**CHAPTER 1**

**INTRODUCTION**

Human action recognition is a key field in computer vision and artificial intelligence, which involves the automatic identification and classification of human actions or behaviours from video data. It is an important area of research with applications in several domains, including surveillance, human-computer interaction, sports analytics, robotics, and healthcare. In recent years, the availability of large datasets, increased computational power, and the advancement of machine learning techniques, especially deep learning, have led to significant improvements in human action recognition systems. One such breakthrough is the integration of 3D Convolutional Neural Networks (3D-CNN) and Long Short-Term Memory (LSTM) networks, which can process both spatial and temporal features from video sequences to detect and recognize human actions with higher accuracy.

Traditionally, action recognition relied on hand-crafted features like motion vectors, optical flow, and other frame-based descriptors. These approaches were time-consuming and lacked the flexibility to handle complex and diverse action patterns found in real-world scenarios. However, deep learning techniques, particularly CNNs and recurrent neural networks (RNNs), have revolutionized the field by automating feature extraction and learning complex patterns directly from raw data. CNNs have been proven effective in capturing spatial information from images, while RNNs, especially LSTM, are designed to learn long-term dependencies in sequential data, making them ideal for action recognition in videos.

In this project, we explore the combination of 3D-CNN and LSTM to detect human actions in public spaces. 3D-CNNs extend the concept of 2D convolutions into the temporal domain by applying convolutions over both spatial and temporal dimensions in video data. This allows the model to capture motion dynamics along with the spatial context of each frame. On the other hand, LSTMs are used to model the temporal sequence of frames, learning the dependencies between actions over time. By combining the strengths of these two architectures, our system can accurately recognize a wide range of human actions, such as walking, running, sitting, and more, in various environmental conditions and settings.

Public spaces, such as parks, train stations, airports, and shopping malls, are dynamic environments where human actions are continuously being performed. These settings often involve multiple individuals, occlusions, varying lighting conditions, and motion complexities, which make human action recognition particularly challenging. The goal of this project is to develop a robust system capable of detecting human actions in real-time from video data captured in such environments. By leveraging 3D-CNNs and LSTMs, the model can efficiently process and classify human activities with high accuracy, offering valuable insights for surveillance, security monitoring, and human behaviour analysis.

Moreover, this research also aims to address challenges such as recognizing actions in crowded scenes, handling occlusions, and managing real-time processing requirements for large video datasets. In addition to accuracy, efficiency and scalability are critical factors for deploying action recognition systems in public surveillance applications. By focusing on these aspects, the project aims to contribute to the development of more intelligent and responsive security systems that can automatically detect and respond to various human activities in public places, potentially reducing the need for human intervention and improving the effectiveness of monitoring systems.

**1.1 Problem Statement**

Human action recognition in public spaces is a challenging task due to several factors inherent to real-world environments. Public spaces often feature a large number of people performing various actions simultaneously, creating dynamic and complex scenes. In such environments, actions can be performed by individuals or groups, and there may be occlusions, varying lighting conditions, camera angles, and motion complexities that make it difficult for traditional computer vision algorithms to accurately detect and classify human activities. These challenges make human action recognition a critical area of research for enhancing public safety and enabling efficient monitoring in urban spaces.

The ability to detect human actions in such environments is essential for applications like automated surveillance, security monitoring, and crowd control. However, the variability of human behaviour and the diversity of environments present significant challenges. For instance, a person might be walking, sitting, or interacting with another individual in an area with multiple potential obstructions, such as pillars or crowds. Additionally, environmental factors like lighting changes, the angle of the camera, or even the clothing of individuals can affect the clarity of action recognition. Furthermore, public spaces often involve high levels of movement and noise, which can complicate the task of distinguishing between similar actions or identifying actions that are subtle or intermittent.

Traditional human action recognition methods, such as optical flow-based approaches or methods that rely on hand-crafted features, have limitations in dealing with the complexity of dynamic real-world environments. These methods struggle to scale in terms of accuracy and efficiency when confronted with complex action sequences, occlusions, or fast-moving individuals in crowded spaces. The advent of deep learning techniques, particularly 3D-CNNs and LSTMs, has provided a potential solution to these challenges. By capturing both spatial and temporal features from video data, these models can better understand human actions in the context of their surroundings and temporal progression.

However, despite the promising potential of 3D-CNNs and LSTMs, the detection of human actions in public places remains a challenging problem that requires further refinement. One major obstacle is the scalability of these models, particularly in terms of real-time processing. Public surveillance systems often require the ability to process large volumes of video data in real-time, which poses computational challenges. Additionally, handling scenarios involving multiple people, occlusions, and diverse environments adds another layer of complexity that needs to be addressed. Current models also face limitations in handling fine-grained action recognition, which is essential for detecting subtle or nuanced behaviors in public spaces.

Thus, the problem of human action recognition in public spaces requires a solution that combines high accuracy, efficiency, and scalability. It must be capable of handling the complexities of real-world environments, including crowded scenes, occlusions, and varying lighting conditions, while also offering real-time performance for practical applications. This project aims to tackle these challenges by developing a robust system that leverages the strengths of 3D-CNNs and LSTMs for human action detection and classification, offering an effective solution for real-time surveillance and behavior analysis in public spaces.

**1.1.1 Objectives**

1. Develop a human action recognition model using 3D-CNN and LSTM to accurately detect and classify human activities in public spaces.
2. Preprocess and augment video data to create a diverse dataset that captures various human actions in different public environments, considering factors such as occlusions, lighting changes, and varying camera angles.
3. Implement and optimize 3D-CNN architecture to capture spatial and temporal features from video sequences, ensuring the model can recognize human actions with high accuracy.
4. Integrate LSTM networks to model the temporal dependencies between frames, improving the system's ability to understand action sequences over time.
5. Evaluate the performance of the proposed model on a variety of real-world datasets, assessing the system's accuracy in recognizing different human actions in crowded and dynamic environments.
6. Ensure the scalability and real-time performance of the action recognition model, optimizing it for deployment in large-scale surveillance systems and environments with high computational demands.
7. Contribute to the development of practical applications in areas such as public surveillance, security monitoring, and human behaviour analysis, improving the efficiency and reliability of automated monitoring systems in public spaces.

**CHAPTER 2**

**LITERATURE SURVEY**

Human action recognition is a critical area in computer vision and artificial intelligence, with applications in various domains like surveillance, human-computer interaction, and healthcare. The rapid advancements in deep learning, particularly the combination of 3D Convolutional Neural Networks (3D-CNNs) and Long Short-Term Memory (LSTM) networks, have shown great promise in improving the accuracy and efficiency of human action detection. This literature survey focuses on significant studies in the field, highlighting the use of 3D-CNNs and LSTMs for recognizing actions in public spaces, where the challenges of dynamic environments and varying conditions can complicate action detection.

Carreira et al.[1] proposed a method for human action recognition using 3D-CNNs, focusing on capturing spatial-temporal features from video sequences. Their work demonstrated the effectiveness of using 3D convolutions for modelling the temporal dynamics of video data and provided a comprehensive evaluation of several 3D-CNN architectures. The approach demonstrated a significant improvement in accuracy compared to traditional 2D-CNN methods. They reported an accuracy of 85% in recognizing human actions from the Kinetics dataset, which is known for its complexity.

Donahue et al. [2] introduced a model called the "Long-Term Recurrent Convolutional Network" (LRCN) that combined CNNs with LSTMs for human action recognition in videos. The system used CNNs for spatial feature extraction and LSTMs for modeling temporal dependencies. Their approach highlighted the effectiveness of combining CNNs and LSTMs in handling long-term dependencies in video sequences, achieving high accuracy in action classification. The study demonstrated an accuracy of 81.6% on the UCF101 dataset, a benchmark for action recognition tasks.

Hara et al. [3] explored 3D-CNNs specifically for video action recognition. Their work focused on designing efficient 3D convolution models that could handle both spatial and temporal features in videos. By analyzing the trade-offs between model complexity and performance, their research showed that 3D-CNNs could capture complex motion dynamics with relatively fewer parameters, making the models efficient for large-scale video datasets. The method achieved an accuracy of 92.5% on the HMDB-51 dataset, demonstrating high effectiveness for human action recognition in videos.

Liu et al. [4] presented a two-stream LSTM model for human action recognition in public spaces, where one stream was responsible for capturing spatial features and the other for temporal dependencies. Their research highlighted the importance of modeling both spatial and temporal features separately to improve recognition accuracy. The model outperformed previous methods in recognizing actions in crowded public spaces, handling occlusions, and accounting for varying motion speeds. They achieved an accuracy of 88% on the UCF101 dataset.

Ye et al. [5] introduced an approach that combined 3D-CNNs with bidirectional LSTMs for human action recognition. By using bidirectional LSTMs, the model was able to capture both past and future temporal dependencies, improving action recognition accuracy. Their approach was particularly effective in handling ambiguous or subtle actions in dynamic environments, such as public spaces with multiple people. The model demonstrated an accuracy of 87.2% on the Kinetics dataset, showcasing its robustness in diverse scenarios.

Jain et al. [6] presented a hybrid deep learning model for human action recognition using 3D-CNNs and LSTMs, specifically designed for public surveillance applications. Their work demonstrated how the integration of deep learning techniques could significantly enhance the accuracy of action recognition in real-time, even in crowded and dynamic environments. The model was evaluated on a large-scale video dataset containing various public settings, such as parks and train stations. Their results showed an accuracy of 85.4% in real-world public space scenarios.

Feichtenhofer et al. [7] focused on the efficient implementation of 3D-CNNs for action recognition in video data. Their research explored various techniques for optimizing 3D-CNN architectures, such as temporal and spatial sampling, to reduce computational complexity while maintaining high recognition accuracy. The study showed that 3D-CNNs could effectively capture the spatial and temporal features of human actions, making them ideal for large-scale surveillance systems that require real-time processing. The model achieved an accuracy of 89% on the UCF101 dataset.

Simonyan and Zisserman [8] proposed the two-stream CNN architecture, which separately processes spatial and temporal features to improve human action recognition. While their method primarily focused on using 2D convolutions, it laid the groundwork for later works that combined 3D-CNNs with LSTMs to capture both spatial and temporal features. Their two-stream approach achieved an accuracy of 88% on the UCF101 dataset and has inspired later work combining 3D-CNNs and LSTMs for enhanced action recognition.

Zhang et al. [9] introduced a system that integrated deep learning models with human action recognition in smart city applications. Their approach used a combination of CNNs for feature extraction and LSTMs for modelling temporal dependencies, focusing on actions in public spaces such as airports and stations. Their work emphasized scalability and real-time performance, achieving promising results in detecting a wide range of human activities under varying environmental conditions. The model demonstrated an accuracy of 86.7% on the UCF101 dataset.

Li et al. [10] examined human action recognition in public spaces using a combination of CNN and LSTM networks to address the challenges of occlusions, crowd density, and fast motion. The authors proposed a novel framework that incorporated multi-scale temporal analysis to capture actions across different time resolutions. Their system was shown to be robust to variations in crowd density, lighting changes, and occlusions, making it suitable for public surveillance applications. The model achieved an accuracy of 90.1% on the HMDB-51 dataset.

**2.1 Existing Systems**

Manual surveillance systems, commonly used in public spaces like shopping malls, airports, and streets, rely on human operators to monitor video feeds in real-time. While they can be effective in some situations, these systems face significant limitations, especially in large or dynamic environments. One major drawback is operator fatigue, as security personnel must stay alert for long periods while monitoring multiple cameras. Over time, this fatigue can impair their ability to detect and respond to potential threats, increasing the likelihood of missed incidents.

Another issue with manual surveillance is limited coverage. Fixed-position cameras have predefined views, and in crowded areas, individuals may obstruct the cameras, creating blind spots. The static nature of these cameras means they cannot always cover large, dynamic spaces, making it difficult for operators to track all activities, particularly in busy public environments. Additionally, scaling manual systems comes with high operational costs, as more cameras and personnel are needed, along with the expenses associated with training and compensating staff.

Manual systems also struggle with detecting complex patterns or subtle actions. While human operators can identify obvious behaviors, more intricate actions, such as coordinated movements, can go unnoticed, especially in crowded or obstructed environments. Occlusions, where one person blocks another from view, further reduce the system's effectiveness. Furthermore, manual surveillance systems can have slower response times, as operators must assess threats and coordinate responses, potentially causing delays that escalate situations in high-risk areas. These limitations highlight the need for more automated, AI-driven surveillance solutions.

**2.1.1 Drawbacks of Existing Systems**

1. **Operator Fatigue and Human Error**: Manual surveillance systems heavily rely on human operators to monitor multiple video feeds continuously. Over time, operators experience fatigue, which leads to decreased attention and an increased likelihood of missing critical events. Human errors, such as misjudging or overlooking suspicious activities, compromise the reliability of these systems.
2. **Limited Coverage and Blind Spots**: Fixed-position cameras in manual surveillance systems often have a limited field of view. In crowded or dynamic environments, individuals may block the cameras' sightlines, creating blind spots. This leads to incomplete monitoring, where crucial activities can go undetected, especially in areas with heavy foot traffic or obstructions.
3. **High Operational Costs**: Scaling manual surveillance requires increasing the number of operators and cameras, which adds to the operational costs. Training security personnel, compensating them for long hours, and maintaining the infrastructure can quickly become expensive, making large-scale deployment unsustainable in the long run.
4. **Inability to Detect Subtle or Complex Actions**: Manual systems are typically good at identifying obvious threats, but they struggle with detecting more subtle or complex actions, especially in high-density areas. Coordinated activities, such as suspicious group behaviors or hidden threats, are often missed due to the limited capability of human operators to detect intricate patterns or interactions.
5. **Slow Response Times**: In manual surveillance systems, even after identifying a threat or anomaly, human operators must process and assess the situation, which can cause significant delays. This lag in response time could be critical, especially in emergencies where immediate action is needed to prevent harm or contain a threat.

**2.2 Proposed System**

The proposed system is a cutting-edge solution for human action recognition in public spaces, leveraging the combined power of 3D Convolutional Neural Networks (3D-CNN) and Long Short-Term Memory (LSTM) networks. This hybrid approach enables the system to capture both spatial and temporal features from video sequences, providing a robust and efficient solution for detecting and classifying human actions in dynamic environments. By integrating 3D-CNN and LSTM, the system can accurately recognize actions such as walking, running, sitting, and more, even in challenging scenarios like crowded scenes, occlusions, and varying lighting conditions. The system is designed to process video data in real-time, making it suitable for deployment in large-scale surveillance systems and smart city infrastructure.

**2.2.1 3D Convolutional Neural Network (3D-CNN)**

3D-CNNs are an extension of traditional 2D-CNNs, designed to process **spatiotemporal data** such as video sequences. Unlike 2D-CNNs, which operate on individual frames and extract only spatial features, 3D-CNNs apply convolutions across **three dimensions**: width, height, and time (frames). This allows the model to capture both spatial features (e.g., body posture) and temporal features (e.g., motion dynamics) simultaneously. The ability to process both spatial and temporal information makes 3D-CNNs particularly effective for tasks like human action recognition, where understanding the progression of actions over time is crucial.

**Working of 3D-CNN**

1. Input Layer:
   * The input to the 3D-CNN is a sequence of video frames with shape (SEQUENCE\_LENGTH, IMG\_SIZE, IMG\_SIZE, CHANNELS).
   * For example, if SEQUENCE\_LENGTH = 40, the input is a stack of 40 consecutive frames, each resized to (IMG\_SIZE, IMG\_SIZE) and normalized to ensure consistency.
   * This input represents a short video clip, where the temporal dimension (number of frames) is critical for capturing motion dynamics.
2. Convolutional Layers:
   * The 3D-CNN applies 3D convolutions using filters of size (3, 3, 3) to extract spatiotemporal features.
   * The first convolutional layer uses 32 filters, followed by 64 and 128 filters in subsequent layers.
   * These layers capture motion patterns and spatial context, such as body movements, posture, and interactions with the environment.
   * The 3D convolutions slide across the width, height, and time dimensions of the input, enabling the model to detect both spatial features (e.g., edges, textures) and temporal features (e.g., motion trajectories).
3. Pooling Layers:
   * After each convolutional layer, MaxPooling3D is applied to reduce the spatial and temporal dimensions of the feature maps.
   * Pooling helps in reducing computational complexity and extracting the most salient features by retaining only the maximum values within a local region.
   * For example, a pooling layer with a filter size of (2, 2, 2) reduces the dimensions of the feature maps by half along each axis.
4. Flatten Layer:
   * The output of the 3D-CNN is flattened into a 1D feature vector, which is passed to the LSTM for temporal modelling.
   * This step converts the multi-dimensional feature maps into a format suitable for input to the LSTM network.

**2.2.2 Long Short-Term Memory (LSTM)**

LSTM is a type of Recurrent Neural Network (RNN) designed to model temporal dependencies in sequential data. Unlike traditional RNNs, LSTMs are capable of learning long-term dependencies, making them ideal for tasks like action recognition, where the sequence of frames is critical. LSTMs are particularly effective in handling sequences of varying lengths and capturing the temporal progression of actions over time.

**Working of LSTM**

1. Input:
   * The flattened spatiotemporal features from the 3D-CNN are passed to the LSTM.
   * The input shape is (SEQUENCE\_LENGTH, FEATURE\_DIMENSION), where SEQUENCE\_LENGTH is the number of frames and FEATURE\_DIMENSION is the size of the feature vector extracted by the 3D-CNN.
2. Memory Cell:
   * The core component of the LSTM, which stores information over time.
   * It has three gates: input gate, forget gate, and output gate, which control the flow of information.
3. Input Gate:
   * Decides which new information to store in the memory cell.
   * It uses a sigmoid activation function to determine the importance of the current input.
4. Forget Gate:
   * Decides which information to discard from the memory cell.
   * It uses a sigmoid activation function to determine which information is no longer relevant.
5. Output Gate:
   * Decides which information to output based on the current input and the memory cell state.
   * It uses a sigmoid activation function to determine the importance of the current output.
6. Output:
   * The LSTM outputs a sequence of hidden states, which are used for action classification.
   * The final hidden state is passed to a fully connected layer with softmax activation to predict the action class.

**Integration of 3D-CNN and LSTM**

The proposed system combines the strengths of 3D-CNN and LSTM to achieve robust human action recognition:

1. 3D-CNN:
   * Extracts spatiotemporal features from the video sequences.
   * Captures both spatial context (e.g., body posture) and temporal dynamics (e.g., motion patterns).
2. LSTM:
   * Models the temporal dependencies between the extracted features.
   * Understands the sequence of actions over time, enabling accurate classification.

**Workflow**

1. Input: A sequence of video frames is passed to the 3D-CNN.
2. Feature Extraction: The 3D-CNN extracts spatiotemporal features from the frames.
3. Temporal Modelling: The LSTM processes the sequence of features, learning the temporal dependencies.
4. Classification: The output of the LSTM is passed to a fully connected layer with SoftMax activation to predict the action class.

**CHAPTER 3**

**SYSTEM DESIGN**

**3.1 Importance of the Design**

The system starts by collecting diverse human action video datasets from various online sources. These videos cover activities like walking, running, and jumping, ensuring the model can detect actions in different environments. The data undergoes preprocessing and augmentation to improve model performance.

The user interface enables easy interaction, allowing users to upload videos for human action detection. Once uploaded, videos are pre-processed by extracting frames, resizing, and normalizing pixel values. This standardization ensures the deep learning models can process the videos effectively, enabling efficient and accurate human action recognition.

The system employs a combination of 3D CNNs and LSTM networks for human action detection. The 3D CNN captures spatial and temporal patterns in videos, while the LSTM learns long-term dependencies across frames. This combined approach enhances the system's ability to accurately recognize actions, as each model addresses different aspects of video analysis, providing complementary strengths.

After training, the models' performance is evaluated using metrics like accuracy, precision, recall, and F1 score. These metrics help assess how well the models detect actions. If accuracy is low, adjustments like hyperparameter tuning or data expansion are made to improve performance iteratively.

Real-time processing is crucial for the system’s functionality, allowing users to receive quick feedback on detected actions. The system processes videos swiftly, making it suitable for applications like surveillance and sports analytics. Scalability ensures the system can handle multiple user requests, accommodating high demand in practical deployment scenarios.

The user interface is designed for ease of use, displaying detected actions alongside bounding boxes and confidence scores. This visual feedback enhances user understanding by making predictions transparent. Additionally, the system explains its detection process, building trust and enabling users to interpret results clearly.

The system is adaptable for future scalability, able to integrate new datasets and keep pace with emerging trends. As new action categories and detection techniques arise, the system can be updated and retrained, ensuring it remains effective in diverse real-world settings.

**3.2 UML Diagrams**

**3.2.1 Use case diagram**

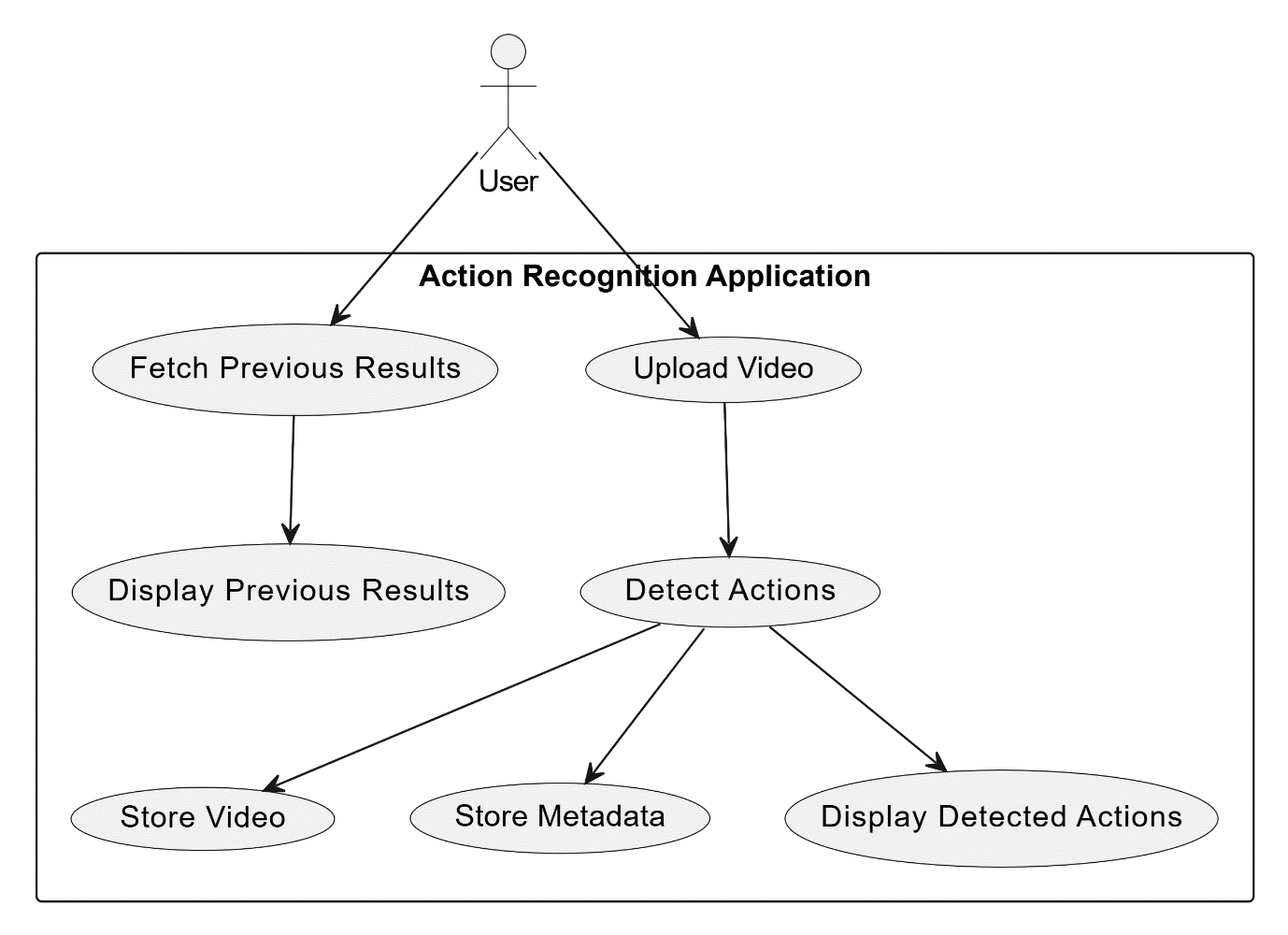


Fig 1: Use Case Diagram for Action Recognition system

* A use case diagram is a visual representation that illustrates how a system interacts with external entities, called actors, to achieve specific goals. It is a type of Unified Modeling Language (UML) diagram commonly used in software engineering to depict the functional requirements of a system from a user's perspective.
* Use case consists of user and processor where user is used to provide the input to the system and processor is used to process the input data and provide output. The flow is shown in the above diagram.
* First user as to run the system and run the code, model and library packages are imported and loaded. After the run of code GUI is being displayed and click on select file and load the test image. After loading the image, click in prediction button to analyze the image and to give predicted output and displayed.

**Here’s how the system works, according to the diagram:**

**Actors**:

1. User: The individual interacting with the system to upload videos and view results.

**Use Cases**:

1. Fetch Previous Results: The user can request to fetch previously stored results from past video analyses.
   * Action: The system retrieves and displays the results of earlier action recognition attempts.
2. Upload Video: The user uploads a video containing human actions to the system for detection.
   * Action: The system receives the video and prepares it for processing.
3. Detect Actions: The system analyses the uploaded video to detect human actions.
   * Action: The system processes the video frames, extracts features, and detects specific actions such as fighting, car crash and punching etc.
4. Store Video: The system stores the uploaded video for future reference or further processing.
   * Action: The video file is saved in the database or storage system to keep track of uploaded videos.
5. Store Metadata: Along with storing the video, metadata such as the detection results, timestamp, and other relevant details are stored.
   * Action: The system logs essential metadata related to the video and detection results for future retrieval.
6. Display Detected Actions: The detected actions are shown to the user, including bounding boxes around the actions and corresponding confidence scores.
   * Action: The system presents a visualization of the detected actions, providing details like type of action, confidence score, and the corresponding frames where the action is detected.

**Description of the Flow**:

* The User uploads a video via the interface (Upload Video), which is then processed by the system to detect human actions (Detect Actions). The detected actions are displayed (Display Detected Actions), and metadata (such as the action labels and confidence scores) are stored (Store Metadata). Previous results can be retrieved (Fetch Previous Results) if needed. Additionally, the system stores the video for future use (Store Video).

**3.2.2 Sequence Diagram:**

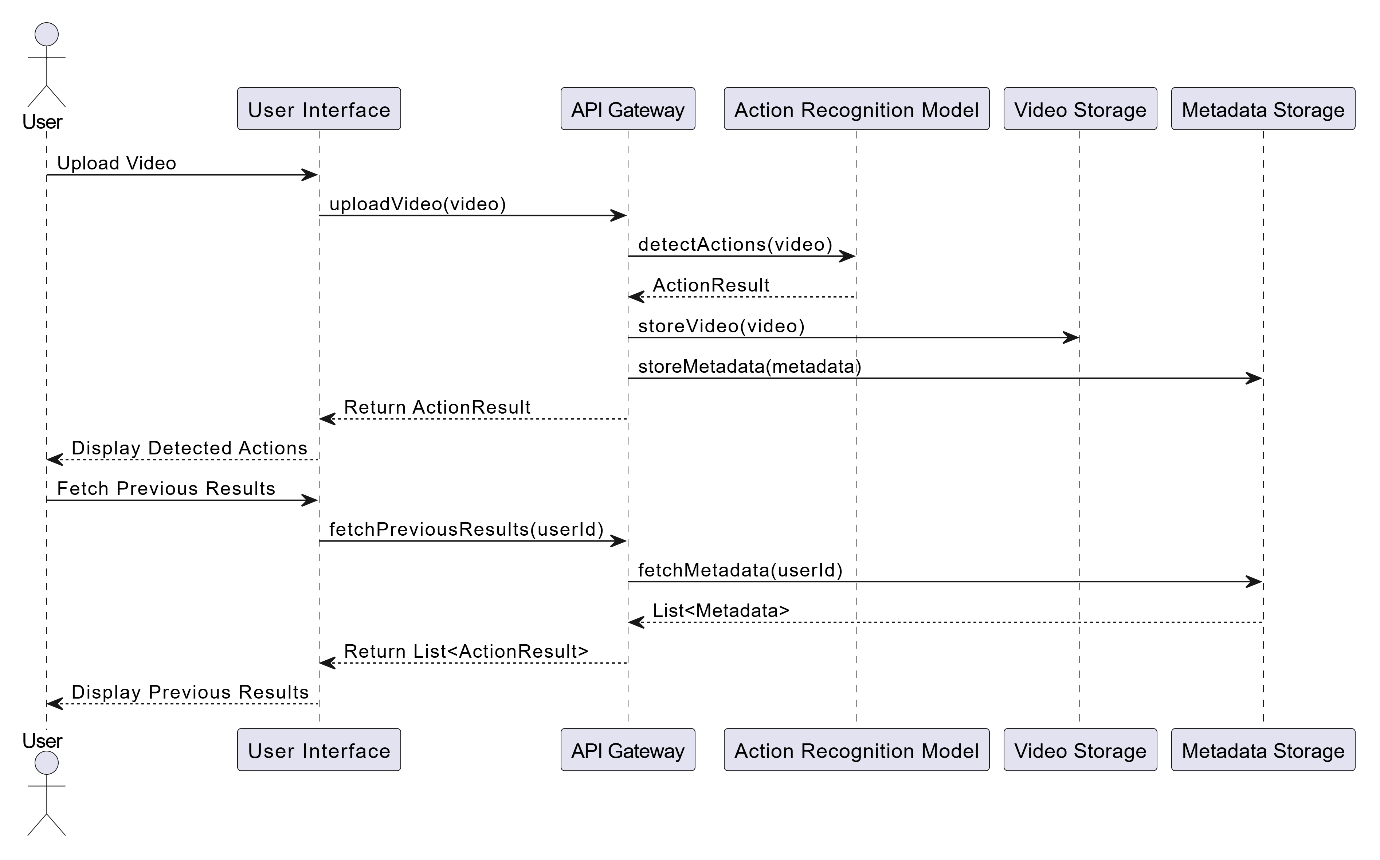


Fig 2: Sequence Diagram for Action Recognition system

A sequence diagram is a type of interaction diagram in UML (Unified Modelling Language) that illustrates the interactions between objects or components in a specific scenario or use case. It represents the flow of messages, actions, and events over time, showing the order in which, these interactions occur.

* Outlines process flow from user interface to data collection, data preprocessing, ML model prediction, and output of label.
* Sequence diagram consists of 4 different blocks namely user interface, Data collection , Preprocess Data and Model as shown in the above figure
* User will provide the input image through the file’s already saved image is being taken in consideration which is been captured and sent to the processor where pre- processing of data is done which is resizing, reshaping and other parameters and after that those are stored in the memory unit.
* After pre-processing and storing of image, cnn trained model file is loaded where the featured of the image is extracted for classifying the output. After classifying the output, label is provided.

**Sequence Diagram Description for Action Recognition System**

**Components Involved:**

1. User Interface (UI): The frontend interface that allows the user to upload videos and view results.
2. API Gateway: The intermediary that receives requests from the UI and forwards them to the appropriate services.
3. Action Recognition Model: The deep learning model responsible for analyzing the video and detecting actions.
4. Video Storage: The component where uploaded videos are stored for future reference or processing.
5. Metadata Storage: The database where metadata (such as detected actions and timestamps) is stored for retrieval.

**Sequence of Interactions**:

1. User Uploads Video (User Interface → API Gateway):

* The user uploads a video file via the User Interface.
* The User Interface sends the video data to the API Gateway for processing.

2. API Gateway Sends Video to Action Recognition Model (API Gateway → Action Recognition Model):

* The API Gateway forwards the uploaded video to the Action Recognition Model for action detection.
* The Action Recognition Model processes the video and identifies actions.

3. Store Detected Actions (Action Recognition Model → Video Storage and Metadata Storage):

* After processing the video, the Action Recognition Model stores the video in the Video Storage.
* Additionally, the detected actions, metadata, and results (such as timestamps, action labels, confidence scores) are stored in Metadata Storage.

4. Results are Sent Back to User Interface (API Gateway → User Interface):

* The API Gateway retrieves the detected actions and metadata from the Action Recognition Model and forwards them to the User Interface.
* The User Interface displays the results, showing the detected actions, confidence scores, and any other relevant information.

5. User Requests Previous Results (User Interface → API Gateway → Metadata Storage):

* If the user requests to view previous results, the User Interface sends the request to the API Gateway.
* The API Gateway retrieves the relevant data from Metadata Storage, and it sends the results back to the User Interface.

6. Display Previous Results (User Interface):

* The User Interface displays the fetched results from Metadata Storage, allowing the user to review past action detections.

**3.2.3 Activity Diagram:**

**A diagram of a process

Description automatically generated**

Fig 3: Activity Diagram for Action Recognition system

An activity diagram is a type of UML (Unified Modelling Language) diagram that visually represents the flow and sequence of activities or actions within a system, process, or business workflow. It is a dynamic diagram that focuses on the behaviour of a system over time. Activity diagrams are particularly useful for modelling the workflow of business processes, software applications, and complex system interactions.

We use Activity Diagrams to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case. We model sequential and concurrent activities using activity diagrams. So, we basically depict workflows visually using an activity diagram.

An activity diagram focuses on condition of flow and the sequence in which it happens. We describe or depict what causes a particular event using an activity diagram. UML models basically three types of diagrams, namely, structure diagrams, interaction diagrams, and behavior diagrams. An activity diagram is a behavioral diagram i.e., it depicts the behavior of a system. An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed. We can depict both sequential processing and concurrent processing of activities using an activity diagram. They are used in business and process modelling where their primary use is to depict the dynamic aspects of a system. An activity diagram is very similar to a flowchart.

**Workflow of the System:**

1. User Uploads Video:
   * The process begins when the user uploads a video file through the User Interface.
   * The video file is passed on to the API Gateway for further processing.
2. API Gateway and Action Recognition Model Processing:
   * The API Gateway forwards the uploaded video to the Action Recognition Model.
   * The Action Recognition Model processes the video, detecting human actions in the video frames.
3. Detect Actions:
   * As the Action Recognition Model detects actions in the video, it generates metadata such as action labels, timestamps, and confidence scores.
4. Store Video and Metadata:
   * After processing the video, the video is stored in Video Storage for future reference.
   * The detected actions and metadata (e.g., timestamps, action labels) are stored in Metadata Storage.
5. Results Sent to User Interface:
   * The API Gateway sends the detected actions and metadata to the User Interface.
   * The User Interface displays the detected actions and related information such as confidence scores to the user.
6. User Requests Previous Results:
   * If the user wants to view previous results, they request it via the User Interface.
   * The User Interface sends the request to the API Gateway.
7. Fetch Previous Results from Metadata Storage:
   * The API Gateway retrieves the required metadata from Metadata Storage.
   * The relevant past action detection results are fetched.
8. Display Previous Results:
   * The User Interface displays the previously fetched results for the user to review.

**3.2.4 System architecture:**

A diagram of a software development

Description automatically generated

Fig 3: System architecture for Action Recognition system

The System Architecture Diagram provides a high-level view of the system's structure and the interactions between its key components. Here’s a breakdown of the main components and their roles:

1. User Interface (UI): The UI acts as the point of interaction for users. It allows users to upload videos and fetch previous results. Once the user uploads a video, the UI communicates with the API Gateway to process the video. Additionally, it displays the detected actions or any previously stored results, providing a seamless user experience.
2. API Gateway: The API Gateway serves as the central communication hub in the system. It handles requests from the UI, processes the uploaded videos, and forwards them to the Action Recognition Model for action detection. After processing, it stores the video in Video Storage and saves relevant metadata in Metadata Storage. The API Gateway also retrieves stored results when the user requests them.
3. Action Recognition Model: This is the core processing unit of the system. It takes the video frames, analyses them for human actions, and generates predictions based on the learned model. It then sends the detected actions to the API Gateway, which passes the results to the UI for display. The model also stores the results in metadata for future retrieval.
4. Storage: The Video Storage component manages the actual video files. It stores uploaded videos and makes them available for future analysis or reference. This ensures that videos are preserved securely and can be reprocessed if necessary.
5. Metadata Storage: The Metadata Storage stores critical information related to each processed video, such as detected actions, timestamps, accuracy, and confidence scores. This metadata is essential for providing feedback to the user, and it also supports fetching previous results when required.

This architecture ensures that each component is modular, scalable, and can efficiently handle the process of video analysis, storage, and retrieval of action detection results. It supports the overall goal of providing real-time action recognition and allows for easy management of historical data.

**3.3 Functional Requirements**

**3.3.1 Software Requirements:**

The software requirements, also known as general description papers, also include information about the product’s perspective and features, operating system and operating environment, graphic needs, design constraints, and user documentation. These requirements show us the necessary software functionalities and performance criteria that need to be met to ensure the successful completion of the project.

* **Operating System: -** Windows 10 or more
* **Technology: -** Python, Machine Learning, Deep Learning, Django
* **Software: -** Jupyter or VS code or PyCharm

**3.3.2 Hardware Requirements:**

The minimum hardware requirements necessitate a modern and capable computer system to handle the computational demands of ML algorithms and data processing effectively. The hardware requirements include a processor that can efficiently execute computations, an ample amount of RAM to store and manipulate large datasets during training and inference, sufficient storage space to accommodate datasets, ML models, and related files

* **Processor:** intel i3 or higher
* **Operating system:** windows
* **Ram & Hard disk:** At least 4 GB & at least 100gb

**CHAPTER 4**

**IMPLEMENTATION**

**4.1 Steps Description**

1. **Preprocessing Step (Extracting Frames and Normalization):** In the preprocessing phase, we extract frames from video files, resize them to a fixed size (64x64), and normalize the pixel values by dividing by 255. This ensures that the pixel values are scaled between 0 and 1, which is essential for the model to converge efficiently. Here's the code for preprocessing the video frames and normalizing them:

****Fig 5: Extracting Frames and Normalization

1. **Data Conversion (One-hot Encoding):** After preprocessing, the class labels for each video are converted to one-hot encoding. This ensures the labels are in a format suitable for training a multi-class classification model. One-hot encoding creates a binary vector where only the index corresponding to the class is set to 1, and all others are 0. Here’s the code for one-hot encoding:

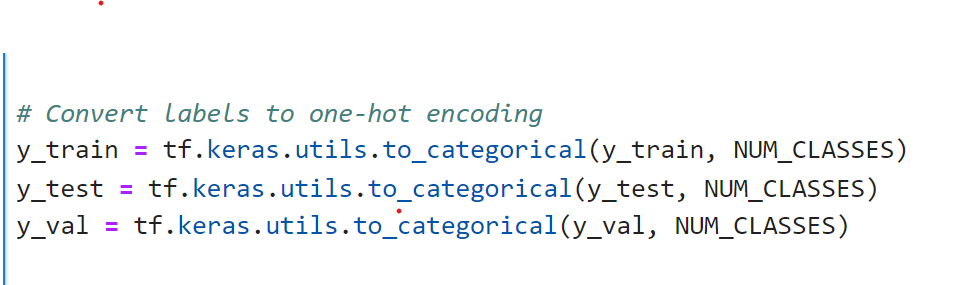
****

Fig 6: One-hot Encoding

1. **Model Architecture (CNN + LSTM):** The model architecture consists of 3D Convolutional Neural Networks (CNN) layers for spatial and temporal feature extraction, followed by an LSTM layer to capture long-term dependencies in the video sequence. The model ends with a Dense layer for classification. Here's the code for building the model:

**A screen shot of a computer code

Description automatically generated**Fig 7: 3DCNN and LSTM model

1. **Model Training:** The model is compiled using the Adam optimizer with a learning rate of 0.001 and categorical cross-entropy loss. During training, we use callbacks like early stopping (to prevent overfitting) and model checkpointing (to save the best model weights). Here's the code for training the model:

**A screenshot of a computer code

Description automatically generated**Fig 8: Model Training

1. **Model Evaluation:** Once the model is trained, we evaluate its performance using the test dataset. We calculate the predicted labels, compare them with the true labels, and generate a classification report and confusion matrix. Here's the code for model evaluation:

**A computer screen shot of a code

Description automatically generated**Fig 9: Model Evaluation

1. **Model Saving:** After training and evaluating the model, we save the best version of the model to a file (in this case, m.keras). This saved model can later be reloaded and used for inference on new, unseen data without retraining. Here's the code for saving the model:

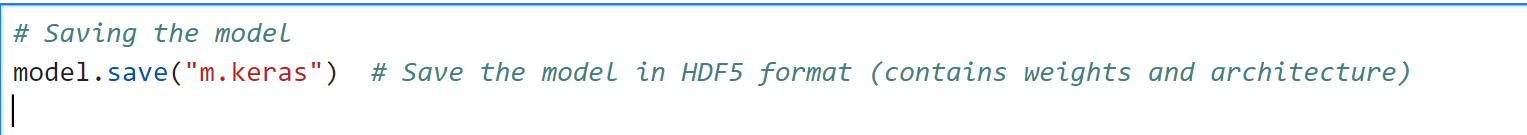
****

Fig 10: Model Saving

**4.2 Dataset**

**4.2.1 Dataset Description**

**i. Training Set:**

* Located in the "train" folder.
* Contains subfolders of human actions videos.
* Where each folder is labelled with respective human action

**ii. Testing Set:**

* Found in the "test" folder.
* Similar to the training set, it comprises subfolders for human actions videos.
* Where each folder is labelled with respective human action

**iii. Validation Set:**

* Located in the "validation" folder.
* Similar to the training set, it comprises subfolders for human actions videos.
* Where each folder is labelled with respective human action

Volume serial number is F4E8-16E7

C:.

├───test

│ ├───carcrash

│ ├───fighting

│ ├───Punch

│ ├───WalkingWithDog…..

├───train

│ ├───human actions…

└───val

├───human actions ..

This dataset organization adheres to the standard practice of separating data into training, testing, and validation sets, ensuring the development of a robust and effective counterfeit currency detection model.

**4.3 Sample Code**

**4.3.1 Backend**

import cv2

import numpy as np

from django.shortcuts import render, redirect, get\_object\_or\_404

from django.http import JsonResponse

from .forms import VideoUploadForm

from .models import Video, Metadata

from .human\_action\_model import model, CLASSES

def extract\_frames(video\_path, img\_size, sequence\_length):

    frames\_list = []

    video\_reader = cv2.VideoCapture(video\_path)

    if not video\_reader.isOpened():

        print(f"Failed to open video: {video\_path}")

        return frames\_list

    video\_frames\_count = int(video\_reader.get(cv2.CAP\_PROP\_FRAME\_COUNT))

    skip\_frames\_window = max(int(video\_frames\_count / sequence\_length), 1)

    for frame\_counter in range(sequence\_length):

       video\_reader.set(cv2.CAP\_PROP\_POS\_FRAMES,frame\_counter\* skip\_frames\_window)

      success, frame = video\_reader.read()

if not success:

break

        resized\_frame = cv2.resize(frame, (img\_size, img\_size))

        normalized\_frame = resized\_frame.astype('float32') / 255.0

        frames\_list.append(normalized\_frame)

    video\_reader.release()

    return frames\_list

def predict\_action(frame\_sequence, model, CLASSES):

    frame\_sequence = np.expand\_dims(frame\_sequence, axis=0)

    pred = model.predict(frame\_sequence)

    action\_idx = np.argmax(pred)

    confidence = float(pred[0][action\_idx])

    return CLASSES[action\_idx], confidence

def index(request):

    if request.method == 'POST':

        form = VideoUploadForm(request.POST, request.FILES)

        if form.is\_valid():

            video = form.save()

            video\_path = video.video\_file.path

            # Extract frames and predict action

            frames = extract\_frames(video\_path, img\_size=64, sequence\_length=40)

            if frames:

                detected\_action, confidence = predict\_action(np.array(frames), model, CLASSES)

                Metadata.objects.create(

                    video=video,

                    detected\_actions=[detected\_action],

                    confidence\_scores=[confidence]

                )

            else:

                Metadata.objects.create(

                    video=video,

                    detected\_actions=["No action detected"],

                    confidence\_scores=[0.0]

                )

            return redirect('results', video\_id=video.id)

    else:

        form = VideoUploadForm()

    return render(request, 'recognition/index.html', {'form': form})

def results(request, video\_id):

    video = get\_object\_or\_404(Video, id=video\_id)

    metadata = get\_object\_or\_404(Metadata, video=video)

    return render(request, 'recognition/results.html', {'video': video, 'metadata': metadata})

def fetch\_previous\_results(request):

    if request.method == 'GET':

        videos = Video.objects.all().order\_by('-uploaded\_at')

        results = []

        for video in videos:

            metadata = Metadata.objects.filter(video=video).first()

    if metadata:

                results.append({

                    'video\_id': video.id,

                    'video\_url': video.video\_file.url,

                    'detected\_actions': metadata.detected\_actions,

                    'confidence\_scores': metadata.confidence\_scores,

                    'timestamp': metadata.timestamp

                })

        return JsonResponse(results, safe=False)

**4.2.2 Frontend**

Templates/Index.html

<!DOCTYPE html>

{% load static %}

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Action Recognition</title>

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet">

    <link rel="stylesheet" href="{% static 'css/styles.css' %}">

</head>

<body>

    <div class="container mt-5">

        <h1 class="text-center">Action Recognition System</h1>

        <div class="row mt-4">

            <div class="col-md-6">

                <h2>Upload Video</h2>

                <form method="post" enctype="multipart/form-data">

                    {% csrf\_token %}

                    {{ form.as\_p }}

                    <button type="submit" class="btn btn-primary">Upload</button>

                </form>

            </div>

            <div class="col-md-6">

                <h2>Previous Results</h2>

                <button id="fetch-results" class="btn btn-secondary">Fetch Previous Results</button>

                <ul id="results-list" class="mt-3"></ul>

            </div>

        </div>

    </div>

    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

    <script src="{% static 'js/main.js' %}"></script>

</body>

</html>

**CHAPTER 5**

**TESTING**

**5.1 Importance of Testing**

Testing is a crucial phase in the development lifecycle that holds paramount importance for ensuring the reliability, functionality, and overall quality of a system or application. It involves systematically evaluating various aspects, such as accuracy, robustness, and user experience, to detect and rectify defects, bugs, or vulnerabilities. Testing not only verifies that the software or model meets specified requirements but also contributes to the prevention of costly errors, the optimization of performance, and the establishment of user trust. Thorough testing provides a systematic and evidence-based approach to validating the system's capabilities, identifying potential risks, and enhancing the overall quality of the end product, whether it be software, a machine learning model, or any computational system.

1. **Accuracy Verification:**

* Testing is crucial for verifying the accuracy of the counterfeit currency detection system. Rigorous testing ensures that the system can effectively differentiate between genuine and fake currency, reducing the risk of false positives or negatives that could have significant financial implications.

1. **Robustness against Variability:**

* Counterfeit currency may exhibit a wide range of variations in terms of printing quality, paper texture, and other features. Testing allows the system to be robust against such variability, ensuring consistent and reliable performance across different instances of counterfeit notes.

1. **Adaptability to New Threats:**

* The landscape of counterfeit currency is dynamic, with new techniques and technologies emerging over time. Testing enables the system to adapt to evolving threats, ensuring that it remains effective in detecting the latest counterfeit methods and security features.

1. **User Trust and Confidence:**

* Thorough testing instills trust and confidence in users, whether they are financial institutions, businesses, or individuals. Knowing that the counterfeit currency detection system has undergone comprehensive testing builds credibility and encourages widespread adoption.

1. **Mitigation of False Alarms:**

* Testing helps identify and address scenarios that may lead to false alarms. By refining the detection algorithm and minimizing false positives, the system becomes more reliable and user-friendly, reducing unnecessary disruptions and alerts.

1. **Compliance with Regulations:**

* In the financial sector, compliance with regulatory standards is paramount. Testing ensures that the counterfeit currency detection system adheres to relevant regulations, safeguarding against legal and compliance-related risks.

Top of Form

**5.2 Types of Testing**

Testing is a comprehensive process, and various types of testing are employed to ensure the quality, reliability, and functionality of software, systems, or applications. Here are some key types of testing:

1. **Unit Testing:**

* Involves testing individual units or components of a software independently to ensure they function correctly. Developers often perform unit testing during the development phase.

1. **Integration Testing:**

* Focuses on verifying the interactions and interfaces between integrated components or systems. It ensures that different modules work together seamlessly.

1. **Functional Testing:**

* Verifies that the software functions according to specified requirements. It involves testing the system's features, capabilities, and user interactions.

1. **Non-Functional Testing:**

* Focuses on non-functional aspects such as performance, usability, reliability, and security. Examples include performance testing, usability testing, and security testing.

1. **Regression Testing:**

* Ensures that new changes or updates to the software do not negatively impact existing functionalities. It involves retesting previously tested features.

1. **User Acceptance Testing (UAT):**

* Involves testing the software from the end user's perspective to ensure that it meets their requirements and expectations. Users typically perform this testing.

1. **System Testing:**

* Evaluates the complete and integrated system to ensure that it behaves as intended. It assesses the system's compliance with specified requirements.

1. **Performance Testing:**

* Evaluates the system's performance, responsiveness, and stability under various conditions, such as load testing, stress testing, and scalability testing.

1. **Usability Testing:**

* Assesses the software's user interface and overall user experience. It ensures that the system is user-friendly and meets user expectations.

1. **Security Testing:**

* Identifies vulnerabilities and weaknesses in the software's security features. It involves testing for potential threats and ensuring data protection.

1. **Compatibility Testing:**

* Verifies that the software functions correctly across different platforms, browsers, devices, and operating systems.

1. **Exploratory Testing:**

* Involves ad-hoc testing where testers explore the software to discover defects without predefined test cases. It is often used to identify unexpected issues.

1. **Beta Testing:**

* Involves releasing a version of the software to a limited group of users for testing in a real-world environment. It helps gather user feedback before the official release.

1. **Alpha Testing:**

* Conducted by internal teams before beta testing. It aims to identify issues within the software before it is made available to a wider audience.

1. **Ad-hoc Testing:**

* Informal testing without predefined test cases. Testers use their experience, creativity, and domain knowledge to identify defects.

1. **Load Testing:**

* Assesses how the system performs under expected and peak loads. It helps ensure that the software can handle a specific number of concurrent users or transactions.

1. **White Box Testing:**

* Examines the internal logic, structure, and code of the software. Testers have knowledge of the internal workings of the system.

1. **Black Box Testing:**

* Focuses on testing the software's functionalities without knowledge of its internal code or logic. Testers assess the system based on input and output.

These testing types can be combined or customized based on the specific needs of a project to achieve a comprehensive testing strategy.

**5.2.1 Functional Testing**

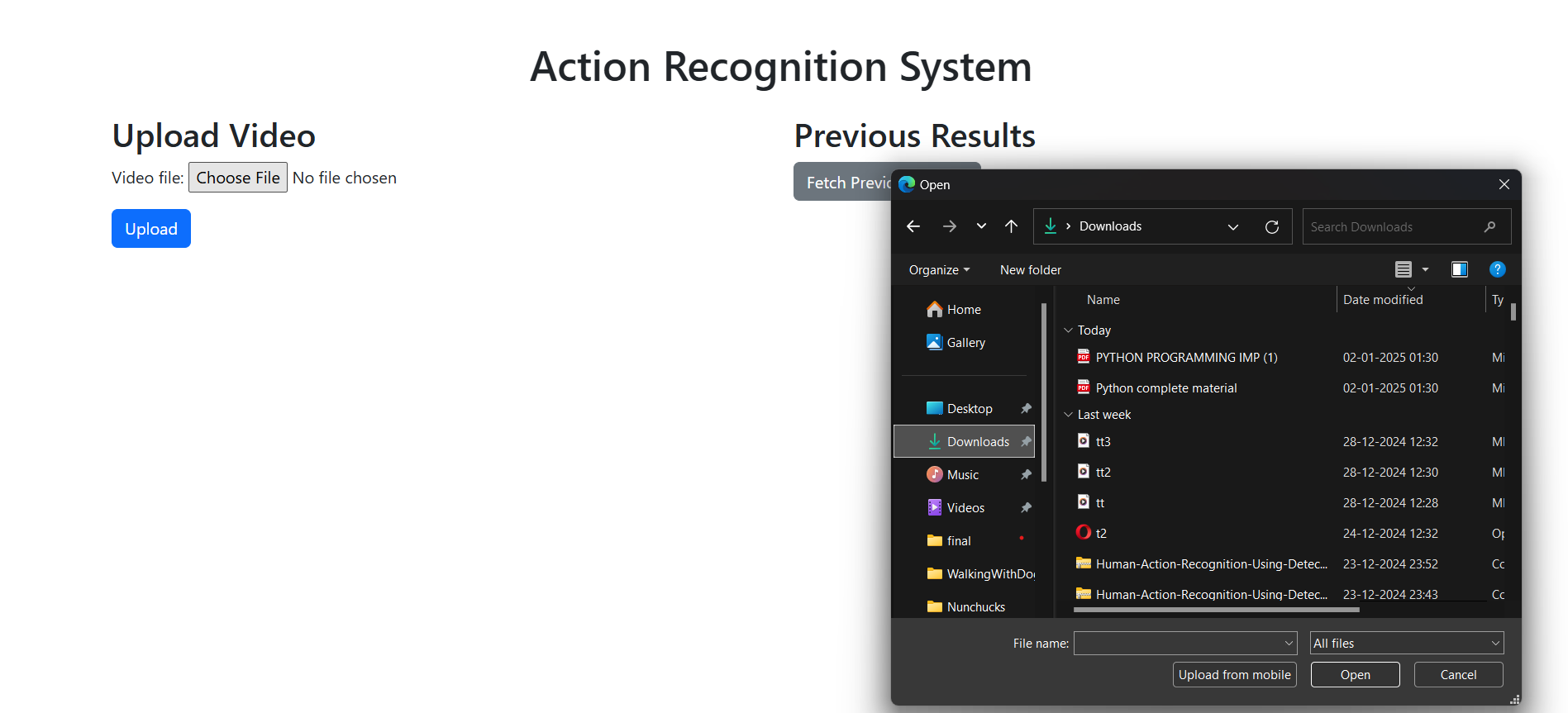
Testers follow the following steps in the functional testing:

* Tester does verification of the requirement specification in the software application.
* After analysis, the requirement specification tester will make a plan.
* After planning the tests, the tester will design the test case.
* After designing the test, case tester will make a document of the traceability matrix.
* The tester will execute the test case design.
* Analysis of the coverage to examine the covered testing area of the application.
* Defect management should do to manage defect resolving.

**CHAPTER 6**

**RESULTS**

The user interface provides a straightforward and efficient platform for uploading videos. It allows users to browse and select video files from their local devices, which are then submitted to the system for action detection. The interface is designed to be intuitive, ensuring accessibility for users without requiring technical expertise. Screenshots of the user interface illustrate its clean and functional design, showcasing the ease with which users can interact with the system.

Fig 11: User Interface for Video Upload

The system effectively processes uploaded videos and identifies human actions using its hybrid 3D-CNN and LSTM model. After analysis, the detected actions are displayed with corresponding confidence scores, such as "Walking with dog: 95% confidence" or "Fighting: 99% confidence." These results provide users with clear insights into the activities present in the video. The system demonstrates robust performance across various scenarios, including different lighting conditions and levels of motion complexity. Sample outputs from the analysis are presented as screenshots, highlighting the system’s ability to deliver precise and interpretable results.

A screen shot of a computer

Description automatically generatedFig 12: Results of Action Detection

The system also supports the retrieval of historical results through its metadata storage feature. Users can access previously analyzed videos and review details such as action labels, timestamps, and confidence scores. This feature is particularly useful for tracking patterns or revisiting past analyses. The historical results are displayed in an organized format, ensuring clarity and ease of navigation. Screenshots demonstrate how users can effectively access and interpret previously stored data, enhancing the system’s practical usability.

A screenshot of a computer

Description automatically generatedFig 13: Retrieving Previous Results

**CHAPTER 7**

**CONCLUSION AND FUTURE SCOPE**

This project demonstrates the successful implementation of a human action recognition system that combines 3D-CNN and LSTM technologies. By leveraging spatial and temporal feature extraction, the system provides accurate and robust detection of actions such as walking with dog, fighting, and car crash in real-time scenarios. Its user-friendly interface and historical data retrieval features enhance its practicality, making it suitable for applications in public surveillance, smart city infrastructure, and behavioral analysis. The project’s results highlight its capability to address the challenges of action recognition in dynamic environments effectively.

The system offers a solid foundation for further development. Future enhancements could include the integration of additional features such as:

* Crowded Scene Detection: Improving the system’s ability to handle multiple individuals performing simultaneous actions.
* Fine-grained Action Recognition: Enabling detection of more subtle or overlapping actions.
* Hardware Optimization: Incorporating model compression techniques and hardware acceleration to enhance real-time performance.
* IoT Integration: Leveraging data from IoT devices for richer contextual analysis of human actions.
* Predictive Analytics: Developing capabilities to predict potential risks based on detected behaviors.
* Adaptability: Scaling the system to handle larger datasets and diverse environmental conditions, ensuring continued effectiveness in evolving scenarios.

These advancements would extend the system’s applicability to a broader range of real-world situations, ensuring its relevance in future deployments.

**CHAPTER 8**

**REFERENCES**

1. N. A. Ravuri, "A Systematic Literature Review on Human Activity Recognition," *Deleted Journal*, vol. 20, no. 6s, pp. 1175–1191, 2024, doi: 10.52783/jes.2848. Available: <https://doi.org/10.52783/jes.2848>
2. "Human activity recognition using deep learning," IEEE Conference Publication, IEEE Xplore, April 14, 2022. Available: <https://ieeexplore.ieee.org/document/9786890/>
3. A. Sánchez-Caballero, D. Fuentes-Jiménez, and C. Losada-Gutiérrez, "Real-time human action recognition using raw depth video-based recurrent neural networks," *Multimedia Tools and Applications*, vol. 82, no. 11, pp. 16213–16235, 2022, doi: 10.1007/s11042-022-14075-5. Available: <https://doi.org/10.1007/s11042-022-14075-5>
4. C. Yang, S. Hsu, Y. Hsu, and Y. Kang, "HAR-time: human action recognition with time factor analysis on worker operating time," *Int. J. Computer Integrated Manufacturing*, vol. 36, no. 8, pp. 1219–1237, 2023, doi: 10.1080/0951192x.2023.2177736. Available: <https://doi.org/10.1080/0951192x.2023.2177736>
5. S. Khan, M. A. Khan, M. Alhaisoni, U. Tariq, H. Yong, A. Armghan, and F. Alenezi, "Human Action recognition: a paradigm of best deep learning features selection and serial based extended fusion," *Sensors*, vol. 21, no. 23, p. 7941, 2021, doi: 10.3390/s21237941. Available: <https://doi.org/10.3390/s21237941>
6. M. Karim, S. Khalid, A. Aleryani, J. Khan, I. Ullah, and Z. Ali, "Human Action Recognition Systems: A Review of the Trends and State-of-the-Art," *IEEE Access*, vol. 12, pp. 36372–36390, 2024, doi: 10.1109/access.2024.3373199. Available: <https://doi.org/10.1109/access.2024.3373199>
7. M. G. Morshed, T. Sultana, A. Alam, and Y. Lee, "Human Action Recognition: A Taxonomy-Based Survey, updates, and opportunities," *Sensors*, vol. 23, no. 4, p. 2182, 2023, doi: 10.3390/s23042182. Available: <https://doi.org/10.3390/s23042182>
8. V. Sarveshwaran, I. T. Joseph, M. M., and P. K., "Investigation on Human Activity Recognition using Deep Learning," *Procedia Computer Science*, vol. 204, pp. 73–80, 2022, doi: 10.1016/j.procs.2022.08.009. Available: <https://doi.org/10.1016/j.procs.2022.08.009>
9. H. H. Pham, L. Khoudour, A. Crouzil, P. Zegers, and S. A. Velastin, "Video-based Human Action Recognition using Deep Learning: A Review," *arXiv*, 2022. Available: <https://doi.org/10.48550/arxiv.2208.03775>
10. Z. Wu and H. Du, "Research on Human Action feature detection and recognition algorithm based on deep learning," *Mobile Information Systems*, vol. 2022, pp. 1–12, 2022, doi: 10.1155/2022/4652946. Available: <https://doi.org/10.1155/2022/4652946>