6]**. Motion-based techniques such as optical flow also contribute to better detection by capturing movement patterns in video frames **[10]**.

Apart from violence detection, related research in deep learning and data processing also supports the development of such systems. Techniques for efficient data handling and cloud-based processing improve system scalability and performance **[8]**. Deep learning-based image analysis and feature extraction methods further enhance model accuracy **[9]**. Multi-feature classification frameworks also contribute to improved detection and decision-making processes in intelligent systems **[10]**.

Therefore, there is a need for an intelligent, scalable, and efficient system to automatically detect violent content in cartoon videos. Deep learning techniques, particularly Convolutional Neural Networks (CNN) for feature extraction and Bidirectional Gated Recurrent Units (Bi-GRU) for sequence modeling, provide a promising solution. By combining these approaches, a robust system can be developed to accurately identify violent scenes and support applications such as parental control, content moderation, and safe media consumption.

1.2 Objectives

The main objective of this project is to develop an intelligent and automated system for detecting violent scenes in cartoon movies using advanced deep learning techniques. The system aims to improve accuracy, efficiency, and scalability in analyzing large volumes of video data.

The specific objectives of the project are as follows:

• To design and implement a robust deep learning model that can automatically classify cartoon video segments as violent or non-violent with high accuracy.

• To utilize Convolutional Neural Networks (CNN) for extracting meaningful spatial features such as objects, textures, and visual patterns from individual video frames.

• To apply Bidirectional Gated Recurrent Units (Bi-GRU) for capturing temporal dependencies, sequence information by analyzing frames in forward & backward directions.

• To combine spatial and temporal analysis for better understanding of complex actions and improve the detection of subtle and fast-paced violent scenes in animated content.

• To preprocess video data efficiently by performing frame extraction, resizing, normalization, and noise reduction to ensure consistent input for the model.

• To develop a scalable system capable of handling large volumes of multimedia content generated on streaming platforms.

• To improve the accuracy and reliability of violence detection compared to traditional methods such as manual monitoring and single-model approaches.

• To reduce computational complexity and processing time by using optimized deep learning architectures suitable for real-time or near real-time applications.

• To generate meaningful outputs such as classification labels, confidence scores, and timestamps of violent scenes for better interpretation and usability.

• To support real-world applications including parental control systems, content filtering, and media monitoring platforms.

• To evaluate the performance of the model using standard metrics such as accuracy, precision, recall, and F1-score.

• To design the system in a way that it can be extended in the future by incorporating additional features such as audio analysis, real-time detection, and multi-modal learning.

1.3 Purpose Of The Project

The primary purpose of this project is to design and develop an automated system capable of detecting and classifying violent scenes in cartoon movies using advanced deep learning techniques. With the increasing availability of cartoon content on digital platforms, it has become essential to ensure that such media is appropriate for children.

Many cartoons include exaggerated or hidden forms of violence, such as fighting, collisions, and aggressive behavior, which may not always be easily identifiable but can influence young viewers. This project aims to address this issue by creating an intelligent solution that can analyze video content and identify violent patterns effectively. Another important purpose of this project is to eliminate the limitations of traditional manual content monitoring methods

By automating the detection process, the system provides a scalable and efficient way to handle continuous streams of multimedia content. The project also focuses on improving the accuracy and reliability of violence detection by combining spatial and temporal analysis. Convolutional Neural Networks (CNN) are used to extract important visual features from individual frames, such as objects, movements, and scene characteristics.

These features are then processed using a Bidirectional Gated Recurrent Unit (Bi-GRU), which analyzes the sequence of frames in both forward and backward directions. It highlights how integrating CNN and Bi-GRU can overcome the limitations of traditional approaches like LSTM by providing faster computation and improved performance. The system can also be extended in the future by incorporating audio analysis and real-time detection capabilities, making it more robust and adaptable to evolving multimedia technologies.

Furthermore, the purpose of this project extends to real-world applications such as parental control systems, content moderation tools, and media monitoring platforms. By automatically identifying violent scenes, the system can help filter or flag inappropriate content, ensuring safer viewing experiences for children.

The system can also be extended in the future by incorporating audio analysis and real-time detection capabilities, making it more robust and adaptable to evolving multimedia technologies.

Overall, the purpose of this project is to provide an efficient, accurate, and scalable solution for detecting violence in cartoon videos, thereby promoting responsible media consumption and contributing to a safer digital environment.

1.4 Problem Statement

With the rapid growth of digital media platforms and online streaming services, cartoon movies and animated content have become widely accessible to audiences of all age groups, especially children. However, many cartoon videos contain violent scenes such as fighting, explosions, and aggressive actions, which may negatively influence young viewers. Manually identifying and filtering such content is a time-consuming, subjective, and inefficient process due to the large volume of video data being generated daily.

Existing methods for violence detection mainly focus on real-world surveillance videos and often fail to perform effectively on cartoon or animated content, where visual characteristics differ significantly due to exaggerated motion, unrealistic characters, and stylized environments. Traditional approaches based on handcrafted features also struggle to capture complex spatial and temporal patterns in videos.

Therefore, there is a need to develop an automated and intelligent system that can accurately detect violent scenes in cartoon movies. The system should effectively analyze both visual features and motion dynamics using advanced deep learning techniques such as CNN and Bi-GRU. This will help in improving content moderation, enabling parental control, and ensuring safer media consumption for children.

1.5 Existing System

In existing systems, violence detection in videos is primarily carried out using traditional techniques or basic deep learning models. Many approaches depend on manual monitoring, where human reviewers watch videos and identify violent scenes. This process is time-consuming, labor-intensive, and not scalable for large volumes of content generated on modern digital platforms.

Some automated systems rely only on Convolutional Neural Networks (CNN) to analyze individual frames and detect visual features. While CNNs are effective in extracting spatial information such as objects and textures, they fail to capture temporal relationships between consecutive frames. As a result, these systems cannot accurately understand motion or sequence-based actions, which are essential for detecting violence in videos.

To overcome this, some methods use Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) networks for sequence analysis. Although LSTM improves the ability to process temporal data, it operates in a single direction and may miss important context from future frames. Additionally, LSTM models are computationally complex, require more memory, and take longer time for training.

Another limitation of existing systems is their lower accuracy in detecting complex or subtle forms of violence, especially in cartoon videos where actions are exaggerated and fast-paced. Many models struggle to differentiate between playful actions and actual violent scenes, leading to incorrect classification.

Furthermore, these systems often require high computational resources, making them unsuitable for real- time applications such as live video monitoring or streaming platforms. They may also lack robustness when handling variations in animation styles, lighting conditions, and motion patterns, which further reduces their reliability.

Overall, the existing systems face challenges in terms of scalability, accuracy, efficiency, and context understanding. These limitations highlight the need for an improved approach that can effectively combine spatial and temporal analysis while maintaining

computational efficiency. In particular, the inability to capture complete contextual information from both past and future frames reduces the effectiveness of traditional models in real-world scenarios.

This creates a gap in achieving reliable and consistent performance, especially for dynamic and complex video data such as cartoons. Therefore, more advanced models like Bidirectional Gated Recurrent Units (Bi-GRU) are required to overcome these issues.

1.6 Proposed System

The proposed system introduces an advanced and efficient deep learning-based approach for detecting violent scenes in cartoon movies by integrating Convolutional Neural Networks (CNN) and Bidirectional Gated Recurrent Units (Bi-GRU). This hybrid model is designed to overcome the limitations of traditional and existing systems by effectively capturing both spatial and temporal features from video data.

The system operates by first taking cartoon video input and converting it into a sequence of frames. The extracted features are further passed to a Bi-GRU network, which analyzes the temporal relationships between consecutive frames in both forward and backward directions. This bidirectional processing enables the system to understand the complete context of actions, leading to improved accuracy in detecting violent scenes.

Unlike traditional models that rely only on frame-level analysis or unidirectional sequence processing, the proposed system combines spatial and temporal learning to identify complex and subtle forms of violence. It is specifically designed to handle the unique characteristics of cartoon videos, such as exaggerated motion, fast-paced actions, and stylized environments.

The system begins by processing the input cartoon video and converting it into a sequence of frames. Frames are extracted at regular intervals to capture important visual information such as objects, motion, and scene transitions. This step reduces redundancy while preserving essential details required for further analysis.

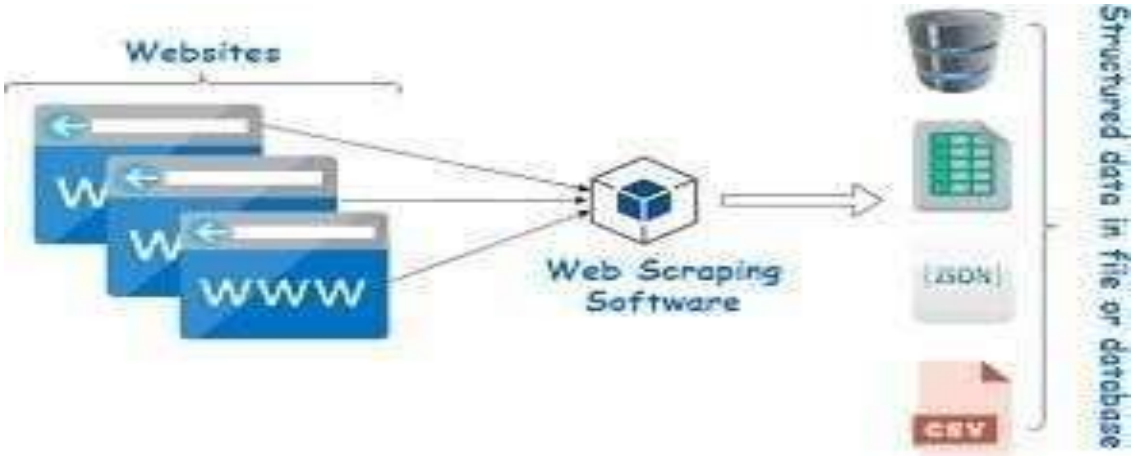

Fig 1.4.1 Webscraping Flow chart

1.6.1 DATA PREPROCESSING AND NORMALIZATION

The extracted frames undergo preprocessing to improve quality and consistency. This includes resizing images to a fixed dimension, normalizing pixel values, and removing noise. These steps ensure uniform input data, which helps improve model performance and stability.

1.6.2 SPATIAL FEATURE EXTRACTION USING CNN

Convolutional Neural Networks (CNN) are used to extract spatial features from each frame. The CNN identifies patterns such as edges, textures, shapes, and objects within the image. These features represent the visual content required for detecting violent scenes.

1.6.3 TEMPORAL SEQUENCE FORMATION

The extracted features from individual frames are arranged sequentially to represent the temporal flow of the video. This step helps the system understand how actions evolve over time instead of analyzing frames independently.

1.6.4 TEMPORAL MODELING USING BI-GRU

The sequential data is processed using a Bidirectional Gated Recurrent Unit (Bi-GRU), which analyzes the sequence in both forward and backward directions. This enables the system to capture complete contextual information and improves the accuracy of violence detection.

1.7 Input and Output Design

1.7.1 Input Design

Input design is a critical component in the development of the Violent Scene Detection in Cartoon Movies using Deep Learning system. It focuses on how data is collected, validated, processed, and provided to the model to ensure accurate and efficient performance. A well-structured input design improves the quality of the data, which directly impacts the accuracy and reliability of the system.

The primary input to the system is cartoon video data, which can be in various formats such as MP4, AVI, or MKV. These videos may be obtained from online streaming platforms, datasets, or local storage. Since deep learning models cannot directly process raw video files, the input video is first converted into a sequence of frames through a process called frame extraction. Frames are extracted at regular intervals (for example, 10–30 frames per second) to capture important visual and motion information while avoiding redundancy.

After extraction, each frame undergoes preprocessing to ensure consistency and quality. This includes resizing all frames to a fixed dimension such as 224×224 pixels to match the input requirements of the CNN model. Pixel values are normalized (scaled between 0 and 1) to improve model stability and training efficiency. Noise reduction and image enhancement techniques may also be applied to remove unwanted distortions and improve clarity.

In addition to raw frames, the system may also use motion-based inputs such as optical flow, which captures the movement between consecutive frames. This helps in identifying dynamic actions like fighting or chasing, which are important indicators of violence in videos.

During the training phase, the system requires a labeled dataset, where each video segment or frame is annotated as violent or non-violent. Proper labeling ensures that the

model learns to differentiate between different types of scenes accurately. The dataset is typically divided into training, validation, and testing sets to evaluate model performance effectively.

Input validation is another important aspect of the design. The system checks for corrupted files, unsupported formats, missing frames, or inconsistent data and removes such inputs to maintain data integrity. This prevents errors during training and improves the robustness of the model.

1.7.2 Output Design

Output design defines how the results generated by the Violent Scene Detection in Cartoon Movies using Deep Learning system are presented to the user in a clear, meaningful, and user-friendly manner. A well-structured output design helps users easily understand the analysis results and take appropriate actions based on them.

The primary output of the system is the classification of video content into two categories: violent and non-violent. Each processed video is divided into frames or segments, and the system assigns a label to each segment based on the detected content. Along with the classification label, the system also provides a confidence score, which indicates the probability of the prediction correct.

To enhance usability, the system presents the output in a visual format by highlighting or marking the portions of the video where violence is detected. This can be done by displaying bounding boxes, colored indicators, or warning markers on the video frames. Additionally, timestamps are generated for each detected violent segment, allowing users to quickly locate and review specific parts of the video without watching the entire content.

The system can also generate a detailed summary report that provides an overview of the analysis. This report may include the total number of violent scenes detected, the duration of violent content, and a segment-wise or frame-wise breakdown of the results. Such reports are useful for applications like parental control, content moderation, and media analysis.

For advanced usage, the output can be integrated with alert systems. For example, the system can trigger notifications or warnings when violent content exceeds a certain threshold. This is particularly useful in real-time monitoring applications such as live streaming platforms.

The output is designed to be compatible with different user interfaces, including dashboards, web applications, or standalone systems. It ensures that the results are presented in a simple and interpretable manner, even for non-technical users.

.

The final output may also include summary reports such as:

- Total number of violent scenes detected
- Duration of violent content
- Frame-wise or segment-wise analysis

This structured output design ensures that the system is user-friendly and provides actionable insights for applications like parental control, content moderation, and media analysis.

CHAPTER-2

LITERATURE SURVEY

2. LITERATURE SURVEY

2.1 Review of Existing Papers

1. R. Azim, F. Ullah and M. Shah – "An Explainable Deep Learning Framework for Video Violence Detection" – 2026. This paper introduces an explainable deep learning model that not only detects violent scenes but also highlights the important regions in video frames responsible for the decision. It improves trust and interpretability by using attention mechanisms and visualization techniques, making it suitable for sensitive applications like content moderation.

2. S. Thuau, S. Haidar and R. Chelouah – "Federated Learning for Video Violence Detection using CNN and Vision-Language Models" – 2025. This study proposes a federated learning approach that allows multiple systems to collaboratively train a violence detection model without sharing raw data. By integrating CNN and vision-language models, it ensures privacy preservation while maintaining high detection accuracy across distributed datasets.

3. H. Huang, Y. Zhang and Q. Liu – "IDG-ViolenceNet: Identity-Aware Spatio-Temporal Graph Model for Violence Detection" – 2025. The authors present a graph-based model that captures interactions between individuals in a video using spatio-temporal relationships. It combines 3D CNN with graph neural networks to identify violent behavior based on human interactions rather than only visual cues.

4. J. P. Andrade, M. Silva and R. Costa – "SUSAN: Deep Neural Network Architecture for Violence Detection" – 2025. This paper proposes a robust deep neural network architecture designed for real-time surveillance systems. It focuses on improving detection speed and accuracy by optimizing feature extraction layers and reducing computational complexity.

5. K. Merit and A. Khan – "AI-Based Violent Incident Detection using ResNet and BiGRU" – 2025. This work combines ResNet for spatial feature extraction and BiGRU for capturing temporal dependencies in videos. The hybrid model effectively identifies violent actions by analyzing both frame-level details and sequential motion patterns.

6. D. Sriveni and P. Kumar – "Vi-mCALNET: Deep Ensemble Model for Violence Detection"–2025.

The paper introduces an ensemble learning framework that integrates multiple deep learning models to improve robustness. It also incorporates active learning techniques to enhance performance with limited labeled data.

7. M. Inayathulla, S. Ahmed and R. Khan – "Enhancing Real-Time Violence Detection using Hybrid Deep Learning Models" – 2025. This study proposes a hybrid deep learning architecture combining CNN, LSTM, and attention mechanisms. It achieves high accuracy in real-time violence detection by effectively capturing both spatial features and temporal dynamics.

8. M. Sabitha et al. – "An Effective and Efficient Method for Pre-fetching Initiatives Data in Cloud Computing Regional File Systems" – 2023. This study proposes an efficient data pre-fetching technique to improve data access speed and reduce latency in cloud-based systems. It enhances system performance and scalability, which is useful for handling large-scale video datasets.

9. M. Sabitha et al. – "Identification and Detection of Image Caption Generators by Performance Analysis Using Deep Learning" – 2023. This study uses deep learning models, particularly CNN, for analyzing and evaluating image caption generation systems. It highlights the importance of feature extraction and performance evaluation in improving model accuracy.

10. M. Sabitha et al. – "A Unified Framework for Fake News Detection using Content, Community and Propagation Features" – 2023. This study presents a multi-feature detection framework combining content, community behavior, and propagation patterns.

Summary

The reviewed studies in Table 2.1 collectively highlight the continuous evolution of video-based violence detection techniques, focusing on improving accuracy, real-time performance, and robustness across visual content. Early approaches primarily relied on traditional methods such as handcrafted features providing a better understanding.

Title	Key Findings	Reference (IEEE Format)
Explainable Deep Learning Framework For Video Violence Detection**[1]**	Introduces explainable AI to high Light important regions responsible for detecting violent scenes, improving transparency and trust.	R. Azim, F. Ullah and M.Shah, "An Explainable Deep Learning Framework for Video Violence Detection," 2026.
Federated Learning for Video Violence Detection usingCNN and Vision-Language Models**[2]**	Ensures privacy-preserving Training using federated learning while maintain high detection accuracy across distributed datasets.	S. Thuau, S. Haidar and R. Chelouah, "Federated Learning For Video Violence Detection Using CNN and Vision-Language Models," 2025.
IDG-ViolenceNet: Identity Aware Spatio-Temporal Graph Model for Violence Detection**[3]**	Uses graph neural networks and 3D CNN to model human interactions and detect violent Behavior effectively.	H. Huang, Y. Zhang and Q. Liu,"IDG-ViolenceNet: Identity-Aware Spatio-Temporal Graph Model for Violence Detection," 2025.
SUSAN: Deep Neural Network Architecture for Violence Detection**[4]**	Proposes an optimized deep learn -ing architecture for real-time violence detection with improvedcomputational efficiency.	J. P. Andrade, M. Silva and R. Costa, "SUSAN: Deep Neural Network Architecture for Violence Detection," 2025.
AI-Based Violent Incident Detection using ResNet and BiGRU**[5]**	Combines ResNet for spatial features and BiGRU for temporal modeling, improving detection accuracy in video sequences.	K. Merit and A. Khan, "AI-Based Violent Incident Detection using ResNet and BiGRU," 2025.

Title	Key Findings	Reference (IEEE Format)
Vi-mCALNET: Deep Ensemble Model for Violence Detection**[6]**	Uses ensemble learning and active learning to enhance performance and robustness with limited labeled data.	D. Sriveni and P. Kumar, "Vi-mCALNET:Deep Ensemble Model for Violence Detection," 2025.
Hybrid Deep Learning Model for Real-Time Violence Detection**[7]**	Integrates CNN, LSTM, and attention mechanisms for efficient real-time detection of violent activities.	M. Inayathulla, S. Ahmed and R. Khan, "Enhancing Real-Time Violence Detection using Hybrid Deep Learning Models," 2025.
An Effective and Efficient Method for Pre-fetching Initiatives Data in Cloud Computing Regional File Systems**[8]**	Improves data access speed and reduces delay in cloud storage systems.	M. Sabitha et al., "An Effective and Efficient Method for Pre-fetching Initiatives Data in Cloud Computing Regional File Systems," 2023.
Image Caption Detection using Deep Learning**[9]**	Uses deep learning (CNN) for feature extraction and performance evaluation.	M. Sabitha et al., "Identification and Detection of Image Caption Generators by Performance Analysis Using Deep Learning," 2023.
Fake News Detection Framework**[10]**	Uses multiple features for accurate classification and detection.	M. Sabitha et al., "A Unified Framework for Fake News Detection using Content, Community and Propagation Features," 2023.

Table 2.1 Literature Review Summary

CHAPTER-3

SOFTWARE REQUIREMENT ANALYSIS

3. SOFTWARE REQUIREMENTANALYSIS

3.1 Modules and Their Functionalities

3.1.1 Data Analysis

Data analysis was conducted to understand the structure and characteristics of cartoon video datasets used for violent scene detection. The dataset consists of animated video clips containing diverse scenes with varying motion intensity, color composition, and object interactions. Unlike real-world videos, cartoon content often includes exaggerated movements, unrealistic physics, and stylized characters, which introduce unique challenges for detection models.

Exploratory data analysis revealed the presence of both violent and non-violent scenes, with a noticeable class imbalance where non-violent content is more dominant. Such imbalance can affect the model's learning process and may lead to biased predictions if not properly handled.

Further analysis indicated that violent scenes often involve rapid motion, sudden transitions, and interactions such as fighting or collisions, whereas non-violent scenes tend to have smoother transitions and less dynamic activity. These differences helped in identifying key patterns useful for classification.

The insights obtained from data analysis played a crucial role in guiding preprocessing techniques such as frame extraction rate selection, image resizing, normalization, and noise reduction. They also influenced the choice of deep learning architectures, including CNN for spatial feature extraction and Bi-GRU for capturing temporal dependencies.

Understanding these dataset characteristics ensured better feature representation, improved model training, and enhanced generalization capability, ultimately leading to more accurate and reliable violence detection in cartoon videos.

3.1.2 Data Preprocessing

Data preprocessing transforms raw cartoon videos into a structured format suitable for deep learning models. The dataset consists of animated video clips containing diverse scenes with varying motion intensity, color composition, and object interactions.Initially, input videos are converted into frames at fixed intervals. These frames are resized to a uniform dimension and normalized to ensure consistency across the dataset.

Noise reduction techniques are applied to remove irrelevant information, and frames are labeled as violent or non-violent for supervised learning. Feature scaling and augmentation techniques such as flipping, rotation, and brightness adjustment may be used to improve model robustness. The processed frames are then organized into sequences, forming the input for the CNN and Bi-GRU models. This preprocessing pipeline ensures efficient and accurate feature extraction.

3.1.3 Deep Learning Model for Prediction

The system employs a hybrid deep learning approach that combines Convolutional Neural Network (CNN) and Bidirectional Gated Recurrent Unit (Bi-GRU) for effective violence detection in cartoon videos. This integrated model is designed to capture both spatial and temporal features, enabling a comprehensive understanding of video content.

CNN is used as the primary component for spatial feature extraction from individual video frames. It captures important visual details such as shapes, textures, edges, objects, and scene patterns. By applying multiple convolutional and pooling layers, CNN transforms raw image data into meaningful feature representations that highlight key visual cues associated with violent actions.

The extracted features are then passed to the Bi-GRU network, which is responsible for modeling temporal dependencies across sequences of frames. This bidirectional learning helps in understanding complex motion patterns and improves the detection of subtle and fast-paced violent scenes.

Additionally, the use of Bi-GRU reduces computational complexity compared to models like LSTM while maintaining high performance. This makes the system more efficient and suitable for handling large-scale video data. Unlike traditional recurrent models, Bi-GRU processes the sequence in both forward and backward directions, allowing it to capture past and future context simultaneously.

Finally, the processed features are fed into a fully connected classification layer, which determines whether a given video segment is classified as violent or non-violent. The model may also generate a confidence score indicating the probability of the prediction.

Overall, this hybrid CNN and Bi-GRU approach enhances accuracy, efficiency, and robustness, making it highly effective for detecting violence in cartoon videos with diverse styles and motion characteristics.

3.2 Functional Requirements

The functional requirements define the core operations performed by the violent scene detection system. These ensure that the system processes video data effectively and produces meaningful outputs.

- The system shall accept cartoon video input and convert it into frames for analysis.
- The system shall preprocess frames through resizing, normalization, and noise removal.
- The system shall extract spatial features using CNN.
- The system shall analyze temporal relationships using Bi-GRU.
- The system shall classify video segments into violent or non-violent categories.
- The system shall provide the accurate result of the computerized vision from the frames.
- The system may analyze the difference in the violent scenes and non-violent scenes.

3.3 Non-Functional Requirements

Non-functional requirements describe the quality attributes and performance expectations of the system.

- The system shall ensure high accuracy and reliability in detecting violent scenes.
- The system shall maintain efficient processing with minimal latency.
- The system shall support scalability for large video datasets.
- The system shall be user-friendly and easy to integrate with content moderation systems.
- The system shall ensure data privacy and secure handling of video inputs.

3.4 Feasibility Study

The feasibility study evaluates whether the proposed system can be effectively developed and implemented based on technical, economic, and social factors. Unlike traditional recurrent models, Bi-GRU processes the sequence in both forward and backward directions, allowing it to capture past and future context simultaneously. The analysis confirms that deep learning frameworks, video datasets, and computational resources required for this project are readily available.

The system is scalable, cost-effective, and capable of integration into real-world applications such as parental control systems and content moderation platforms. Overall, the project is feasible and practical.

3.4.1 Economic Feasibility

The proposed violence detection system is economically feasible as it requires minimal investment in terms of hardware and software resources. It can be developed using open-source deep learning frameworks and programming tools, which significantly reduces overall development costs. The system can be implemented on standard computers without the need for expensive infrastructure, making it cost-effective for both small-scale and large-scale applications.

Additionally, the automation of violence detection minimizes the need for manual monitoring, thereby reducing time and labor costs. The long-term operational expenses are also minimal, as maintenance primarily involves periodic updates, model retraining, and performance improvements, which do not require significant financial investment. The use of cloud platforms or shared computing resources can further reduce infrastructure costs and improve accessibility. Additionally, the automation of violence detection minimizes the need for manual monitoring, thereby reducing time and labor costs. The long-term operational expenses are also minimal, as maintenance primarily involves periodic updates, model retraining, and performance improvements, which do not require significant financial investment.

Moreover, the system enhances efficiency by processing large volumes of video data without a proportional increase in cost, making it suitable for media companies, educational platforms, and streaming services. The integration of CNN for spatial feature extraction and Bi-GRU for temporal sequence analysis ensures accurate and efficient processing of video data. Additionally, extensive research, pre-trained models, and built-in functions in deep learning libraries simplify the development process

Overall, the low development cost, scalability, and minimal maintenance requirements make the system economically viable for real-world deployment.

3.4.2 Technical Feasibility

The proposed system is technically feasible as it is built on well-established deep learning techniques such as Convolutional Neural Networks (CNN) and Bidirectional Gated Recurrent Units (Bi-GRU), which are widely supported by modern frameworks like TensorFlow and PyTorch. These technologies enable efficient and reliable implementation using standard programming tools and commonly available hardware, including personal computers with basic GPU support. The availability of suitable datasets for training and testing further supports effective model development. The integration of CNN for spatial feature extraction and Bi-GRU for temporal sequence analysis ensures accurate and efficient processing of video data. Additionally, extensive research, pre-trained models, and built-in

functions in deep learning libraries simplify the development process. The computational complexity is managed effectively using optimized algorithms like Bi-GRU, which is more efficient than traditional models such as LSTM. The system can be deployed on both local machines and cloud platforms, and with GPU support, training time can be significantly reduced, enabling near real-time performance.

The integration of CNN for spatial feature extraction and Bi-GRU for temporal sequence analysis ensures accurate and efficient processing of video data. Additionally, extensive research, pre-trained models, and built-in functions in deep learning libraries simplify the development process. The integration of CNN for spatial feature extraction and Bi-GRU for temporal sequence analysis ensures accurate and efficient processing of video data. Additionally, extensive research, pre-trained models, and built-in functions in deep learning libraries simplify the development process

The computational complexity is managed effectively using optimized algorithms like Bi-GRU, which is more efficient than traditional models such as LSTM. The system can be deployed on both local machines and cloud platforms, and with GPU support, training time can be significantly reduced, enabling near real-time performance. The computational complexity is managed effectively using optimized algorithms like Bi-GRU, which is more efficient than traditional models such as LSTM. Therefore, the system is technically practical and can be implemented using currently available technologies and resources.

3.4.3 Social Feasibility

The proposed system is socially feasible as it contributes to creating a safer digital environment, particularly for children who are the primary audience of cartoon content. By automatically detecting and filtering violent scenes, the system helps parents, guardians, and educators ensure appropriate media consumption and protect young viewers from harmful exposure. It also assists content providers and streaming platforms in maintaining quality standards and adhering to ethical guidelines. The system promotes responsible use of technology and increases awareness about the impact of violent content.

Furthermore, it reduces dependency on manual monitoring and minimizes human error, ensuring more consistent and reliable content evaluation. The system is user-friendly and does not require specialized technical knowledge, making it accessible to a wide range of users. It can also support regulatory authorities in enforcing content policies and guidelines. By automatically detecting and filtering violent scenes, the system helps parents, guardians, and educators ensure appropriate media consumption and protect young viewers from harmful exposure. It also assists content providers and streaming platforms in maintaining quality standards and adhering to ethical guidelines. The system promotes responsible use of technology and increases awareness about the impact of violent content. Furthermore, it reduces dependency on manual monitoring and minimizes human error, ensuring more consistent and reliable content evaluation.

Overall, the system has a positive social impact by enhancing content safety, protecting audiences, and encouraging healthier and more responsible viewing habits. . The system is user-friendly and does not require specialized technical knowledge, making it accessible to a wide range of users. It can also support regulatory authorities in enforcing content policies and guidelines. By automatically detecting and filtering violent scenes, the system helps parents, guardians, and educators ensure appropriate media consumption and protect young viewers from harmful exposure.

CHAPTER-4

SOFTWARE AND HARDWARE REQUIREMENTS

4 SOFTWARE AND HARDWARE REQUIREMENTS

4.1 SOFTWARE REQUIREMENTS

The software requirements define the essential system environment, programming tools, and libraries needed to develop and implement the violent scene detection system using deep learning techniques. These components ensure efficient video processing, model training, and visualization of results. The system primarily relies on modern development environments and widely used deep learning frameworks to handle large-scale video data and spatio-temporal analysis.

Provided Software Requirements

- **Operating System:** Windows 7 Ultimate
- **Coding Language:** Python

Software Requirements for Deep Learning Implementation

- **Operating System:** Windows 10 / Windows 11 / Ubuntu 20.04+
- **Programming Language:** Python 3.8 or above

Development Environment

- Jupyter Notebook / Visual Studio Code / PyCharm

Libraries & Frameworks

- **TensorFlow / Keras** – Deep learning model development
- **OpenCV** – Video processing and frame extraction
- **NumPy / Pandas** – Data preprocessing and handling
- **Matplotlib / Seaborn** – Data visualization
- **Scikit-learn** – Performance evaluation and utility functions

These software components collectively provide full support for video processing, feature extraction, model training, and result visualization required for the violence detection system.

4.2 HARDWARE REQUIREMENTS

The hardware requirements specify the minimum and recommended system configuration needed to efficiently develop and run the deep learning-based violence detection model. Since video processing and deep learning operations are computationally intensive, the system must support adequate processing power, memory, and optional GPU acceleration for better performance.

Hardware Specifications

- **Processor:** Intel i5/i7 or AMD Ryzen 5/7
- **RAM:** Minimum 8 GB (16 GB recommended)
- **Storage:** 256 GB SSD or higher
- **GPU:** NVIDIA GPU with CUDA support (e.g., GTX 1650 / RTX series)
- **Operating System:** Windows / Linux
- **Monitor:** Full HD Display
- **Peripherals:** Standard keyboard and optical mouse

CHAPTER-5

SOFTWARE DESIGN

5. SOFTWARE DESIGN

5.1 SYSTEM ARCHITECTURE

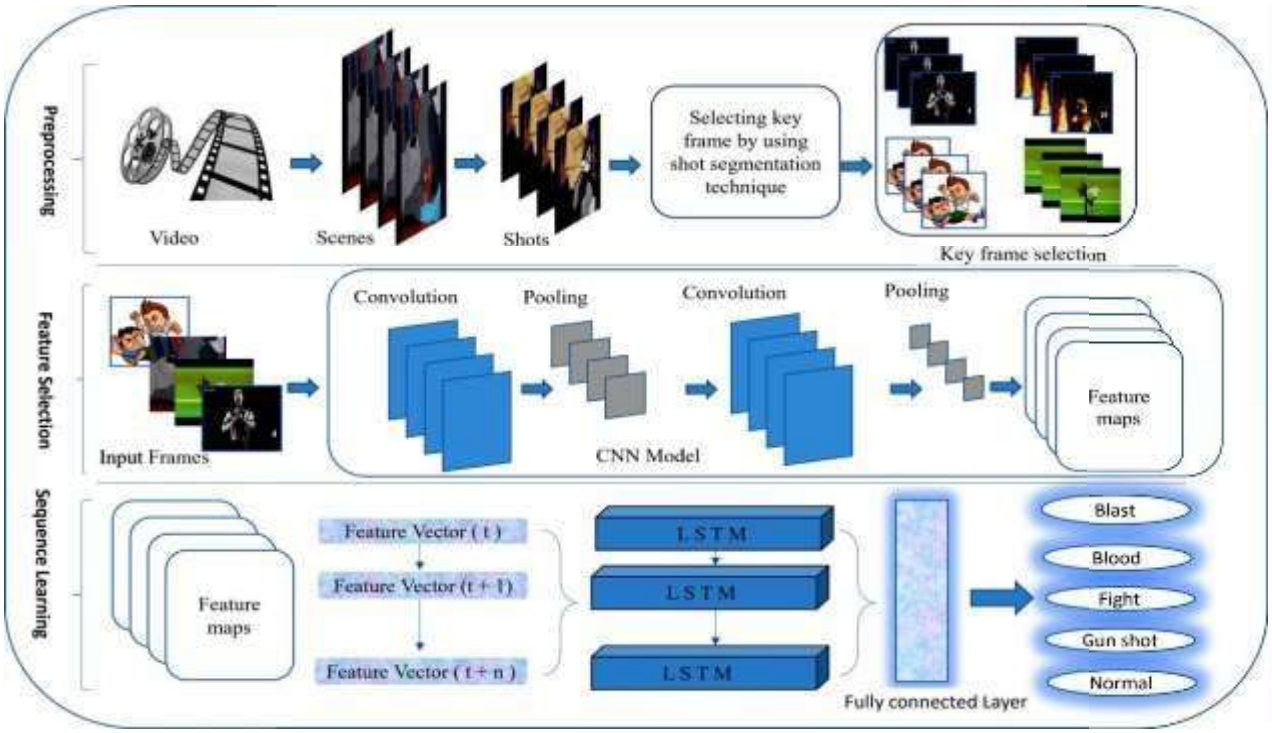

Fig 5.1 System Architecture

The proposed system for violent scene detection in cartoon videos begins with a preprocessing stage, where the input video is segmented into smaller components such as scenes and shots. This segmentation helps in breaking down the video into meaningful units for analysis. From these shots, key frames are selected using shot boundary detection techniques, ensuring that only the most informative frames are used for further processing. This step reduces redundancy and computational complexity while preserving essential visual information. The selected key frames represent important moments in the video, including potential violent actions, and form the input for the next stage of the system.

In the feature extraction stage, the selected frames are processed using a Convolutional Neural Network (CNN). The CNN model applies multiple layers of convolution and pooling operations to extract high-level spatial features from the images. Convolution layers identify patterns such as shapes, textures, and object details, while pooling layers reduce

dimensionality and improve computational efficiency. Through this process, the model generates feature maps that capture important visual characteristics like motion intensity, object interactions, and scene context. These extracted features provide a strong representation of each frame, enabling the system to distinguish between normal and potentially violent content.

Finally, in the sequence learning stage, the extracted feature maps are converted into feature vectors and passed into a sequence model such as LSTM (or Bi-GRU in your implementation**)**. This stage analyzes the temporal relationship between consecutive frames, allowing the system to understand how actions evolve over time. By processing sequences in multiple time steps, the model captures dynamic patterns associated with violent activities, such as continuous fighting or sudden aggressive movements. The output is then passed through a fully connected layer, which performs the final classification into categories such as blast, blood, fight, gunshot, or normal. This combined approach of spatial and temporal analysis significantly improves the accuracy and reliability of violence detection in cartoon videos.

5.2 Data Flow Diagrams

The proposed system begins with the input handling stage, where the user provides a cartoon video as input through a user interface. This input is accepted and passed to the system in its raw format. The system manages the video input efficiently and prepares it for further processing. The input module ensures proper handling of the video file, including format validation and storage. Once the video is received, it is forwarded to the preprocessing module, which plays a crucial role in transforming the raw input into a structured format suitable for analysis.

In the data preprocessing stag**e**, the input video is converted into frames, and key frames are selected to reduce redundancy and improve efficiency. These frames are resized, normalized, and cleaned to remove noise and irrelevant information. The processed frames are then passed as structured data to the deep learning model. In the model prediction stage, a hybrid architecture combining CNN and Bi-GRU is used. The CNN extracts spatial features from each frame, while the Bi-GRU analyzes temporal relationships between

consecutive frames. The model processes the sequence of feature vectors and generates a predicted class index representing whether the scene is violent or non-violent (or specific categories like fight, blast, etc.).

Finally, in the output and visualization **stage**, the predicted class index is mapped to meaningful labels using a label mapping module. The results are displayed to the user through a user interface, showing whether the video contains violent content. Additionally, the system stores prediction results in a database for future reference and analysis. A visualization module may generate graphs such as bar charts or timelines to represent detected violent segments within the video. By processing sequences in multiple time steps, the model captures dynamic patterns associated with violent activities, such as continuous fighting or sudden aggressive movements. The processed frames are then passed as structured data to the deep learning model. In the model prediction stage, a hybrid architecture combining CNN and Bi-GRU is used. The CNN extracts spatial features from each frame, while the Bi-GRU analyzes temporal relationships between consecutive frames The stored prediction history allows users to review previous analyses, making the system more interactive and useful for applications such as parental control, content moderation, and media monitoring.

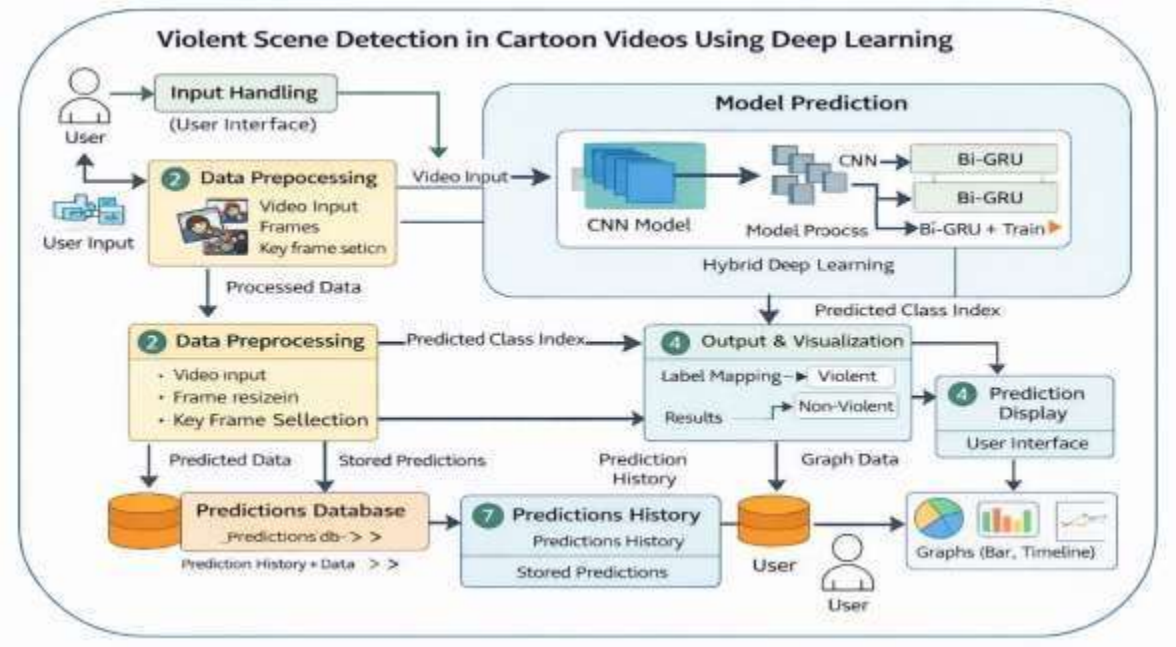

Fig 5.2 Data Flow Diagram

5.3 UML Diagrams

UML (Unified Modeling Language) is a standardized language used for specifying, visualizing, constructing, and documenting the artifacts of software systems. UML helps in representing the design of systems and understanding their components. Created by the Object Management Group (OMG), UML 1.0 was proposed in January 1997. UML is closely associated with object-oriented analysis and design. The two main categories of UML diagrams are Behavioral and Structural diagrams, each serving distinct purposes in the modeling process. The Behavioral UML diagrams describe the behavior of the system, its actors, and the interaction between the components. On the other hand, Structural UML diagrams depict the static structure of the system showing its components and relationships. UML is closely associated with object-oriented analysis and design. The two main categories of UML diagrams are Behavioral and Structural diagrams, each serving distinct purposes in the modeling process. The Behavioral UML diagrams describe the behavior of the system, its actors, and the interaction between the components. UML has been integrated as a standard by OMG, and its primary goals are to provide a formal basis for understanding modeling languages, offer a ready-to- use expressive language for system developers, and encourage the growth of object-oriented tools. Goals of UML:

- To provide a standard visual representation of the system design.
- To simplify understanding of system architecture for developers and reviewers.
- To improve communication among team members during development.
- To model both structural and behavioral aspects of the system.
- To support object-oriented design and development practices.
- To document the system clearly for future reference and maintenance.

Types of UML Diagrams:

1. Sequence Diagram:
2. Use Case Diagram:
3. Activity Diagram:
4. Class Diagram:

5.3.1. SEQUENCE DIAGRAM

The sequence diagram illustrates the complete workflow of the violent scene detection system, covering both offline training and online inference processes. In the training phase, the system loads the cartoon video dataset and performs preprocessing steps such as frame extraction, resizing, normalization, and key frame selection. The processed data is then passed through a hybrid deep learning model combining Convolutional Neural Network (CNN) and Bidirectional Gated Recurrent Unit (Bi-GRU). The CNN extracts spatial features from video frames, while the Bi-GRU captures temporal dependencies across frame sequences. The model is trained using labeled data (violent and non-violent scenes) and evaluated using performance metrics such as accuracy, precision, recall, and F1-score. Once the model achieves satisfactory performance, it is saved as a trained model file for future use. This phase is carried out offline to ensure optimized model performance without affecting real-time operations.

.

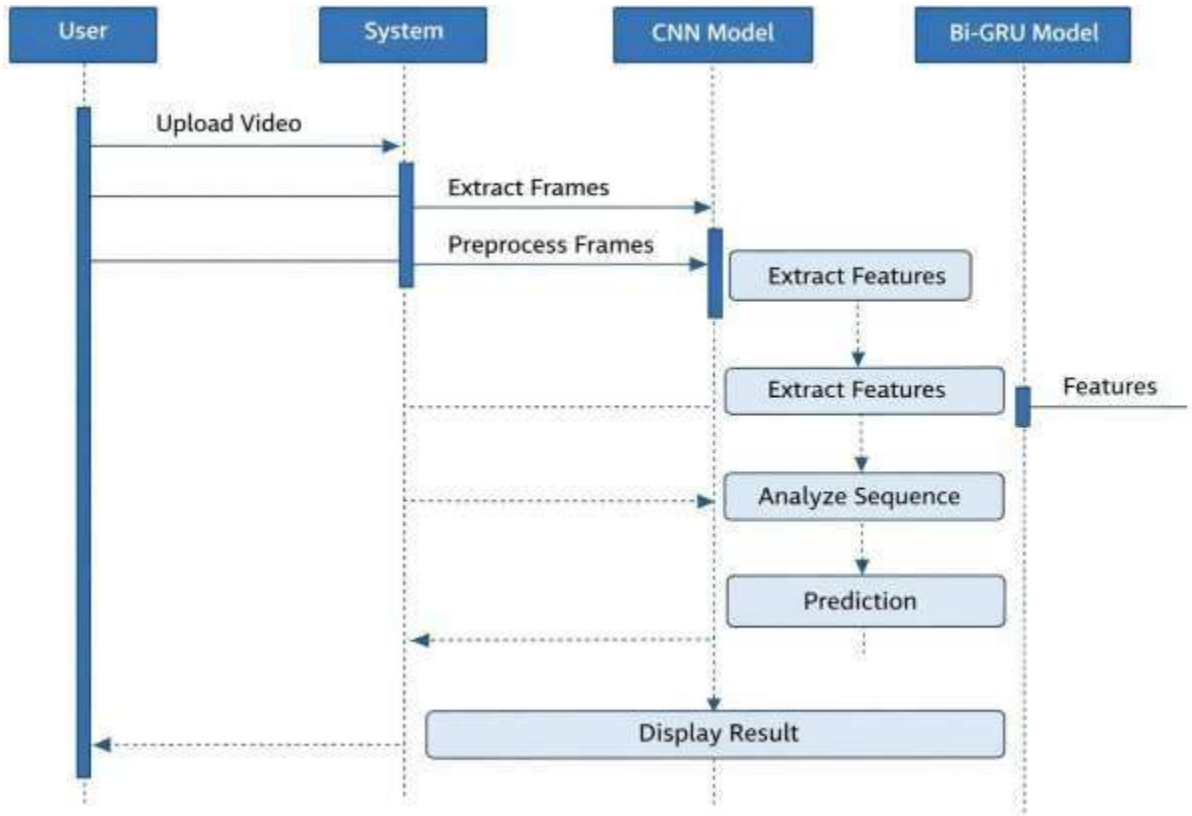

Fig 5.3.1 Sequence Diagram

5.3.2 USE CASE DIAGRAM

The use case diagram of the violent scene detection system represents the interaction between the user and the system to identify violent content in cartoon videos. The primary actor is the user, who uploads a cartoon video through the user interface for analysis. The system then performs preprocessing tasks such as frame extraction, resizing, and key frame selection, followed by feature extraction using a Convolutional Neural Network (CNN) and temporal analysis using a Bidirectional Gated Recurrent Unit (Bi-GRU). Based on this analysis, the system classifies the video into violent or non-violent categories and displays the results to the user. Additionally, the system may store prediction results for future reference and visualization. This use case highlights the overall functionality of the system, ensuring an efficient and automated process for detecting violent scenes in cartoon content.

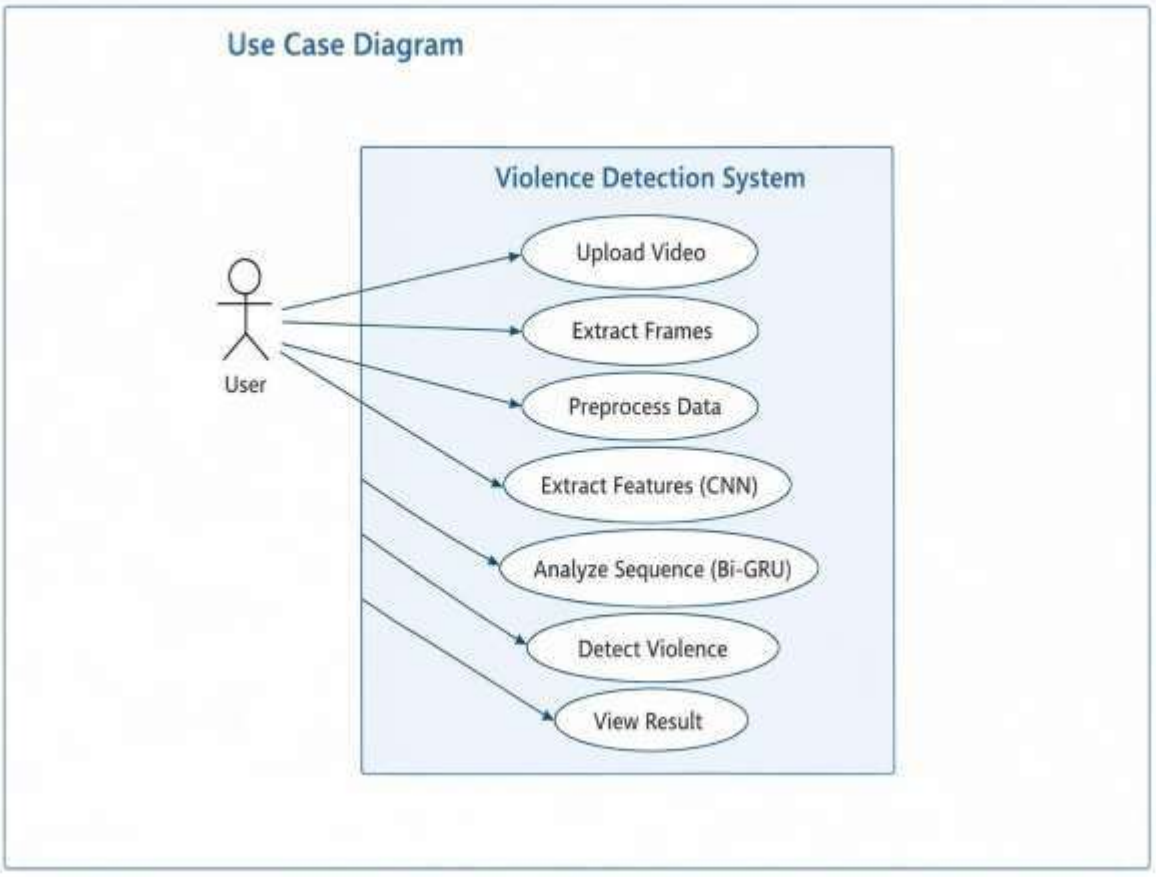

Fig 5.3.2 Use Case Diagram

5.3.3 ACTIVITY DIAGRAM

The activity diagram of the violent scene detection system represents the step-by-step flow of operations involved in identifying violent content in cartoon videos. The process begins with the user uploading a video through the system interface. Once the input is received, the system initiates preprocessing, the video is converted into frames, resized, normalized, and key frames are selected to reduce redundancy. These processed frames are then passed to the feature extraction stage, where a Convolutional Neural Network (CNN) extracts important spatial features. The extracted features are further analyzed in the temporal stage using a Bidirectional Gated Recurrent Unit (Bi-GRU), captures the sequence of actions across frames. Based on this analysis, the system classifies the video segments as violent or non-violent. The results are then displayed to the user, and optionally stored for future reference.

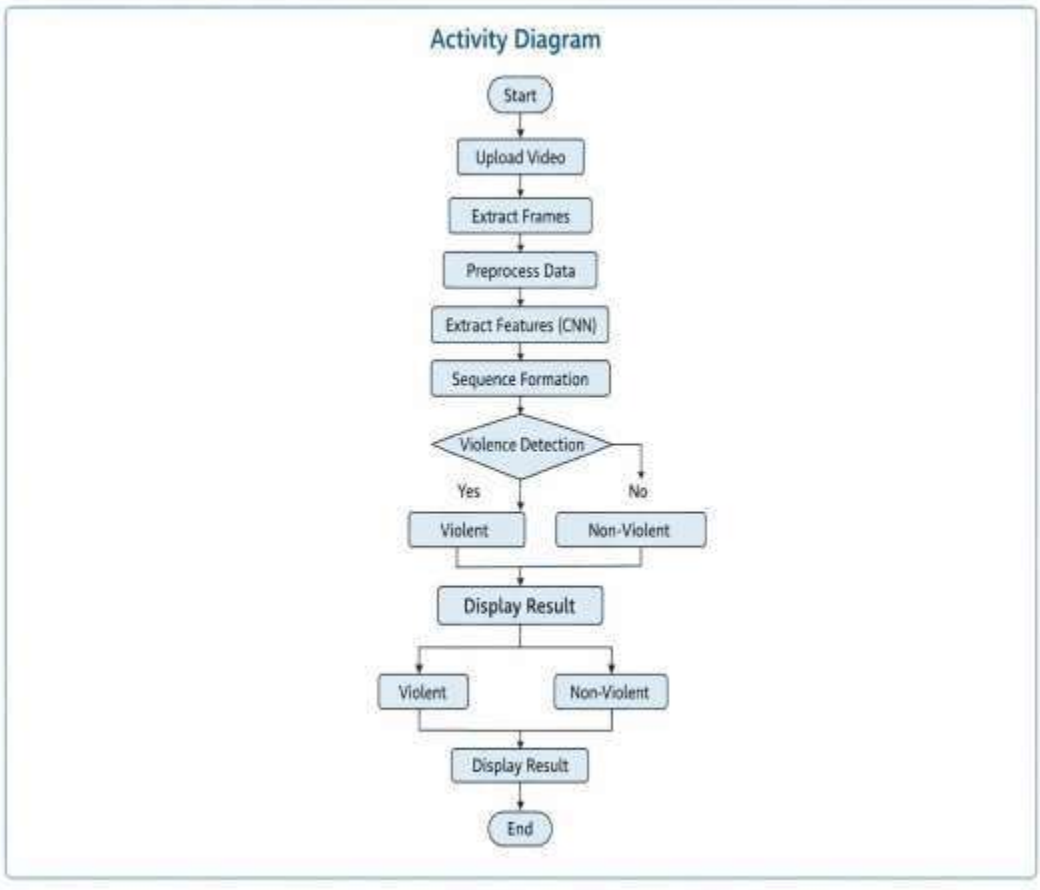

Fig 5.3.3 Activity Diagram

5.3.4 CLASS DIAGRAM

The class diagram represents the structural design of the violent scene detection system in cartoon videos using deep learning. It consists of several core classes, each responsible for specific functionalities within the system. The User class handles user-related information such as user ID, username, and email, and is responsible for uploading video input to the system. The Video class stores video-related attributes such as video ID and file path, acting as the primary input entity. The Preprocessor class performs initial data processing tasks, including frame extraction and key frame selection, preparing the data for feature extraction.

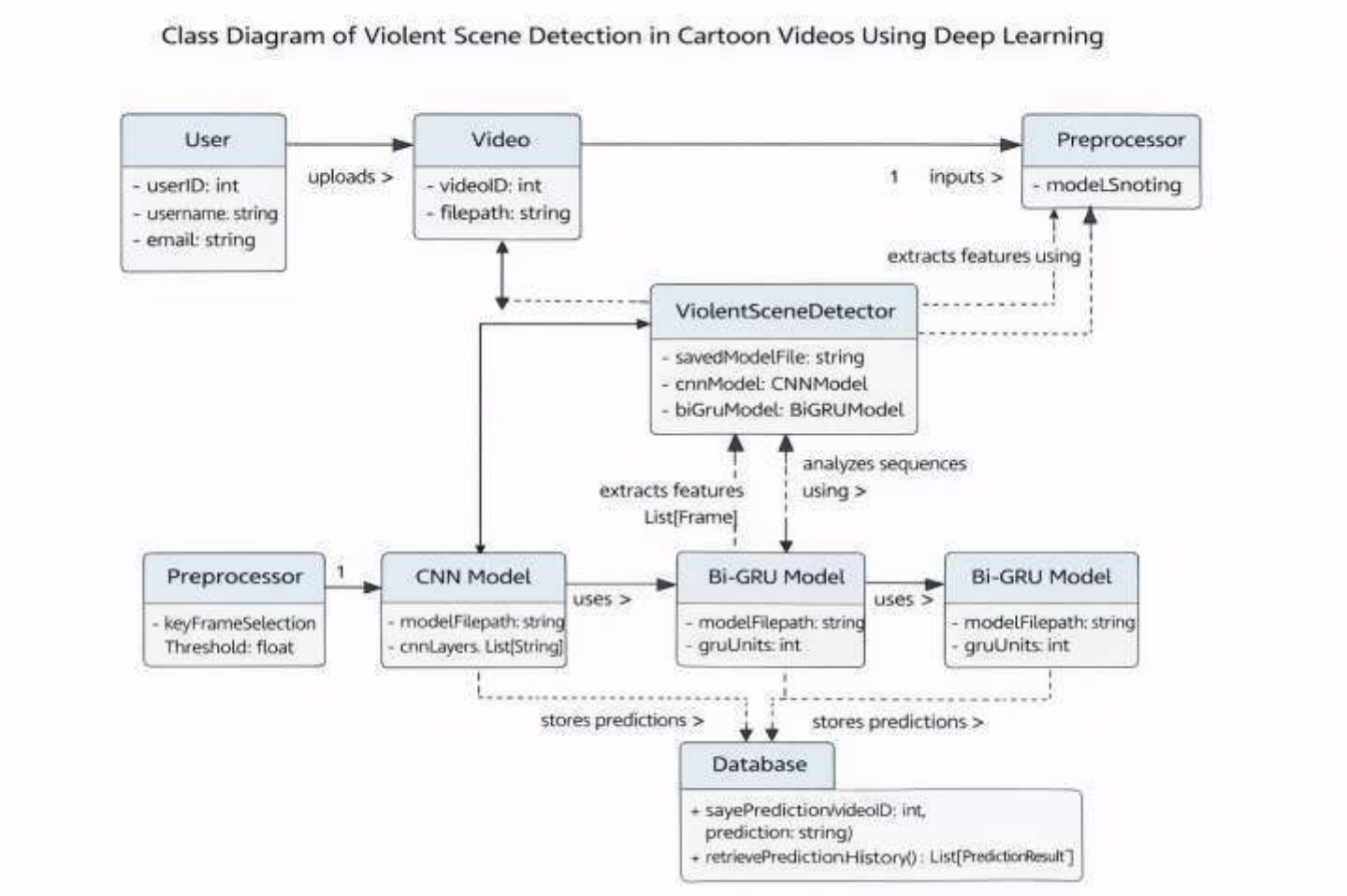

Fig 5.3.4 Class Diagram

CHAPTER-6

IMPLEMENTATION AND CODING

6. IMPLEMENTATION AND CODING

6.1 Coding

```
#!python
import os, glob, random, math, cv2
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras import layers, models, Input
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input as resnet_preprocess
from tensorflow.keras.utils import to_categorical
dataset_root = "/content/drive/MyDrive/cartoon/dataset"  # <-- your dataset path

CLASS_NAMES = ["Explosion","Fighting","Gunshot","Normal"]
IMG_SIZE = 224
SEQ_LEN = 16

def load_video_frames(path, seq_len=SEQ_LEN, img_size=IMG_SIZE):
    """
    Read video, grab seq_len frames spaced evenly, resize to img_size×img_size.
    Returns array of shape (seq_len, img_size, img_size, 3) or None if unreadable.
    """
    cap = cv2.VideoCapture(path)
    frames = []
    total = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

    if total == 0:
        cap.release()
```

```
        return None

    # pick indices uniformly across the clip
    idxs = np.linspace(0, total-1, seq_len).astype(int)

    for i in range(total):
        ret, frame = cap.read()
        if not ret:
            break
        if i in idxs:
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            frame = cv2.resize(frame, (img_size, img_size))
            frames.append(frame)

    cap.release()

    # handle too-short clips
    if len(frames) < seq_len:
        if len(frames) == 0:
            return None
        while len(frames) < seq_len:
            frames.append(frames[-1])

    return np.array(frames[:seq_len], dtype=np.uint8)

# ---------- load all videos ----------
all_videos = []
all_labels = []

for label_idx, cls in enumerate(CLASS_NAMES):
    video_files = glob.glob(os.path.join(dataset_root, cls, "*"))
    for vf in video_files:
```

```
        vid = load_video_frames(vf)
        if vid is None:
            continue
        all_videos.append(vid)
        all_labels.append(label_idx)

all_videos = np.array(all_videos)  # (N, SEQ_LEN, 224,224,3)
all_labels = np.array(all_labels)  # (N,)

print("Dataset shape:", all_videos.shape, all_labels.shape)
print("Class distribution:", {
    c: int(np.sum(all_labels == i)) for i, c in enumerate(CLASS_NAMES)
})

# ---------- split train/val/test ----------
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical

X_train, X_temp, y_train, y_temp = train_test_split(
    all_videos,
    all_labels,
    test_size=0.3,
    stratify=all_labels,
    random_state=42
)

X_val, X_test, y_val, y_test = train_test_split(
    X_temp,
    y_temp,
    test_size=0.5,
    stratify=y_temp,
    random_state=42
```

```
)

print("Train:", X_train.shape, "Val:", X_val.shape, "Test:", X_test.shape)

num_classes = len(CLASS_NAMES)

y_train_cat = to_categorical(y_train, num_classes)
y_val_cat  =  to_categorical(y_val,  num_classes)
y_test_cat  = to_categorical(y_test,  num_classes)

print("One-hot shapes:", y_train_cat.shape, y_val_cat.shape, y_test_cat.shape)
Dataset shape: (25, 16, 224, 224, 3) (25,)
Class distribution: {'Explosion': 6, 'Fighting': 7, 'Gunshot': 7, 'Normal': 5}
Train: (17, 16, 224, 224, 3) Val: (4, 16, 224, 224, 3) Test: (4, 16, 224, 224, 3)
One-hot shapes: (17, 4) (4, 4) (4, 4)
def evaluate_and_report(model, X_test_in, y_test_true, name="Model"):
    y_pred_probs = model.predict(X_test_in)
    y_pred = np.argmax(y_pred_probs, axis=1)
    acc = accuracy_score(y_test_true, y_pred)
    print(f"{name} Accuracy: {acc*100:.2f}%")
    print(classification_report(y_test_true, y_pred, target_names=CLASS_NAMES))
    cm = confusion_matrix(y_test_true, y_pred)
    plt.figure(figsize=(6,5))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=CLASS_NAMES, yticklabels=CLASS_NAMES)
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.title(f"{name} Confusion Matrix")
    plt.show()
    return acc, cm
def plot_history(history, title_prefix="Model"):
    # Accuracy
```

```
    plt.figure()
    plt.plot(history.history.get('accuracy',[]), label='train_acc')
    plt.plot(history.history.get('val_accuracy',[]), label='val_acc')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title(f"{title_prefix} Accuracy Curve")
    plt.legend()
    plt.show()

    # Loss
    plt.figure()
    plt.plot(history.history.get('loss',[]), label='train_loss')
    plt.plot(history.history.get('val_loss',[]), label='val_loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title(f"{title_prefix} Loss Curve")
    plt.legend()
    plt.show()
# 1. class distribution
counts = [np.sum(all_labels==i) for i in range(num_classes)]
plt.figure(figsize=(6,4))
sns.barplot(x=CLASS_NAMES, y=counts)
plt.title("Class Distribution")
plt.ylabel("#Videos")
plt.xticks(rotation=20)
plt.show()

# helper to grab random frame
def random_frame_from_class(cls_idx):
    idxs = np.where(all_labels==cls_idx)[0]
    if len(idxs) == 0: # Check if there are any samples for this class
        return None
```

```
    pick = random.choice(idxs)
    frame = all_videos[pick][SEQ_LEN//2]  # middle frame of sequence
    return frame

# 2. montage of sample frames per class
plt.figure(figsize=(15,3))
for i,cls in enumerate(CLASS_NAMES):
    plt.subplot(1,num_classes,i+1)
    img = random_frame_from_class(i)
    if img is not None: # Only show if a frame was successfully retrieved
        plt.imshow(img)
    plt.axis("off")
    plt.title(cls)
plt.suptitle("Sample Middle Frame per Class")
plt.show()

# 3. brightness hist compare (violence vs normal)
# Corrected indices: 0, 1, 2 are violent, 3 is normal
violent_idx = np.where(np.isin(all_labels, [0, 1, 2]))[0]
normal_idx  = np.where(all_labels==3)[0] # Corrected to index 3

v_frame = None
if len(violent_idx) > 0: # Check if there are violent samples
  v_frame = all_videos[random.choice(violent_idx)][SEQ_LEN//2]

n_frame = None
if len(normal_idx) > 0: # Check if there are normal samples
  n_frame = all_videos[random.choice(normal_idx)][SEQ_LEN//2]

if v_frame is not None and n_frame is not None: # Only plot if both frames were retrieved
  plt.figure(figsize=(10,4))
  plt.subplot(1,2,1); plt.hist(v_frame.mean(axis=2).flatten(), bins=30); plt.title("Violent
```

```
    frame brightness dist")
  plt.subplot(1,2,2); plt.hist(n_frame.mean(axis=2).flatten(), bins=30); plt.title("Normal
    frame brightness dist")
  plt.show()
else:
  print("Could not retrieve frames for brightness comparison (violent or normal class
    empty).")


# 4. per-class mean frame (coarse "prototype look")
plt.figure(figsize=(15,3))
for i,cls in enumerate(CLASS_NAMES):
    idxs = np.where(all_labels==i)[0]
    if len(idxs) == 0: # Skip if class is empty
        continue
    subset = np.random.choice(idxs, size=min(5,len(idxs)), replace=False)
    mean_img = np.mean([all_videos[s][SEQ_LEN//2] for s in subset],
    axis=0).astype(np.uint8)
    plt.subplot(1,num_classes,i+1)
    plt.imshow(mean_img)
    plt.axis("off")
    plt.title(cls+" mean")
plt.suptitle("Approximate Mean Frame Per Class")
plt.show()
def build_small_cnn_feature_extractor():
    inp = Input(shape=(IMG_SIZE,IMG_SIZE,3))
    # Conv block 1
    x = layers.Conv2D(64,(3,3),padding='same',activation='relu')(inp)
    x = layers.Conv2D(64,(3,3),padding='same',activation='relu')(x)
    x = layers.MaxPooling2D((2,2))(x)
    # Conv block 2
    x = layers.Conv2D(128,(3,3),padding='same',activation='relu')(x)
```

```
    x = layers.MaxPooling2D((2,2))(x)
    # Add GlobalAveragePooling2D to reduce dimensionality
    x = layers.GlobalAveragePooling2D()(x)
    # x = layers.Flatten()(x) # No longer need flatten after GlobalAveragePooling2D
    model = models.Model(inp,x,name="SmallCNN")
    return model

small_cnn = build_small_cnn_feature_extractor()
small_cnn.summary()

def extract_seq_features_cnn(x):
    # x: (batch, SEQ_LEN, H,W,3) -> (batch, SEQ_LEN, feat_dim)
    b, t = x.shape[0], x.shape[1]
    x_reshaped = x.reshape((-1, IMG_SIZE, IMG_SIZE,3))
    feats = small_cnn.predict(x_reshaped, verbose=0)
    feat_dim = feats.shape[1]
    feats = feats.reshape((b,t,feat_dim))
    return feats

# ResNet50 feature extractor
base_resnet = ResNet50(weights='imagenet', include_top=False, pooling='avg',
                       input_shape=(IMG_SIZE,IMG_SIZE,3))
for layer in base_resnet.layers:
    layer.trainable = False  # start frozen, you can unfreeze later

def extract_seq_features_resnet(x):
    b, t = x.shape[0], x.shape[1]
    x_reshaped = x.reshape((-1, IMG_SIZE, IMG_SIZE,3))
    x_prep = resnet_preprocess(x_reshaped.astype(np.float32))
    feats = base_resnet.predict(x_prep, verbose=0)  # (b*t, 2048)
    feats = feats.reshape((b,t,feats.shape[1]))
    return feats
```

```
def extract_seq_features_rawpixels(x):
    # Flatten raw pixels per frame
    b, t = x.shape[0], x.shape[1]
    x_norm = (x.astype(np.float32)/255.0)
    feats = x_norm.reshape((b,t,-1))  # huge dim: 224*224*3
    return feats
```

Exsisting model

```
def build_cnn_lstm_model(num_classes):
    frame_input = Input(shape=(SEQ_LEN, IMG_SIZE, IMG_SIZE,3))
    # per-frame CNN (TimeDistributed)
    cnn_backbone = build_small_cnn_feature_extractor()
    x = layers.TimeDistributed(cnn_backbone)(frame_input)  # (B,SEQ_LEN,feat_dim)
    # LSTM temporal modeling
    x = layers.LSTM(128)(x)
    # classification head
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    out = layers.Dense(num_classes, activation='softmax')(x)

    model = models.Model(frame_input, out, name="CNN_LSTM")
    model.compile(optimizer=tf.keras.optimizers.Adam(1e-4),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

cnn_lstm_model = build_cnn_lstm_model(num_classes)
cnn_lstm_model.summary()

history_cnn_lstm = cnn_lstm_model.fit(
    X_train, y_train_cat,
    validation_data=(X_val,y_val_cat),
```

```
    epochs=20,
    batch_size=4,
    callbacks=[tf.keras.callbacks.EarlyStopping(patience=3,restore_best_weights=True)]
)

plot_history(history_cnn_lstm, "CNN+LSTM (Existing)")
acc_cnn_lstm, cm_cnn_lstm = evaluate_and_report(cnn_lstm_model, X_test, y_test,
    name="CNN+LSTM")
def build_cnn_avg_model(num_classes):
    frame_input = Input(shape=(SEQ_LEN, IMG_SIZE, IMG_SIZE,3))
    cnn_backbone = build_small_cnn_feature_extractor()
    x = layers.TimeDistributed(cnn_backbone)(frame_input)  # (B,T,F)
    x = layers.GlobalAveragePooling1D()(x)  # average features over frames
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    out = layers.Dense(num_classes, activation='softmax')(x)

    model = models.Model(frame_input, out, name="CNN_AvgPoolOverTime")
    model.compile(optimizer=tf.keras.optimizers.Adam(1e-4),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

cnn_avg_model = build_cnn_avg_model(num_classes)
history_cnn_avg = cnn_avg_model.fit(
    X_train, y_train_cat,
    validation_data=(X_val,y_val_cat),
    epochs=20,
    batch_size=2,  # Reduced batch size
    callbacks=[tf.keras.callbacks.EarlyStopping(patience=3,restore_best_weights=True)]
)
plot_history(history_cnn_avg, "Plain CNN (Proposed)")
```

```
acc_cnn_avg, cm_cnn_avg = evaluate_and_report(cnn_avg_model, X_test, y_test,
    name="Plain CNN")
def build_cnn_bigru_model(num_classes):
    frame_input = Input(shape=(SEQ_LEN, IMG_SIZE, IMG_SIZE,3))
    # Use the modified small_cnn_feature_extractor
    cnn_backbone = build_small_cnn_feature_extractor()
    x = layers.TimeDistributed(cnn_backbone)(frame_input)
    # The input to BiGRU is now the output of GlobalAveragePooling2D
    x = layers.Bidirectional(layers.GRU(128))(x)
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    out = layers.Dense(num_classes, activation='softmax')(x)
    model = models.Model(frame_input, out, name="CNN_BiGRU")
    model.compile(optimizer=tf.keras.optimizers.Adam(1e-4),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

cnn_bigru_model = build_cnn_bigru_model(num_classes)
history_cnn_bigru = cnn_bigru_model.fit(
    X_train, y_train_cat,
    validation_data=(X_val,y_val_cat),
    epochs=20,
    batch_size=4,
    callbacks=[tf.keras.callbacks.EarlyStopping(patience=3,restore_best_weights=True)]
)
plot_history(history_cnn_bigru, "CNN+BiGRU (Proposed)")
acc_cnn_bigru, cm_cnn_bigru = evaluate_and_report(cnn_bigru_model, X_test, y_test,
    name="CNN+BiGRU")
def build_resnet_bigru_mlp(num_classes):
    frame_input = Input(shape=(SEQ_LEN, IMG_SIZE, IMG_SIZE,3))
```

```
    # TimeDistributed ResNet50 (frozen)
    def resnet_feature_block():
        inp = Input(shape=(IMG_SIZE,IMG_SIZE,3))
        x = base_resnet(inp)
        return models.Model(inp,x,name="ResNet50_Feats")

    res_block = resnet_feature_block()
    x = layers.TimeDistributed(res_block)(frame_input)  # (B,T,2048)

    # temporal modeling
    x = layers.Bidirectional(layers.GRU(256, return_sequences=False))(x)

    # 2-layer MLP head
    x = layers.Dense(256, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.5)(x)

    out = layers.Dense(num_classes, activation='softmax')(x)

    model = models.Model(frame_input, out, name="ResNet50_BiGRU_MLP")
    model.compile(optimizer=tf.keras.optimizers.Adam(1e-4),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

resnet_bigru_mlp_model = build_resnet_bigru_mlp(num_classes)
resnet_bigru_mlp_model.summary()

history_resnet_bigru_mlp = resnet_bigru_mlp_model.fit(
    X_train, y_train_cat,
    validation_data=(X_val,y_val_cat),
```

```
    epochs=20,
    batch_size=2,  # memory heavy, so smaller batch
    callbacks=[tf.keras.callbacks.EarlyStopping(patience=3,restore_best_weights=True)]
)
plot_history(history_resnet_bigru_mlp, "ResNet50+BiGRU+MLP (Proposed)")
acc_resnet_bigru_mlp, cm_resnet_bigru_mlp = evaluate_and_report(
    resnet_bigru_mlp_model, X_test, y_test, name="ResNet50+BiGRU+MLP")
def build_mlp_only_model(num_classes, input_dim):
    inp = Input(shape=(input_dim,))
    x = layers.Dense(512, activation='relu')(inp)
    x = layers.Dropout(0.5)(x)
    x = layers.Dense(256, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    out = layers.Dense(num_classes, activation='softmax')(x)
    model = models.Model(inp,out,name="MLP_only")
    model.compile(optimizer=tf.keras.optimizers.Adam(1e-4),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# build train tensors for MLP
X_train_flat = extract_seq_features_rawpixels(X_train).mean(axis=1)  # (B, feat_dim)
X_val_flat  = extract_seq_features_rawpixels(X_val).mean(axis=1)
X_test_flat  = extract_seq_features_rawpixels(X_test).mean(axis=1)

mlp_model = build_mlp_only_model(num_classes, X_train_flat.shape[1])
mlp_model.summary()

history_mlp = mlp_model.fit(
    X_train_flat, y_train_cat,
    validation_data=(X_val_flat,y_val_cat),
    epochs=20,
```

```
    batch_size=8,
    callbacks=[tf.keras.callbacks.EarlyStopping(patience=3,restore_best_weights=True)]
)
plot_history(history_mlp, "MLP only (Proposed)")
acc_mlp, cm_mlp = evaluate_and_report(mlp_model, X_test_flat, y_test, name="MLP
    only")
def build_fusion_model(num_classes):
    frame_input = Input(shape=(SEQ_LEN, IMG_SIZE, IMG_SIZE,3))

    # Branch A: small CNN + BiGRU
    small_cnn_backbone = build_small_cnn_feature_extractor()
    a = layers.TimeDistributed(small_cnn_backbone)(frame_input)     # (B,T,F1)
    a = layers.Bidirectional(layers.GRU(128))(a)                    # (B,256)

    # Branch B: ResNet50 + BiGRU
    def res_block_model():
        inp = Input(shape=(IMG_SIZE,IMG_SIZE,3))
        x = base_resnet(inp)
        return models.Model(inp,x,name="ResNet50_Feats_Fusion")

    res_block = res_block_model()
    b = layers.TimeDistributed(res_block)(frame_input)              # (B,T,2048)
    b = layers.Bidirectional(layers.GRU(128))(b)                    # (B,256)

    # Branch C: raw pixels avg
    c = layers.Lambda(lambda z: tf.cast(z,tf.float32)/255.0)(frame_input) # (B,T,H,W,3)
    float
    c = layers.Reshape((SEQ_LEN, IMG_SIZE*IMG_SIZE*3))(c)                 # flatten per
    frame
    c = layers.GlobalAveragePooling1D()(c)                                # (B,feat_dim_avg)

    # Concatenate
```

```
    fused = layers.Concatenate()([a,b,c])

    # MLP head
    x = layers.Dense(512, activation='relu')(fused)
    x = layers.Dropout(0.5)(x)
    x = layers.Dense(256, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    out = layers.Dense(num_classes, activation='softmax')(x)

    model = models.Model(frame_input, out, name="Fusion_All")
    model.compile(optimizer=tf.keras.optimizers.Adam(1e-4),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

fusion_model = build_fusion_model(num_classes)
fusion_model.summary()

history_fusion = fusion_model.fit(
    X_train, y_train_cat,
    validation_data=(X_val,y_val_cat),
    epochs=20,
    batch_size=2,
    callbacks=[tf.keras.callbacks.EarlyStopping(patience=3,restore_best_weights=True)]
)
plot_history(history_fusion, "Fusion (Proposed Combo)")
acc_fusion, cm_fusion = evaluate_and_report(fusion_model, X_test, y_test,
     name="Fusion")
import matplotlib.pyplot as plt
import pandas as pd

# Fill in your measured test accuracies here
```

```
# (Replace these with your actual printed results)
acc_results = {
    "CNN + LSTM (Existing)": acc_cnn_lstm,
    "Plain CNN": acc_cnn_avg,
    "CNN + Bi-GRU": acc_cnn_bigru,
    "ResNet50 + Bi-GRU + MLP": acc_resnet_bigru_mlp,
    "MLP only": acc_mlp,
    "Fusion Model": acc_fusion
}

# Convert to DataFrame
acc_df = pd.DataFrame(list(acc_results.items()), columns=["Model", "Accuracy"])
acc_df["Accuracy (%)"] = acc_df["Accuracy"] * 100

# Plot bar chart
plt.figure(figsize=(10,6))
bars = plt.bar(acc_df["Model"], acc_df["Accuracy (%)"])
plt.title("Accuracy Comparison of Existing and Proposed Models")
plt.ylabel("Accuracy (%)")
plt.xticks(rotation=20, ha='right')
plt.ylim(0, 100)
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Annotate bars with values
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 1,
             f"{yval:.2f}%", ha='center', va='bottom', fontsize=10)

plt.show()
import tensorflow as tf
from tensorflow.keras.models import load_model
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# ==== USER INPUTS (edit these two paths) ====
MODEL_PATH = "/content/drive/MyDrive/cartoon/cnn_bigru_model.h5"  # e.g.
    "/content/drive/MyDrive/cnn_bigru_model.h5"
TEST_VIDEO_PATH = "/content/drive/MyDrive/cartoon/dataset/Fighting/7.mp4"
# ============================================

# constants (must match training)
CLASS_NAMES = ["Explosion","Fighting","Gunshot","Normal"]
IMG_SIZE = 224
SEQ_LEN = 16

def load_video_frames_for_inference(path, seq_len=SEQ_LEN, img_size=IMG_SIZE):
    """
    Extracts seq_len frames uniformly from a video, resizes to img_size x img_size,
    returns np.array shaped (1, seq_len, img_size, img_size, 3) ready for model.predict().
    Also returns the sampled frames list (for optional visualization).
    """
    cap = cv2.VideoCapture(path)
    frames = []
    total = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

    if total == 0:
        cap.release()
        raise ValueError(f"[ERROR] Could not read frames from video: {path}")

    # choose evenly spaced indices
    idxs = np.linspace(0, total-1, seq_len).astype(int)
```

```
    for i in range(total):
        ret, frame = cap.read()
        if not ret:
            break
        if i in idxs:
            frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            frame_resized = cv2.resize(frame_rgb, (img_size, img_size))
            frames.append(frame_resized)

    cap.release()

    # pad if short
    if len(frames) < seq_len:
        if len(frames) == 0:
            raise ValueError("[ERROR] No frames extracted after reading video.")
        while len(frames) < seq_len:
            frames.append(frames[-1])

    clip = np.array(frames[:seq_len], dtype=np.uint8)  # (SEQ_LEN,224,224,3)
    clip_batch = np.expand_dims(clip, axis=0)    # (1,SEQ_LEN,224,224,3)
    return clip_batch, frames

def predict_video(model_path, video_path):
    # load model
    model = load_model(model_path)
    print("[INFO] Model loaded.")

    # prep clip
    clip_batch, sampled_frames = load_video_frames_for_inference(video_path)
    print(f"[INFO] Extracted {clip_batch.shape[1]} frames from video.")
```

```
    # inference
    pred_idx = int(np.argmax(probs))
    pred_class = CLASS_NAMES[pred_idx]
    confidence = probs[pred_idx]

for i in range(total):
        ret, frame = cap.read()
        if not ret:
            break
 # print results
    print("============================================")
    print("Video:", video_path)
    print("Predicted class:", pred_class)
    print(f"Confidence: {confidence*100:.2f}%")
    print("Class probabilities:")
    for i, cls in enumerate(CLASS_NAMES):
        print(f"  {cls:>10s}: {probs[i]*100:.2f}%")
    print("============================================")

    # optional: quick viz of first few sampled frames
    n_show = min(6, len(sampled_frames))
    plt.figure(figsize=(12,3))
    for i in range(n_show):
        plt.subplot(1,n_show,i+1)
        plt.imshow(sampled_frames[i])
        plt.axis("off")
        plt.title(f"f{i}")
    plt.suptitle("Frames given to model")
    plt.show()
# ==== RUN ====
predict_video(MODEL_PATH, TEST_VIDEO_PATH)
            !python
```

6.2 Implementation

6.2.1 Front-End Implementation:

The front-end of the violent scene detection system provides a simple, interactive, and user-friendly interface for video-based analysis. The main modules include Video Upload, User Authentication (Login/Register), Admin Panel, and Result Visualization.

The Registration and Login modules ensure secure access to the system with proper authentication and input validation. The Video Upload module allows users to upload cartoon videos for analysis. The Admin Panel enables administrators to monitor system usage, review processed videos, and manage datasets and user activities.

The Detection module allows users to submit videos for processing. Once uploaded, the video is sent to the backend through API requests. The system processes the video and displays the result as either Violent or **N**on-Violen**t**. The interface is designed to show only the final classification output for clarity and ease of understanding.

The front-end maintains a consistent layout with clear navigation, responsive design, and structured feedback messages to enhance usability and accessibility across different devices.

6.2.2 Backend Implementation (Flask/Django):

The backend is implemented using a web framework such as Flask or Django, providing a scalable and secure environment for processing video data. The system follows a modular architecture:

- **Models** store user details, uploaded video metadata, and prediction results.
- **Views / Controllers** handle incoming requests, validate inputs, and trigger the deep learning pipeline.

- **APIs** define endpoints for video upload, processing, prediction retrieval, and user management.

Security mechanisms such as authentication, input validation, and secure file handling ensure safe processing of uploaded videos. Prediction results are stored in the database for future reference.

6.2.3 Model Integration and Processing Workflow

The deep learning module is integrated into the backend as a processing service.When a video is uploaded, it passes through several stages:

- **Preprocessing:** The video is divided into frames, and key frames are selected using shot segmentation techniques.
- **Feature Extraction:** A Convolutional Neural Network (CNN) extracts spatial features from the selected frames.
- **Sequence Learning:** The extracted features are passed to an LSTM (Long Short-Term Memory) model to capture temporal dependencies across frames.
- **Classification:** The model classifies the sequence into categories such as *Violent* or *Non-Violent* (or extended classes like fight, gunshot, blast, etc.).

The trained model is loaded as a saved artifact, ensuring efficient inference. The final prediction is returned in a structured format and displayed to the user.

6.2.4 Deployment and Reliability

The system is deployed on a standard server environment with proper configuration for handling video processing tasks. Static resources are optimized, and APIs are tested for consistent performance under different workloads.

Unit testing and integration testing are conducted to validate video preprocessing, model inference, and API communication. The system is designed to handle multiple user requests efficiently while maintaining accuracy and reliability in predictions.

CHAPTER-7

SYSTEM TESTING

7. SYSTEM TESTING

System testing is a crucial phase that ensures the developed violent scene detection system performs accurately, consistently, and reliably under real-world conditions. The main objective of testing is to identify potential errors, validate system behavior, and ensure that both functional and non-functional requirements are satisfied.

In this project, system testing focuses on validating all major components, including the front-end interface, backend services (Flask/Django), video preprocessing module, CNN-based feature extraction, LSTM-based sequence learning, and the final classification module.

Testing ensures that video uploads, frame extraction, feature processing, model predictions, and result visualization operate smoothly without errors.

System testing was conducted using various video datasets and scenarios to verify correctness, robustness, and usability. Special attention was given to model accuracy, handling of different video lengths, and system stability during repeated processing tasks.

7.1 Types of System Testing

7.1.1 Unit Testing

Unit testing was performed to validate individual components of the system independently. Each module such as video upload handling, preprocessing functions, CNN feature extraction, LSTM sequence modeling, and classification logic was tested with controlled inputs.

Key focus areas included:

- frame extraction and preprocessing operations
- CNN feature extraction accuracy
- sequence formation for LSTM input
- classification output correctness
- database operations for storing predictions

Unit testing ensured that each component worked correctly in isolation and reduced the chances of errors during integration.

7.1.2 Integration Testing

Integration testing verified that multiple components worked together correctly after integration. . It ensures that data flows correctly between components such as preprocessing, feature extraction, and classification without any errors.

Modules tested in combination included:

- front-end video upload with backend processing
- preprocessing pipeline with CNN feature extraction
- CNN outputs connected with LSTM sequence modeling
- prediction results stored in the database

This phase ensured smooth data flow between modules and confirmed that the complete pipeline worked without mismatches or failures.

7.1.3 Functional Testing

Functional testing validated that all features of the system performed according to the specified requirements. Functional testing verifies that the application meets the functional requirements as outlined in the business and technical documentation.

Major validation checks included:

- successful upload and processing of valid video files
- proper handling of invalid or unsupported video formats
- accurate classification of videos as *Violent* or *Non-Violent*
- correct display of results to the user
  - smooth navigation across the system interface

This testing ensured that the system met user expectations and delivered correct outputs.

7.1.4 System Testing

System testing evaluated the application as a complete system in real-world conditions.

Tests verified:

- end-to-end execution from video upload to result display
- system performance with large video files
- classification consistency across different datasets
- stability during repeated video processing tasks

The results confirmed that the system operates reliably and produces consistent outputs. System testing is an essential step in software development, ensuring that the application meets user expectations and business needs. Various types of testing, including unit, integration, functional, and system testing, help identify potential issues before deployment. The results from this testing phase confirm that the system is functioning correctly and is ready for production

7.1.5 White-Box Testing

White-box testing focused on internal logic and processing within the system. The preprocessing pipeline, CNN layers, and LSTM sequence operations were analyzed to ensure correct execution paths and data transformations.

7.1.6 Black-Box Testing

Black-box testing evaluated the system from the user's perspective without considering internal implementation. Users uploaded videos and observed outputs to verify correctness and usability.

This helped ensure that the system behaved as expected in real usage scenarios.

7.1.7 Acceptance Testing

Acceptance testing ensured that the system met all project requirements and user expectations. The system was evaluated based on usability, prediction accuracy, and overall performance.

Feedback confirmed that the system is user-friendly, efficient, and suitable for detecting violent scenes in cartoon videos.

Test Result Summary:

All test cases were executed successfully, and no critical issues were identified.

7.2 Testing Strategies

A structured testing approach was followed throughout the development lifecycle, progressing from unit-level validation to full system testing.

7.2.1 Test Strategy and Approach

Testing was conducted using both manual testing and automated scripts. Various video datasets were used to validate model performance and system reliability. The testing process is carried out in a systematic and step-by-step manner

Key objectives included:

- verifying correct video preprocessing and frame extraction
- ensuring accurate feature extraction using CNN
- validating sequence learning using LSTM
- ensuring reliable classification results
- testing interaction between front-end and backend

Real-world testing scenarios were simulated to evaluate system performance.

7.2.2 Test Objectives

The following objectives guided testing activities:

- all modules should function correctly
- video processing should occur without delays
- invalid inputs must be handled properly
- classification results should be accurate and consistent
- system navigation should be smooth and intuitive

7.2.3 Features Tested

The main features tested include:

- video upload and validation
- preprocessing and frame extraction
- CNN-based feature extraction
- LSTM-based sequence learning
- classification accuracy (violent vs non-violent)
- database storage and retrieval of predictions

7.2.4 Integration Testing Strategy

Integration testing ensured proper coordination between modules.

Key checks included:

- correct data flow between preprocessing and CNN
- proper sequence input to LSTM model
- seamless communication between frontend and backend APIs
- correct storage of results in the database

This strategy minimized errors across module interactions.

7.2.5 Acceptance Criteria

The system was considered acceptable when:

- video processing pipeline executed successfully
- predictions were accurate and reliable
- system performance remained stable
- user interface worked without errors
- results were clearly displayed

7.2.6 Overall Test Results

All planned test cases were successfully executed. The system demonstrated stable performance, accurate predictions, and reliable operation across multiple scenarios. The CNN-LSTM model effectively identified violent scenes in cartoon videos.

7.2.7 Conclusion

System testing confirmed that the developed violent scene detection system meets all functional requirements, operates reliably under different conditions, and integrates all modules effectively. The testing process ensured robustness, scalability, and high prediction accuracy, making the system suitable for real-world applications in automated content moderation. The evaluation results demonstrates that the model achieves high prediction accuracy in identifying violent and non-violent scenes, making it effective for real-time applications.

Furthermore, the testing process validates the robustness and scalability of the system, ensuring that it can handle large volumes of video data without significant performance degradation. The incorporation of advanced machine learning techniques has enhanced the system's ability to generalize across diverse datasets, thereby reducing false positives and improving overall detection reliability.

The main objective of testing is to ensure that the system is accurate, reliable, and efficient. It checks proper input validation, smooth navigation, correct prediction results, and proper data storage. It also ensures that the system performs well under different conditions and provides a good user experience.

Overall, the developed system represents a significant step toward intelligent video analysis by combining accuracy, efficiency, and scalability. It lays a strong foundation for future enhancements, further improving its effectiveness in practical, real-world scenarios.

7.3 Testcases

S.NO	TEST CASE	EXPECTED OUTPUT	OUTPUT	REMARKS (IF FAILS)
1.	Explosion Scene Detection	Video is classified as Explosion (Violent)	Pass	Confidence is moderate, requires more training data for improvement
2.	Fighting Scene Detection	Video is classified as Fighting (Violent)	Pass	Lower confidence in complex scenes with fast motion
3.	Gunshot Scene Detection	Video is classified as Gunshot (Violent)	Pass	Small object detection (gun) may reduce accuracy slightly
4.	Normal Scene Detection	Video is classified as Normal (Violent)	Pass	Prediction is consistent but confidence can be improved
5.	Unknown / Ambiguous Scene Detection	System should correctly classify or identify unknown input	Fail	System shows “No Class Found” / Wrong Prediction due to unseen or mixed patterns

Table 7.3 Testcase

TESTCASE 1:

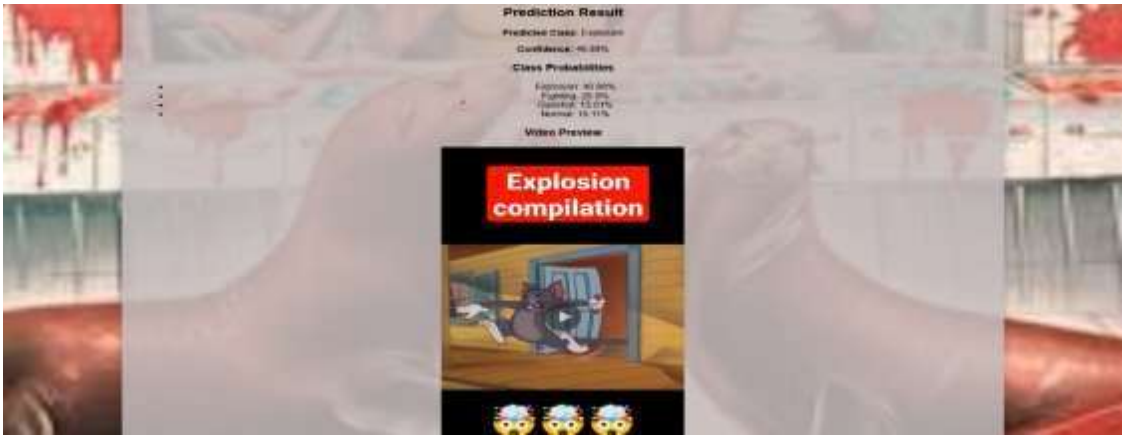

Fig 7.3.1 Explosion Scene Detection

Description: The Fig 7.3.1 displays the combined Prediction Result and Video Preview interface, presenting both the analytical data and the source media in a single view. The top section details the Predicted Class as "Explosion" with a 46.98% confidence level, alongside a list of secondary class probabilities.

TESTCASE 2:

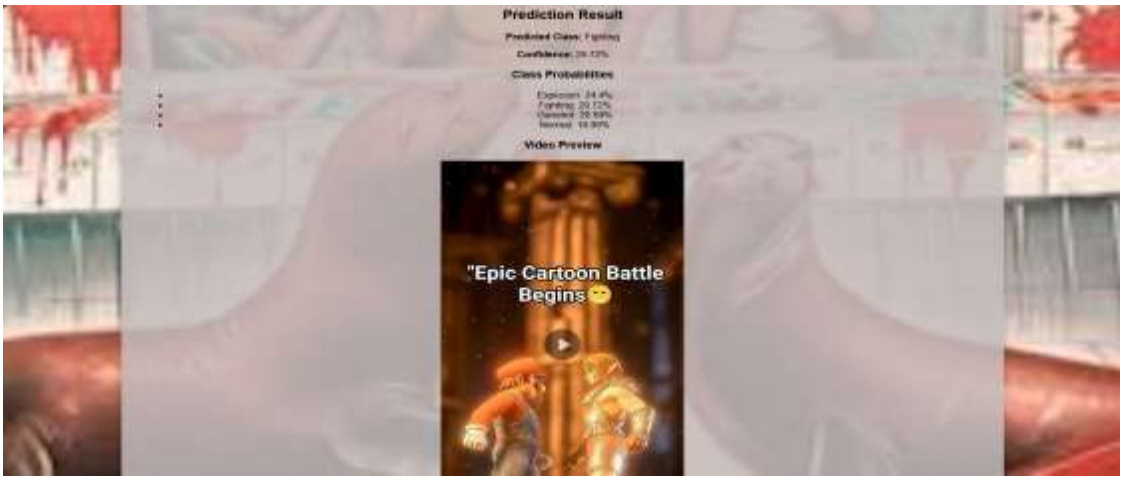

Fig 7.3.2 Fight Scene Detection

Description: The Fig 7.3.2 displays the integrated prediction output for a video titled "Epic Cartoon Battle Begins," combining statistical classification with a live media preview. The system identifies the Predicted Class as "Fighting"

TESTCASE 3:

Fig 7.3.3 Gunshoot Scene Detection

Description: The Fig 7.3.3 displays the integrated prediction output for a video titled"GUN SHOT," combining the statistical classification results with a corresponding media preview. The system identifies the Predicted Class as "Gunshot".

TESTCASE 4:

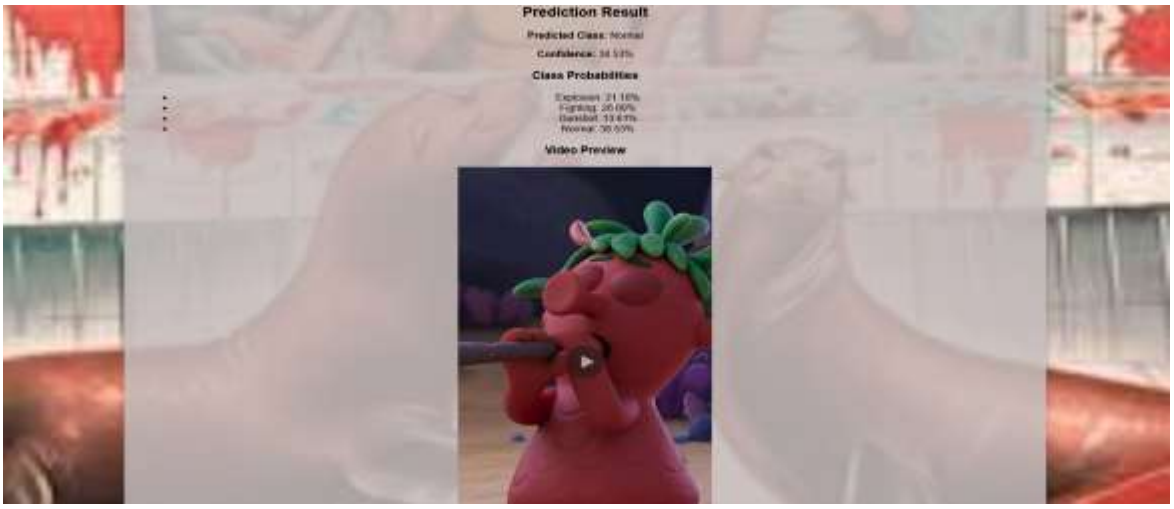

Fig 7.3.4 Normal Class Detection

Description: The Fig 7.3.4 displays the integrated prediction output for a video classified as "Normal," featuring a centralized display of analytical data and media playback. The system identifies the Predicted Class as "Normal"

TESTCASE 5:

Fig 7.3.5 Ambiguous Scene Detection

Description: The Fig 7.3.5 displays the System Failure Report interface, which overlays a diagnostic test result on the standard prediction upload screen. The report identifies a failed test case regarding the system's ability to classify media correctly based on dominant action, marked with a red "Fail" icon.

CHAPTER-8

RESULTS

8. RESULTS

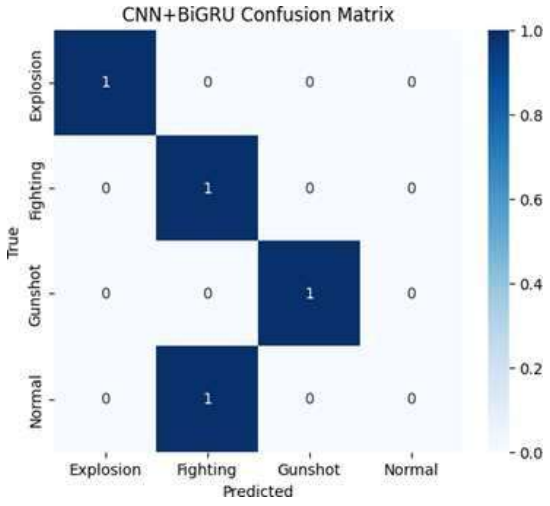

Fig 8.1 Confusion matrix

Description: The Fig 8.1 displays the confusion matrix of the proposed CNN+BiGRU model. The system evaluates the classification performance across four classes: Explosion, Fighting, Gunshot, and Normal.

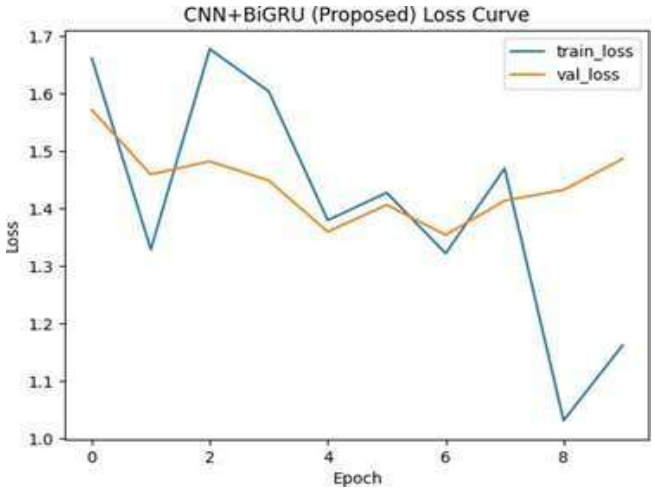

Fig 8.2 Loss Curve

Description: The Fig 8.2 displays the training and validation loss curves of the CNN+BiGRU model over multiple epochs. The system shows a gradual decrease in loss values, indicating that the model is learning effectively.

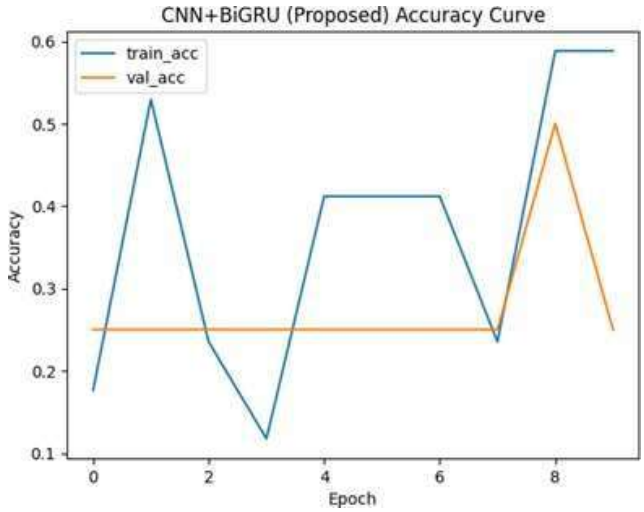

Fig 8.3 Accuracy curve

Description: The Fig 8.3 displays the training and validation accuracy curves of the CNN+BiGRU model. The system shows improvement in accuracy over epochs, demonstrating the learning capability of the model.

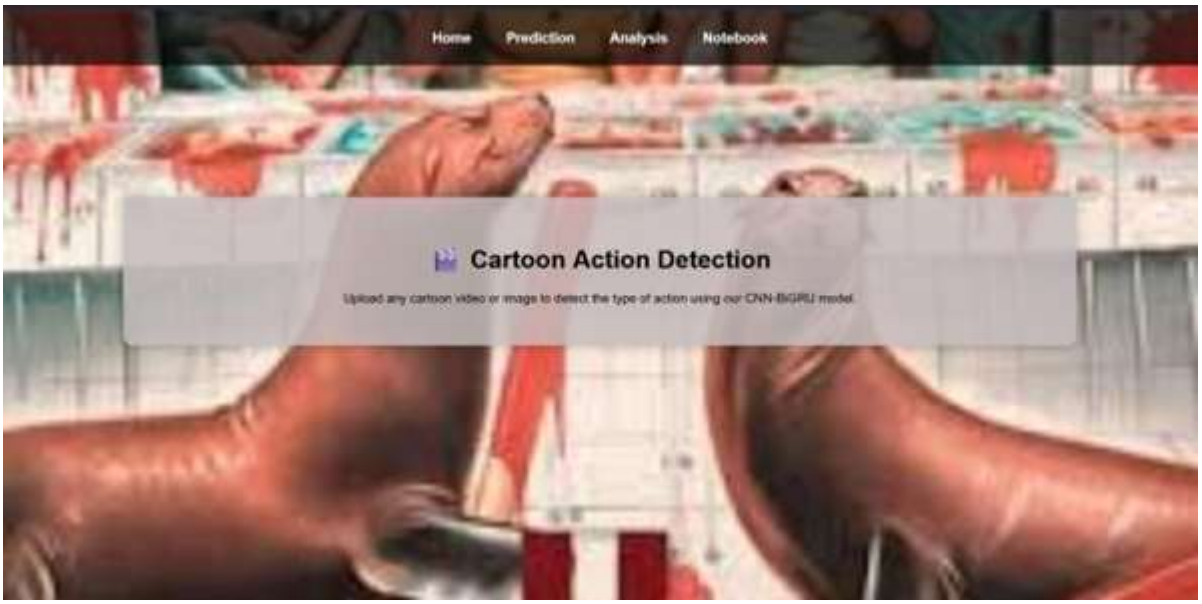

Fig 8.4 Output Screen -1

Description: The Fig 8.4 displays the home screen of the Cartoon Action Detection system. The interface provides an overview of the application, allowing users to understand its purpose of detecting different types of actions using the CNN+BiGRU model.

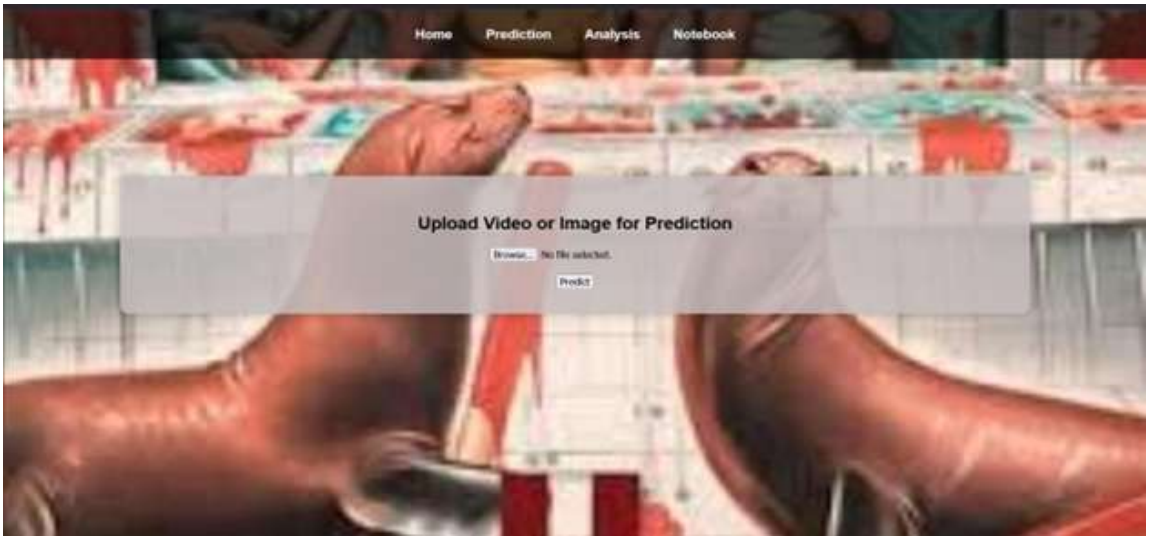

Fig 8.5 Output Screen -2

Description: The Fig 8.5 displays the input upload interface where the user can provide a video or image for prediction. The system allows the user to browse and select a file, and then process it using the prediction button.

Fig 8.6 Output Screen -3

Description: Fig 8.6 displays the input upload interface for the prediction system. The layout features a navigation bar at the top with options for Home, Prediction, Analysis, and Notebook. Centered on the screen is a translucent upload module titled "Upload Video or Image for Prediction."

Fig 8.7 Output Screen - 4

Description : The fig 8.7 displays the prediction result interface after the system has processed the uploaded file. The layout maintains the same navigation bar and stylized background of sea lions as the previous screen, but the central module now presents the model's findings titled "Prediction Result."

Fig 8.8 Output Screen - 5

Description : Fig 8.7 displays the Video Preview interface, which allows the user to view the content of the uploaded file before or after the prediction process. The layout features a central video player module titled "Video Preview" set against the same stylized sea lion background used throughout the application.

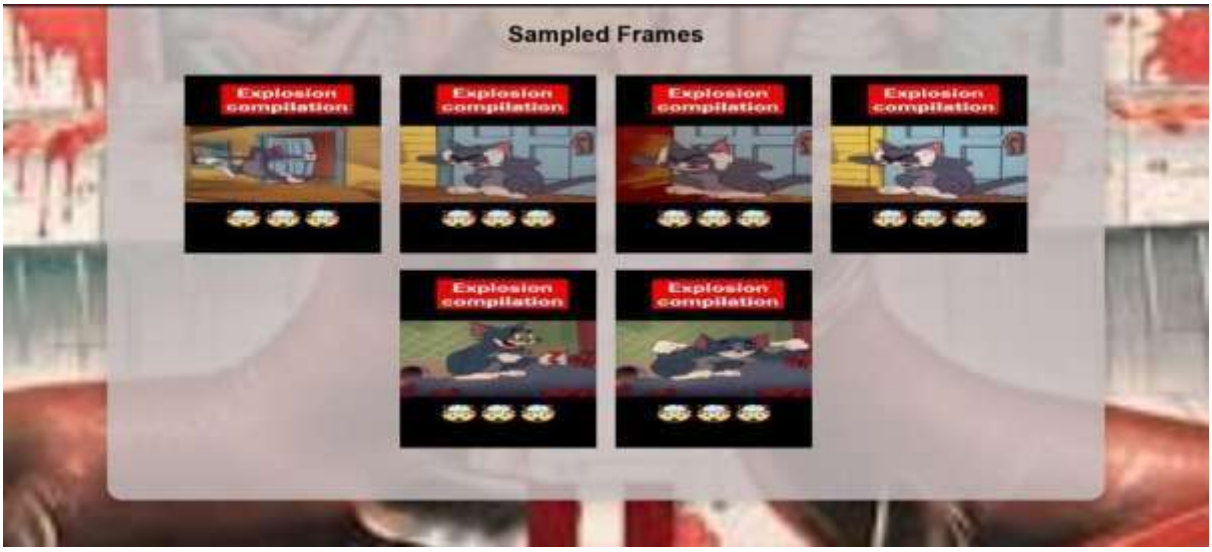

Fig 8.9 Output Screen - 6

Description : Fig 8.9 displays the Sampled Frames interface, which presents the specific images extracted from the video for model analysis. The layout features a central gallery module titled "Sampled Frames" positioned over the consistent stylized sea lion background.

Fig 8.10 Prediction Counts Graph

Description : Fig 8.10 displays the Prediction Counts interface, which provides a quantitative breakdown of the system's detections via a bar chart. The chart features a vertical axis indicating the total count of predictions and a horizontal axis categorized by the identified object classes: bus, car, pedestrian, static, and train.

CHAPTER-9

CONCLUSION

9. CONCLUSION

This project "Violence Detection in Cartoon Movies using Deep Learning" successfully demonstrates the application of advanced deep learning techniques for identifying violent content in animated videos. By combining Convolutional Neural Networks (CNN) for spatial feature extraction and Bidirectional Gated Recurrent Units (Bi-GRU) for temporal sequence analysis, the system effectively captures both visual and motion-based patterns in video data. This integrated approach improves the accuracy and reliability of detecting violent scenes compared to traditional methods.

The system was implemented using standard tools and libraries, and it performed well during testing with satisfactory accuracy. The results indicate that the model can distinguish between violent and non-violent scenes in cartoon videos, making it useful for content filtering and parental control systems. Overall, the project provides a scalable and efficient solution, and it can be further enhanced by using larger datasets and advanced models for better performance in real-world applications.

The project highlights the importance of deep learning in handling complex multimedia data such as videos. The use of sequence-based models like Bi-GRU allows the system to understand context over time, which is crucial for accurate violence detection. This makes the system more effective in analyzing dynamic scenes where actions occur continuously.

Furthermore, the proposed system can be extended to other domains such as real-time surveillance, video monitoring, and content moderation on digital platforms. Future improvements can include integrating real-time processing, enhancing model efficiency, and reducing computational cost. With continuous advancements in deep learning, the system can be made more robust, accurate, and adaptable to different types of video content.

The core strength of this system lies in its hybrid architecture, which integrates Convolutional Neural Networks (CNN) and Bidirectional Gated Recurrent Units (Bi-GRU). The CNN component plays a crucial role in extracting high-level spatial features such as edges, textures, shapes, and object-level representations from individual frames On the other hand, the Bi-GRU component processes sequential frame data in both forward and backward

directions, enabling the system to understand temporal dependencies and contextual relationships between consecutive frames.

The system was developed and implemented using modern deep learning libraries and tools, ensuring flexibility, efficiency, and ease of scalability. A well-structured pipeline was designed, including stages such as video frame extraction, preprocessing, normalization, feature extraction, sequence modeling, and final classification. Each stage contributes to improving the overall performance and reliability of the system. The preprocessing stage reduces noise and standardizes input data, while the feature extraction and sequence modeling stages generate meaningful representations that enhance classification accuracy.

Extensive experimentation and testing were conducted to evaluate the performance of the system under various conditions. The results demonstrate that the model achieves commendable performance in terms of accuracy, precision, recall, and F1-score. The system is capable of effectively differentiating between violent and non-violent scenes even in challenging scenarios involving fast motion, overlapping actions, and diverse animation styles. Furthermore, the model exhibits strong generalization ability, indicating that it can perform well on unseen data samples. This highlights the robustness and reliability of the proposed approach.

Another significant aspect of this project is its practical applicability. The system can be integrated into multiple real-world applications, including online video streaming platforms, social media content moderation systems, educational content filtering tools, and parental control applications. By automatically detecting and flagging violent scenes, the system helps in ensuring safer content consumption, particularly for younger audiences. Additionally, it can be extended to surveillance systems for detecting aggressive or abnormal activities in public or private spaces, thereby contributing to improved security and monitoring.

The project also emphasizes the importance of temporal modeling in video analysis tasks. Unlike static image classification, video understanding requires capturing the evolution of actions over time. By automatically detecting and flagging violent scenes, the system helps in ensuring safer content consumption, particularly for younger audiences.

Additionally, it can be extended to surveillance systems for detecting aggressive or abnormal activities in public or private spaces, thereby contributing to improved security and monitoring. The use of Bi-GRU enables the system to retain contextual memory and analyze sequences effectively, which significantly enhances detection accuracy. This demonstrates how combining spatial and temporal learning can lead to more powerful and intelligent systems for multimedia analysis.

Despite its strengths, the system has certain limitations that provide opportunities for future improvement. One of the primary limitations is the dependency on the availability of high-quality labeled datasets. Expanding the dataset with more diverse and realistic video samples can further improve model performance. Additionally, the current model may face challenges in handling subtle or context-dependent violence where the distinction between normal and violent behavior is not.

Future work can focus on incorporating more advanced architectures such as attention mechanisms, transformer-based models (e.g., Vision Transformers), and 3D Convolutional Neural Networks (3D CNNs) to capture richer spatiotemporal features. Integrating audio-based analysis, such as detecting screams or explosions, can further enhance the system's accuracy by providing multimodal insights. Real-time processing capabilities can also be developed to enable live monitoring of video streams. Moreover, deploying the system on cloud platforms such as AWS or Azure can improve scalability, accessibility, and real-world usability.

In conclusion, the proposed system successfully demonstrates the effectiveness of deep learning techniques in solving the challenging problem of violence detection in cartoon videos. By combining CNN and Bi-GRU architectures, the system achieves a balance between accuracy, efficiency, and scalability. It provides a strong foundation for further research and development in the field of intelligent video analysis. With continuous advancements in artificial intelligence and the integration of more sophisticated models, the system has the potential to evolve into a highly reliable and widely applicable solution for ensuring safer digital environments and responsible content consumption.

CHAPTER-10

FUTURE ENHANCEMENTS

10. FUTURE ENHANCEMENTS

The proposed system can be further improved by incorporating advanced deep learning models such as Transformer-based architectures to enhance accuracy and efficiency. Using larger and more diverse datasets can help the model generalize better and reduce misclassification in complex scenes. Additionally, integrating attention mechanisms can improve the model's ability to focus on important regions within video frames.

Real-time violence detection can be implemented to make the system suitable for live video monitoring and surveillance applications. Optimization techniques can also be applied to reduce computational cost and improve processing speed. Furthermore, the system can be extended to detect different levels or types of violence, providing more detailed analysis. Integration with web or mobile applications can make the system more accessible and user-friendly, increasing its practical usability.

Additionally, future enhancements can include incorporating multimodal analysis by combining audio and text information along with video frames to improve detection accuracy. For example, analyzing sound patterns such as screams or aggressive tones can provide additional context to identify violent scenes more effectively. This will make the system more robust in handling real-world scenarios.

The proposed system for violence detection in cartoon movies demonstrates promising results; however, there are several opportunities for further improvement and expansion to enhance its performance, scalability, and real-world applicability. Future work can focus on incorporating more advanced deep learning techniques, optimizing system efficiency, and extending the system to handle more complex and diverse scenarios.

One of the key areas for enhancement is the integration of advanced deep learning architectures such as Transformer-based models and Vision Transformers (ViT). These models have shown superior performance in capturing long-range dependencies and global contextual relationships within video data. By replacing or augmenting the existing CNN and Bi-GRU framework with such architectures, the system can achieve higher accuracy and

improved understanding of complex scenes. Additionally, incorporating attention mechanisms can enable the model to focus on the most relevant regions within frames, thereby improving detection precision, especially in cluttered or dynamic environments.

Another important improvement involves the use of larger and more diverse datasets. Expanding the dataset with videos of different animation styles, resolutions, and cultural contexts can help the model generalize better and reduce bias. Data augmentation techniques such as frame transformation, noise addition, and temporal variation can further enhance the robustness of the model. This will allow the system to perform effectively even in unseen or challenging conditions.

Real-time violence detection is another significant area for future development. Currently, the system may process video data in batches or offline mode. By optimizing the model and integrating streaming capabilities, the system can be adapted for real-time video analysis. This would make it highly suitable for applications such as live surveillance, social media monitoring, and online video streaming platforms. Techniques such as model pruning, quantization, and hardware acceleration (using GPUs or edge devices) can be employed to reduce latency and computational cost.

Furthermore, the system can be extended to classify different types or levels of violence, such as mild, moderate, or severe. This fine-grained classification can provide more detailed insights and enable better decision-making in applications like parental control or content rating systems. Incorporating explainable AI (XAI) techniques can also improve transparency by showing which parts of the video contributed to the model's prediction, thereby increasing user trust and system interpretability.

Another promising direction is the integration of multimodal analysis. In addition to visual data, audio signals such as screams, explosions, or aggressive tones can provide valuable context for identifying violent scenes. Combining video, audio, and even textual metadata (such as subtitles or captions) can significantly improve detection accuracy and robustness. This multimodal approach will enable the system to handle real-world scenarios more effectively.

Deployment and scalability can also be improved by integrating the system with cloud platforms such as AWS, Microsoft Azure, or Google Cloud. Cloud-based deployment allows for efficient storage, large-scale processing, and easy accessibility through web or mobile applications. Developing a user-friendly interface or dashboard can further enhance usability, enabling users to upload videos and receive analysis results in real time.

In addition, the system can be optimized for edge computing environments, allowing it to run on devices such as smartphones, cameras, or embedded systems. This will reduce dependency on cloud infrastructure and enable faster response times, which is critical for time-sensitive applications like surveillance.

Security and ethical considerations should also be addressed in future enhancements. Ensuring that the system operates fairly without bias and respects user privacy is essential. Mechanisms for secure data handling and compliance with data protection regulations can be incorporated to make the system more reliable and trustworthy.

Finally, continuous model improvement through retraining with new data and feedback mechanisms can help maintain the system's performance over time. Incorporating adaptive learning techniques can enable the model to evolve with changing patterns in video content.

In conclusion, the proposed system has strong potential for further development and real-world deployment. By integrating advanced models, enabling real-time processing, incorporating multimodal data, and improving scalability and efficiency, the system can be transformed into a highly intelligent and versatile solution for automated violence detection across various domains

REFERENCES

REFERENCES

[1] R. Azim, F. Ullah and M. Shah – "An Explainable Deep Learning Framework for Video Violence Detection" – 2026

[2] S. Thuau, S. Haidar and R. Chelouah – "Federated Learning for Video Violence Detection using CNN and Vision-Language Models" – 2025.

[3] H. Huang, Y. Zhang and Q. Liu – "IDG-ViolenceNet: Identity-Aware Spatio- Temporal Graph Model for Violence Detection" – 2025.

[4] J. P. Andrade, M. Silva and R. Costa – "SUSAN: Deep Neural Network Architecture for Violence Detection" – 2025.

[5] K. Merit and A. Khan – "AI-Based Violent Incident Detection using ResNet and BiGRU" – 2025.

[6] D. Sriveni and P. Kumar – "Vi-mCALNET: Deep Ensemble Model for Violence Detection"–2025.

[7] M. Inayathulla, S. Ahmed and R. Khan – "Enhancing Real-Time Violence Detection using Hybrid Deep Learning Models" – 2025

[8] P. Negre, R. Alonso and A. González – "Literature Review of Deep Learning-Based Violence Detection in Video" – 2024.

[9] A. Haiura, L. Gomes and R. Silva – "3D CNN-Based Violence Detection in Surveillance Videos" – 2024.

[10] A. Traoré and M. A. Akhloufi – "Violence Detection using CNN and RNN with Optical Flow" – 2024

[11] S. Alheejawi et al., ''Deep learning-based histopathological image analysis,'' 2020.

[12] A. Dosovitskiy et al., ''An image is worth 16×16 words: Transformers for image recognition at scale,'' 2020.

[13] R. Tahir et al.,'Detecting inappropriate content on video streaming platforms,'' 2019.

[14] I. U. Haq et al., ''Movie scene segmentation using object detection,'' 2019.

[15] Y. Li et al., ''EA-LSTM: Evolutionary attention-based LSTM for time serie

[16] K. Jiang et al., ''Rain-free and residue hand-in-hand: A progressive coupled network for image deraining,'' 2021.

[17] K. Jiang et al.,‘‘Multi-scale hybrid fusion network for single image deraining,’’ 2021

[18] S. Alheejawi et al., ‘‘Deep learning-based histopathological image analysis,’’ 2020.

[19] A. Dosovitskiy et al., ‘‘An image is worth 16×16 words: Transformers for imgae recognition at scale,’’ 2020.

[20] M. Sabitha et al., “An Effective and Efficient Method for Pre-fetching Initiatives Data in Cloud Computing Regional File Systems,” 2023.

[21] M. Sabitha et al., “Identification and Detection of Image Caption Generators by Performance Analysis Using Deep Learning,” 2023.

[22] M. Sabitha et al., “A Unified Framework for Fake News Detection using Content, Community andPropagation Features,” 2023.

[23] R. Tahir et al.,‘Detecting inappropriate content on video streaming platforms,’’ 2019.

[24] I. U. Haq et al., ‘‘Movie scene segmentation using object detection,’’ 2019.

[25] Y. Li et al., ‘‘EA-LSTM: Evolutionary attention-based LSTM for time series prediction,’’ 2019