A PROJECT REPORT

on

"HUMAN POSE ESTIMATION AND DANGER DETECTION"

Submitted to

KIIT Deemed to be University

BY

RISHIKA PAL 22051876

UPAL PAHARI 22052256

SAUVATRA PAUL 22052236

SRINJOY KUNDU 22051898

UNDER THE GUIDANCE
OF
DR. RAJDEEP CHATTERJEE

SCHOOL OF COMPUTER ENGINEERING
KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY
BHUBANESWAR, ODISHA - 751024
April 2025

KIIT Deemed to be University

School of Computer Engineering Bhubaneswar, ODISHA
751024

CERTIFICATE

This is certify that the project entitled

"HUMAN POSE ESTIMATION AND DANGER DETECTION"

BY

RISHIKA PAL 22051876

UPAL PAHARI 22052256

SAUVATRA PAUL 22052236

SRINJOY KUNDU 22051898

is a record of bonafide work carried out by them, in the partial fulfillment of the requirement for the award of Degree of Bachelor of Engineering (Computer Science and Engineering) at KIIT Deemed to be university, Bhubaneswar. This work is done during year 2024-2025 , under our guidance.

Date: 03/04/2025

DR. RAJDEEP CHATTERJEE
(Project Guide)

Acknowledgement

We are profoundly grateful to Dr. Rajdeep Chatterjee for his expert guidance and constant support throughout, without which the completion of this project would not have been possible. His continuous encouragement has helped us in completing the project in the provided time.

RISHIKA PAL
UPAL PAHARI
SAUVATRA PAL
SRINJOY KUNDU

ABSTRACT

Human Pose Estimation is crucial in computer vision for understanding human movement in applications like activity recognition, human-computer interaction, and surveillance. This project aims to detect dangerous actions—such as punching, kicking, and shooting—using a combination of LSTM networks, MediaPipe, and TensorFlow. A custom dataset was built by extracting pose landmarks from video sequences using MediaPipe, capturing key human body points to analyze temporal motion patterns. These sequences were then used to train an LSTM-based model, which is well-suited for recognizing sequential and time-dependent data.

The model utilizes Softmax activation for classifying multiple dangerous actions with high accuracy. OpenCV was employed to handle real-time video processing tasks such as frame extraction and playback control, while TensorFlow enabled the training and optimization of the model. The resulting system performs effectively in recognizing violent behavior and can be applied in real-world safety monitoring environments. Future improvements include edge computing deployment for reduced latency and enhanced robustness using multi-modal sensing or more diverse datasets.

Keywords: MediaPipe, dataset, LSTM, Softmax, OpenCV, scalable, violent behaviour, safety.

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview of Human Activity Recognition

Human activity recognition (HAR) is an essential field in computer vision and artificial intelligence that focuses on identifying and classifying human actions based on visual and sensor-based data. With the increasing deployment of surveillance systems, security agencies require intelligent algorithms capable of detecting potentially dangerous activities in real-time. Traditional HAR techniques primarily rely on handcrafted features, which often fail to generalize well across different environments and human movements.

1.2 Importance in Security & Surveillance

The ability to distinguish between dangerous and non-dangerous activities is crucial for applications such as:

Public Safety: Detecting violent incidents in public places.

Industrial Safety: Monitoring workers to prevent accidents.

Home Security: Identifying break-ins or suspicious movements.

Advanced deep learning methods, particularly those based on pose estimation and sequential learning, have shown significant promise in improving the accuracy of HAR systems. By leveraging MediaPipe Pose for extracting human skeletal landmarks and Long Short-Term Memory (LSTM) networks for sequence modeling, this project aims to develop an efficient system for danger activity recognition.

1.3 Challenges in Recognizing Threatening Actions

While traditional computer vision-based HAR models perform well in constrained environments, they struggle with:

Occlusion & Partial Visibility: Actions may be obstructed due to objects or camera angles.

Variation in Human Poses: The same activity (e.g., fighting) can have different poses across individuals.

Generalization Across Environments: HAR models need to adapt to different lighting conditions, clothing variations, and backgrounds.

1.4 Objectives of this Project

This project aims to address these challenges by developing a pose-based danger detection model with the following objectives:

- Extract 33 key skeletal landmarks from human body movements using MediaPipe Pose.
- Structure pose key points into time-series sequences for action recognition.
- Train a Bidirectional LSTM model to classify actions as Danger (e.g., fighting, kicking, punching, shoving) vs. Non-Danger (e.g., walking, sitting, jumping).
- Deploy the trained model to analyze real-time video feeds and classify activities accordingly.

Chapter 2

Literature Survey

Human pose estimation and activity recognition have been extensively studied for applications in security and surveillance. Many researchers have employed deep learning techniques, such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs), to improve accuracy and real-time detection capabilities. Toshev and Szegedy (2014) introduced DeepPose, which used deep learning to estimate human body joints with high accuracy. This approach inspired subsequent research into using deep architectures for pose estimation. OpenPose (Cao et al., 2017) further advanced the field by introducing a real-time multi-person pose estimation framework, allowing the extraction of skeletal key points from images and videos. More recent research integrates MediaPipe Pose (Lugaresi et al., 2019), which provides a lightweight yet efficient solution for detecting 33 key skeletal landmarks in real-time. These advancements in pose estimation have enabled their integration into action recognition models. Long Short-Term Memory (LSTM) networks, a variant of RNNs, have been widely used for sequential data modeling, making them particularly useful for human activity recognition (HAR). Graves et al. (2013) demonstrated the power of LSTMs in time-series forecasting, which has been further applied to video-based action recognition. Bidirectional LSTMs, which process sequences in both forward and backward directions, enhance the contextual understanding of movements, making them well-suited for danger detection in surveillance videos.

Chapter 3

3.1 Proposed Work

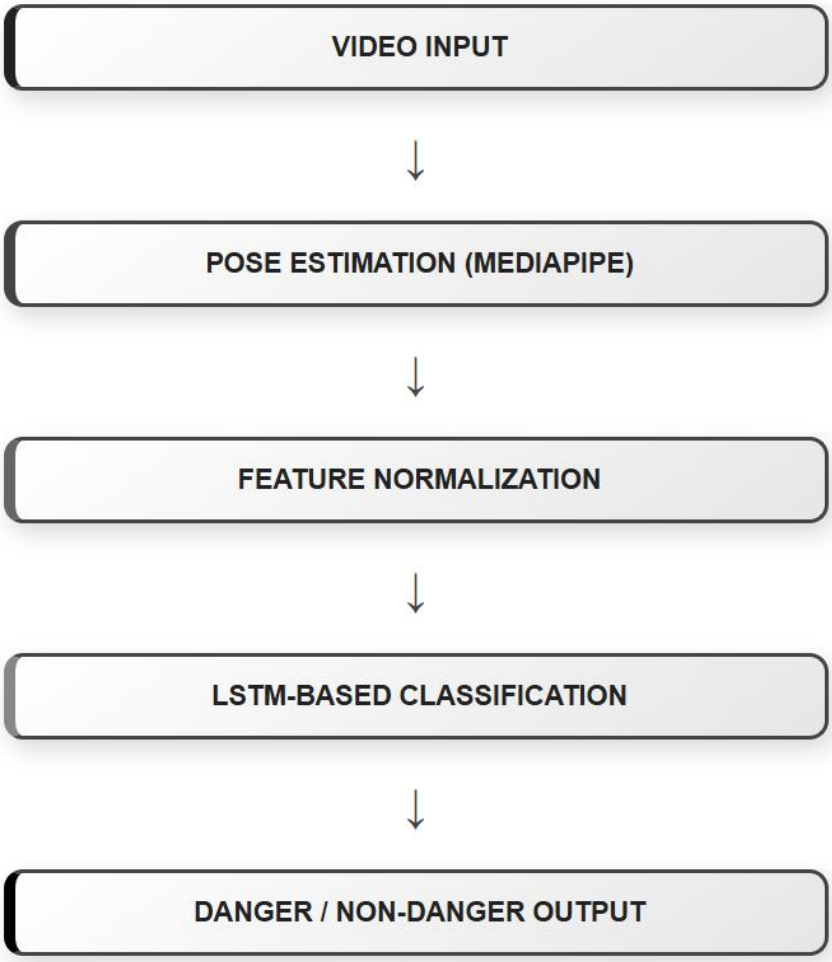

Fig 3.1.1 Workflow Diagram

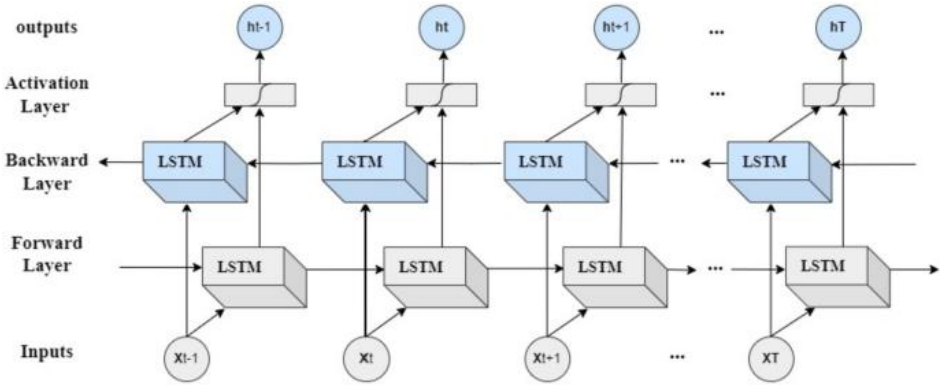

Fig 3.1.2 LSTM Workflow Diagram

3.2 Dataset

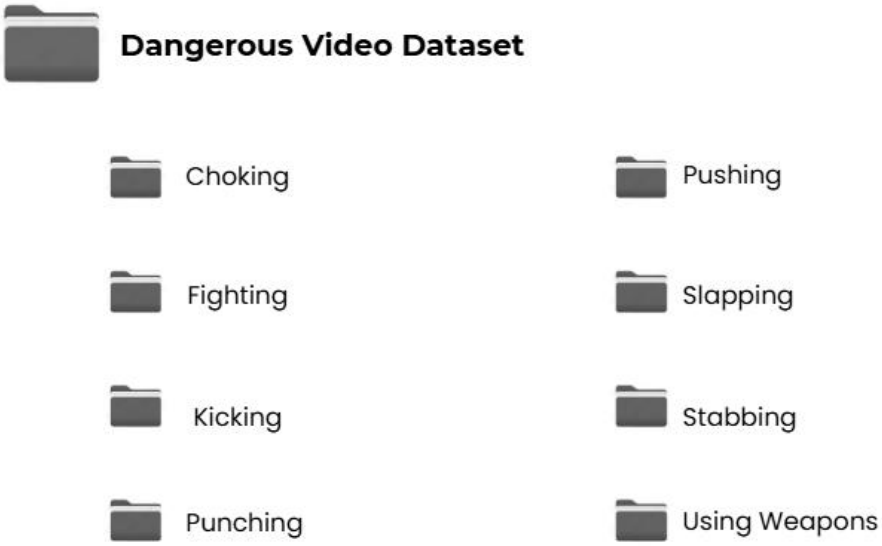

Fig 3.2.1 Dangerous Video Dataset

Fig 3.2.2 Non-Dangerous Video Dataset

3.3 Models

In this report we discuss a few a few techniques that utilizes historical data from datasets to train simple machine learning models, which are essential for the project.

The models we discuss here are:

1.MediaPipe
2.LSTM
3.Tensorflow
4.Softmax

Chapter 4

Implementation

4.1 Pose Landmarks Extraction

4.1.1 Libraries Used

OpenCV, MediaPipe, NumPy, Pandas, OS

4.1.2 Pose Detector Class

```
class PoseDetector():
    def __init__(self, detectionCon=0.5, trackCon=0.5):
        self.mpPose = mp.solutions.pose
        self.pose = self.mpPose.Pose(
            min_detection_confidence=detectionCon, min_tracking_confidence=trackCon)
```

Initializes MediaPipe Pose with:

- Min_detection_confidence (default 0.5): Confidence level for detecting a pose.
- min_tracking_confidence (default 0.5): Confidence level for tracking pose landmarks**.**

4.1.3 Extract Pose Landmarks from Each Frame

```
def findPose(self, img):
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    self.results = self.pose.process(imgRGB)
    return img
```

- Converts BGR to RGB (since MediaPipe works with RGB images).
- Processes the image and detects pose landmarks**.**

```
def findPosition(self, img):
    lmList = []
    if self.results.pose_landmarks:
        h, w, _ = img.shape

        left_hip = self.results.pose_landmarks.landmark[23]
        right_hip = self.results.pose_landmarks.landmark[24]
        mid_hip_x = (left_hip.x + right_hip.x) / 2
        mid_hip_y = (left_hip.y + right_hip.y) / 2
```

```
        left_shoulder = self.results.pose_landmarks.landmark[11]
        right_shoulder = self.results.pose_landmarks.landmark[12]
        shoulder_width = abs(left_shoulder.x - right_shoulder.x) + 1e-6

        for lm in self.results.pose_landmarks.landmark:
            cx = (lm.x - mid_hip_x) / shoulder_width
            cy = (lm.y - mid_hip_y) / shoulder_width
            cz = lm.z / shoulder_width
            lmList.append([cx, cy, cz])

    return np.array(lmList).flatten() if lmList else np.zeros(99)
```

Key Points of Normalization:

1.Reference Points:

- Mid-Hip (mid_hip_x, mid_hip_y): Used as a reference for position normalization.
- Shoulder Width (shoulder_width): Used for scaling the coordinates

2.Normalization Formula:

$$\text{normalized}_x = \frac{\text{landmark}_x - \text{mid\_hip}_x}{\text{shoulder\_width}}$$

$$\text{normalized}_y = \frac{\text{landmark}_y - \text{mid\_hip}_y}{\text{shoulder\_width}}$$

$$\text{normalized}_z = \frac{\text{landmark}_z}{\text{shoulder\_width}}$$

3.Returns a Flattened Numpy Array (np.array(lmList).flatten())

- Each frame contains 33 landmarks with (X, Y, Z) coordinates.
- Final output: 99 features per frame (33 landmarks × 3 coordinates)**.**

4.1.4 Extract Pose Data from Videos

```
def extract_landmarks(video_path, label, max_frames=1000):
    if not os.path.exists(video_path):
        print(f"⚠️ ERROR: Video file '{video_path}' not found.")
        return []

    cap = cv2.VideoCapture(video_path)
    detector = PoseDetector()
    all_landmarks = []

    frame_count = 0
```

```
  while cap.isOpened():
      success, img = cap.read()
      if not success or frame_count >= max_frames:
          break

      img = detector.findPose(img)
      lm_data = detector.findPosition(img)

      if lm_data is not None and np.any(lm_data):
          all_landmarks.append(np.append(lm_data, label))
          frame_count += 1

  cap.release()

print(f"✅ Extracted {frame_count} frames from {video_path}")
return all_landmarks
```

Function Explanation:

1. Checks if Video Exists (os.path.exists(video_path))
   - Avoids errors by verifying if the video file is present.

2.Opens Video (cv2.VideoCapture(video_path))
   - Reads the video file frame by frame.

3.Extracts Landmarks (detector.findPosition(img))
   - If pose landmarks are detected, they are added to all_landmarks.

4.Stores Features & Label (np.append(lm_data, label))
   - Features: 99 pose landmarks.
   - Label: 0 (Dangerous), 1 (Non-Dangerous).

5.Limits Maximum Frames (max_frames=1000)
   - Prevents excessive data collection from long videos.

4.1.5 Define Activity Labels (Danger = 0, Safe = 1)

```
activities = {
    "fighting.mp4": 0,
    "Kick1.mp4": 0,
    "Punch1.mp4": 0,
    "Shove1.mp4": 0,
    "slap1.mp4": 0,
    "Nun1.mp4": 0,
    "Shoot1.mp4": 0,
    "sitting.mp4": 1,
    "walking3.mp4": 1,
    "jumping.mp4": 1,
    "bowling1.mp4": 1
}
```

Assigns Labels:

- Dangerous Activities (0): Fighting, kicking, punching, etc.
- Non-Dangerous Activities (1): Walking, sitting, jumping.

4.1.6 Process All Videos & Save Dataset

```
all_data = []
for video, label in activities.items():
    all_data.extend(extract_landmarks(video, label))
```

- Loops through all videos and extracts pose landmarks for each activity.

```
if all_data:
    df = pd.DataFrame(all_data)
    df.to_csv("pose_dataset.csv", index=False)
    print("✅ Dataset saved as 'pose_dataset.csv' 🚀")
else:
    print("⚠️ No data extracted. Check video files.")
```

- Converts the extracted landmarks to a Pandas DataFrame and saves it as pose_dataset.csv.

```
✅ Extracted 54 frames from fighting.mp4
✅ Extracted 55 frames from Kick1.mp4
✅ Extracted 280 frames from Punch1.mp4
✅ Extracted 122 frames from Shove1.mp4
✅ Extracted 54 frames from slap1.mp4
✅ Extracted 146 frames from Nun1.mp4
✅ Extracted 78 frames from Shoot1.mp4
✅ Extracted 910 frames from sitting.mp4
✅ Extracted 1000 frames from walking3.mp4
✅ Extracted 117 frames from jumping.mp4
✅ Extracted 443 frames from bowling1.mp4
✅ Dataset saved as 'pose_dataset.csv'
```

Fig 4.1 Pose Landmark Extraction

4.2 Training a Bi-LSTM Model for Danger Detection

4.2.1 Libraries Used

sklearn.model_selection, train_test_split, tensorflow,tensorflow.keras.models Sequential, tensorflow.keras.layers, LSTM, Dense, Dropout, Bidirectional

4.2.2 Load and Process Dataset

```
csv_file = "pose_dataset.csv"
df = pd.read_csv(csv_file).values
X = df[:, :-1]
y = df[:, -1]
```

- Reads pose_dataset.csv into a NumPy array.
- X (Features): Contains pose landmark data (99 values per frame).
- y (Labels): Contains activity labels (0 = Danger, 1 = Safe).

4.2.3 Prepare Sequences for LSTM Model

```
TIME_STEPS = 30
FEATURES = 99

X_seq, y_seq = [], []
for i in range(len(X) - TIME_STEPS):
    X_seq.append(X[i:i+TIME_STEPS])
    y_seq.append(y[i+TIME_STEPS])

X_seq, y_seq = np.array(X_seq), np.array(y_seq)
```

Explanation:

- LSTMs require sequential data, so we use sliding windows of TIME_STEPS = 30 frames.
- Each sequence consists of 30 consecutive frames (X[i:i+TIME_STEPS]).
- The corresponding label is the activity at the next time step (y[i+TIME_STEPS]).
- Converts sequences into NumPy arrays for model training.

4.2.4 Split Dataset into Train & Test Sets

```
X_train, X_test, y_train, y_test = train_test_split(X_seq, y_seq, test_size=0.2, random_state=42
```

- Splits data into training (80%) and testing (20%) sets.
- Ensures reproducibility (random_state=42).

4.2.5 Define the LSTM Model

```
model = Sequential([
    Bidirectional(LSTM(128, return_sequences=True, input_shape=(TIME_STEPS, FEATURES))),
    Dropout(0.3),
    Bidirectional(LSTM(128)),
    Dense(64, activation="relu"),
    Dropout(0.3),
    Dense(2, activation="softmax")  # 2 Classes: Danger (0) & Non-Danger (1)
])
```

Layer Breakdown

1.Bidirectional LSTM (128 Units, Return Sequences: True)

- Processes both past and future context in a sequence.
- Returns sequences for stacking another LSTM layer.

2.Dropout (30%)

- Prevents overfitting by randomly dropping 30% of neurons.

- 3.Bidirectional LSTM (128 Units)
- Captures long-term dependencies for action recognition.

4.Dense Layer (64 Neurons, ReLU Activation)

- Extracts deeper features.

- 5.Dropout (30%)
- Further prevents overfitting.

6.Dense Output Layer (Softmax Activation, 2 Neurons)

- Classifies into 2 categories: Danger (0) & Safe (1).

4.2.6 Compile The Model

```
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["sparse_categorical_accuracy"])
```

- Optimizer: "adam" → Adaptive optimizer for efficient training.

- Loss: "sparse_categorical_crossentropy" → Suitable for multi-class classification with integer labels (0 or 1).

- Metric: "sparse_categorical_accuracy" → Measures accuracy for sparse labels.

4.2.7 Train the Model

```
model.fit(X_train, y_train, epochs=50, batch_size=64, validation_data=(X_test, y_test))
```

Training Details

- 50 Epochs → Model trains for 50 iterations.
- Batch Size = 64 → Processes 64 sequences per batch.
- Validation Data → Evaluates performance on test data.

4.2.8 Save the Model

```
model.save("lstm_activity_model.h5")
print("✅ Model trained & saved as 'lstm_activity_model.h5'")
```

- Saves the trained model as "lstm_activity_model.h5" for future use in real-time danger detection.

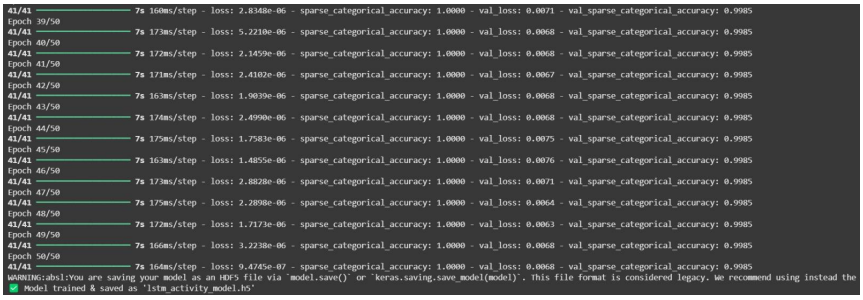

Fig 4.2.1 Training a Bi-LSTM Model for Danger Detection

4.3 Real-Time Danger Detection Using LSTM Model

4.3.1 Libraries Used

OpenCV, NumPy, TenserFlow

4.3.2 Load Trained Model & Define Labels

```
model = tf.keras.models.load_model("lstm_activity_model.h5")

actions = ["DANGER", "SAFE"]
colors = [(0, 0, 255), (0, 255, 0)]  # RED for Danger, GREEN for Safe
```

1.Loads the saved model (lstm_activity_model.h5).

2.Defines two activity classes:

- "DANGER" (Label: 0) → Red warning
- "SAFE" (Label: 1) → Green indicator

4.3.3 Load Test Video & Initialize Pose Detector

```
cap = cv2.VideoCapture("slap1 (1).mp4")
detector = PoseDetector()
sequence = []
cv2.namedWindow("Activity Recognition", cv2.WINDOW_NORMAL)
```

- Loads the test video (slap1 (1).mp4) for analysis.
- Initializes PoseDetector (a custom class using MediaPipe Pose) to extract landmarks.
- Creates an empty sequence list to store landmark sequences for LSTM prediction.

4.3.4 Process Video Frame-by-Frame

```
while cap.isOpened():
    success, img = cap.read()
    if not success:
        break

    img = detector.findPose(img)
    lm_data = detector.findPosition(img)
```

- Reads each frame (cap.read()) until the video ends.
- Detects human pose landmarks using findPose() and findPosition() functions

4.3.5 Prepare Input for LSTM Model

```
    if lm_data is not None and np.any(lm_data):
        sequence.append(lm_data)
        if len(sequence) > TIME_STEPS:
            sequence.pop(0)

        if len(sequence) == TIME_STEPS:
            input_seq = np.expand_dims(np.array(sequence), axis=0)
            prediction = np.argmax(model.predict(input_seq), axis=1)[0]
            label = actions[prediction]
            color = colors[prediction]
```

Step-by-Step Breakdown:

1.Checks if pose data is available (lm_data is not None).

2.Stores pose landmarks in a sequence (sequence.append(lm_data)).

3.Maintains a fixed sequence length (TIME_STEPS):

- If the sequence grows beyond 30 frames, removes the oldest frame (pop(0)).

4.When the sequence reaches TIME_STEPS (30 frames):

- Converts it into a NumPy array and reshapes it (expand_dims()) for model input.
- Predicts the activity class using the trained LSTM model.
- Assigns the corresponding label (DANGER or SAFE).
- Selects the display color (RED or GREEN).

4.3.6 Display Activity Prediction

```
cv2.rectangle(img, (20, 30), (250, 100), color, -1)  # Display box
cv2.putText(img, label, (50, 80),
            cv2.FONT_HERSHEY_COMPLEX, 1.2, (255, 255, 255), 3)
```

1.Draws a rectangle (cv2.rectangle()) on the screen:

- Red for danger
- Green for safe

2.Displays the label (cv2.putText()) on the video frame.

4.3.7 Show Resized Output Window

```
img_resized = cv2.resize(img, (800, 600))
cv2.imshow("Activity Recognition", img_resized)
```

- Resizes the frame to 800x600 for better visibility.
- Displays the processed frame in a window titled "Activity Recognition".

4.3.8 Exit When User Presses 'Q'

```
if cv2.waitKey(1) & 0xFF == ord('q'):
    break

cap.release()
cv2.destroyAllWindows()
```

- Press q to stop the program.
- Releases video resources (cap.release()).
- Closes all OpenCV windows (cv2.destroyAllWindows()).

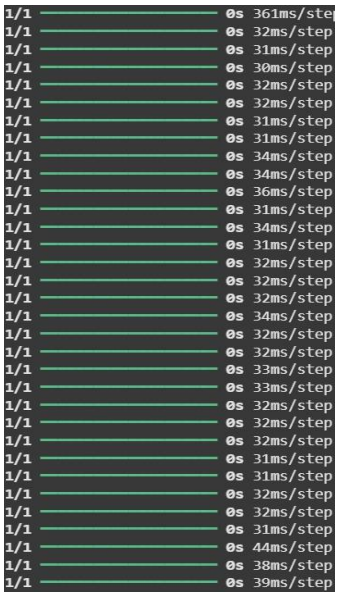

Fig 4.3.1 Real-Time Danger Detection Using LSTM Model

4.4 Implementation Explanation of LSTM-Based Danger Detection Using Pose Landmarks

4.4.1 Libraries Used

OpenCV, NumPy, TenserFlow

4.4.2 Load the Trained LSTM Model

```
model = tf.keras.models.load_model("lstm_activity_model.h5")
```

- Loads the pre-trained LSTM model stored as "lstm_activity_model.h5".
- This model was trained on sequential pose landmark data to classify dangerous vs. non-dangerous activities.

4.4.3 Define Labels and Colors for Classification Output

```
actions = ["DANGER", "SAFE"]
colors = [(0, 0, 255), (0, 255, 0)]  # RED (Danger) & GREEN (Non-Danger)
```

- The actions list contains two categories: "DANGER" and "SAFE".
- The colors list assigns red ((0, 0, 255)) for dangerous activities and green ((0, 255, 0)) for safe activities.

4.4.4 Load the Test Video

```
cap = cv2.VideoCapture("test_video.mp4")
```

- Opens and reads the "test_video.mp4" file, which is analyzed frame by frame

4.4.5 Initialize Pose Detector and Sequence Storage

```
detector = PoseDetector()
sequence = []
```

- PoseDetector() is a class (not shown in this snippet) that extracts pose landmarks from each video frame.
- sequence is a list that stores a sequence of pose frames (used as LSTM input).

4.4.6 Create a Display Window for Real-Time Activity Recognition

```
cv2.namedWindow("Activity Recognition", cv2.WINDOW_NORMAL)
```

- Creates a resizable window named "Activity Recognition" to display the processed video.

4.4.7 Read Video Frames and Extract Pose Landmarks

```
while cap.isOpened():
    success, img = cap.read()
    if not success:
        break

    img = detector.findPose(img)
    lm_data = detector.findPosition(img)
```

- Loop runs until the video ends or the user stops it.
- findPose(img) detects and marks the human skeletonon the frame.
- findPosition(img) extracts pose landmarks (X, Y, Z coordinates) for model prediction.

4.4.8 Store Pose Landmarks in Sequence for LSTM Model

```
    if lm_data is not None and np.any(lm_data):
        sequence.append(lm_data)
        if len(sequence) > TIME_STEPS:
            sequence.pop(0)
```

- If pose landmarks are detected, they are added to the sequence list.
- If the sequence length exceeds TIME_STEPS (LSTM input size), the oldest frame is removed (ensuring fixed input length).

4.4.9 Perform Activity Classification Using LSTM Model

```
        if len(sequence) == TIME_STEPS:
            input_seq = np.expand_dims(np.array(sequence), axis=0)
            prediction = np.argmax(model.predict(input_seq), axis=1)[0]
            label = actions[prediction]
            color = colors[prediction]
```

- When the sequence reaches the required TIME_STEPS, it is converted into a NumPy array and reshaped into the model's expected format.
- The LSTM model predicts the activity class ("DANGER" or "SAFE").
- The np.argmax() function extracts the class with the highest probability.
- The predicted label and color are selected based on the model's output.

4.4.10 Display the Classification Result on the Video Frame

```
        cv2.rectangle(img, (20, 20), (180, 60), color, -1)  # Smaller box
        cv2.putText(img, label, (30, 50),
                    cv2.FONT_HERSHEY_COMPLEX, 0.8, (255, 255, 255), 2)  # Smaller font
```

- A colored rectangle (red for danger, green for safe) is drawn at the top-left corner of the video frame.
- The classification label ("DANGER"/"SAFE") is displayed within the box using OpenCV's cv2.putText() function.

4.4.11 Resize and Show the Processed Video Frame

```
    img_resized = cv2.resize(img, (800, 600))
    cv2.imshow("Activity Recognition", img_resized)
```

- The frame is resized to 800x600 pixels for better visualization.
- Displays the frame with overlaid skeleton, label, and classification box.

4.4.12 Quit Video Processing on Key Press

```
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

- Press 'q' to exit the video processing loop.

4.4.13 Release Resources & Close All Windows

```
cap.release()
cv2.destroyAllWindows()
```

- Releases the video file (cap.release()).
- Closes all OpenCV windows (cv2.destroyAllWindows()).

Fig4.4.1 chocking - Danger

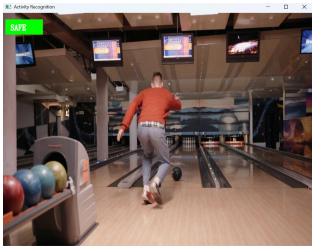

Fig4.4.2 Bowling - Safe

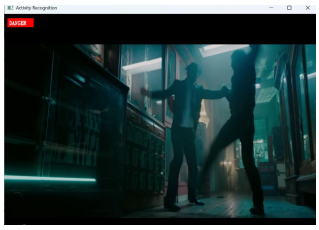

Fig4.4.3 Punching - Danger

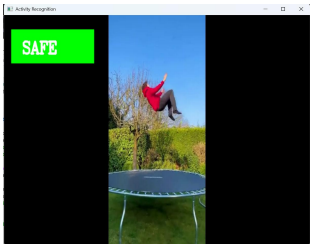

Fig4.4.4 Jumping_Safe

Chapter 5

Conclusion & Future Scope

5.1 Conclusion

This project successfully implemented an LSTM-based model for recognizing dangerous and non-dangerous human activities using pose landmarks extracted from videos. By leveraging MediaPipe for pose estimation and structuring the landmark data into time-series sequences, the model could accurately classify human motion patterns. The integration of bidirectional LSTM with normalized pose data proved effective for real-time activity recognition. The model's robustness against environmental variations demonstrates its potential in smart surveillance and security systems. Overall, it validates the use of deep learning in motion-based behavior detection, particularly for safety-critical applications.

5.2 Future Scope

i. Expand the Dataset
Including more diverse and realistic danger activities like falling, weapon use, or aggressive gestures will improve model generalization.

ii. Real-Time Optimization
Techniques like model quantization or deployment on edge devices (e.g., Raspberry Pi, Jetson Nano) can enhance performance in real-time scenarios.

iii. Multi-Camera and 3D Tracking
Implementing multi-camera systems and exploring 3D pose estimation will help track actions across different angles and improve accuracy.

iv. Hybrid Model Integration
Combining pose-based recognition with object detection (e.g., YOLO for weapon detection) can enhance context awareness in surveillance.

v. Automated Alert Systems
Integrating the system with alerts (SMS, Email, IoT alarms) will make it more useful for real-world emergency response setups.

Reference

1.Research Papers & Articles

- J. Wang, S. Tan, X. Zhen, S. Xu, F. Zheng, Z. He, and L. Shao, "Deep 3D human pose estimation: A review," Computer Vision and Image Understanding, vol. 210, p. 103225, 2021.

- J. Liu, G. Wang, L.-Y. Duan, K. Abdiyeva, and A. C. Kot, "Skeleton-based human action recognition with global context-aware attention LSTM networks," IEEE Transactions on Image Processing, vol. 27, no. 4, pp. 1586–1599, Apr. 2018.

- B. Pang, E. Nijkamp, and Y. N. Wu, "Deep learning with TensorFlow: A review," Journal of Educational and Behavioral Statistics, vol. 45, no. 2, pp. 227–248, 2020.

2.Official Documentation & Libraries

- MediaPipe Pose – https://developers.google.com/mediapipe

- TensorFlow – https://www.tensorflow.org/tutorials

- OpenCV Video Processing – https://docs.opencv.org/

- LSTM – https://colah.github.io/posts/2015-08-Understanding- LSTMs/

Contributions

Tasks	Contributers
Created comprehensive project report in PDF format	Rishika Pal, Upal Pahari
Designed and developed presentation slides showcasing key aspects of the project	Upal Pahari, RIshika Pal
Developed and organized the complete code files for pose detection, data preprocessing, LSTM training, and prediction.	Srinjoy Kundu, Sauvatra Paul
Focused on organizing and preparing the dataset to align with LSTM input requirements, which was crucial for accurate activity classification.	Rishika Pal, Upal Pahari
Produced a concise 1.5-2.5 minute teaser video (.mp4) demonstrating the project functionality and results	Sauvatra Paul, Srinjoy Kundu