HUMAN ACITIVTY TRACKING SYSTEM

MINOR PROJECT REPORT

By

VASIST ACHARYA RA2311056010077
ADITYA CHEBROLU RA2311056010116

Under the guidance of

RADHA.R

In partial fulfilment for the Course

of

21CSS202T – FUNDAMENTALS OF DATA SCIENCE

In

DATA SCIENCE AND BUSINESS SYSTEMS

FACULTY OF ENGINEERING AND TECHNOLOGY

SCHOOL OF COMPUTING

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR

NOVEMBER 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this minor project report for the course **21CSS202T FUNDAMENTALS OF DATA SCIENCE** entitled in "**HUMAN ACITIVTY RECOGNITION SYSTEM**" is the Bonafide work of **Vasist Acharya (RA2311056010077) and Aditya Chebrolu (RA2311056010116)** who carried out the work under my supervision.

SIGNATURE

Radha R

Fundamentals of Data Science Faculty

Data Science and Business Systems

SRM Institute of Science and Technology

Kattankulathur

SIGNATURE

HOD

ABSTRACT

Human activity recognition is a crucial area of research with applications spanning surveillance, healthcare, sports analytics, and human-computer interaction. This project presents a Human Activity Recognition System (HARS) that leverages a pre-trained deep learning model to identify and classify human activities in real-time from video streams. Utilizing OpenCV's deep learning module, the system processes input frames to recognize actions and provides immediate feedback through an annotated display. Key challenges addressed in this project include ensuring robustness against variations in video quality, optimizing for real-time processing, and managing the complexity of distinguishing similar activities.

The primary objectives are to achieve accurate and efficient activity recognition and to create an intuitive user interface that facilitates interaction and analysis. The system is designed to operate in various environments, providing a foundational framework for applications in security monitoring, physical rehabilitation, and performance evaluation. The project demonstrates the feasibility of using deep learning for real-time activity recognition and lays the groundwork for future advancements in the field.

TABLE OF CONTENTS

1.INTRODUCTION

1.1 Motivation

The rapid evolution of technology has transformed how we interact with the world around us, leading to a growing demand for systems that can intelligently monitor and interpret human activities. As society increasingly emphasizes health, safety, and performance, there is a pressing need for reliable and efficient human activity recognition systems. Such systems can provide valuable insights into behavior patterns, enhance training and rehabilitation efforts, and improve security measures in various settings. This project is motivated by the desire to harness deep learning techniques to create an effective and practical solution for real-time human activity recognition.

1.2 Objective

The primary objective of this project is to develop a Human Activity Recognition System (HARS) that can accurately identify and classify human activities from video input in real-time. By utilizing a pre-trained deep learning model and OpenCV's deep learning module, the system aims to deliver timely feedback on recognized activities. Additionally, the project seeks to create a user-friendly interface that displays predictions on the video feed, facilitating immediate interaction and analysis. Ultimately, the goal is to provide a foundational framework for further enhancements and applications in various domains, including sports analytics, healthcare, and security.

1.3 Problem Statement

Despite the advancements in computer vision and machine learning, accurately recognizing human activities in real-time remains a significant challenge. Existing systems often struggle with issues such as variability in lighting conditions, occlusions, and the need for extensive labeled datasets for training models. Additionally, many solutions lack the capability to provide immediate feedback to users, limiting their practical applications. This project addresses these challenges by implementing a system that leverages deep learning techniques to enhance the accuracy and efficiency of human activity recognition while ensuring real-time processing capabilities.

1.4 Challenges

The development of the Human Activity Recognition System faces several challenges, including:

1. **Data Variability**: Differences in video quality, lighting conditions, and camera angles can significantly impact the model's performance. Ensuring the system is robust against these variations is crucial for reliable recognition.
2. **Real-Time Processing**: Achieving real-time activity recognition requires optimizing the processing pipeline to minimize latency while maintaining accuracy. This involves efficient model inference and resource management.
3. **Complex Activity Recognition**: Distinguishing between similar activities or recognizing complex actions that involve multiple movements can pose challenges for the model. Enhancing the model's capability to handle these scenarios is essential.
4. **User Interaction**: Creating an intuitive user interface that effectively displays recognized activities and allows for seamless user interaction is critical for the system's usability.

2.DATA UNDERSTANDING

The Kinetics Human Action Video Dataset

This dataset is used for **Human Activity Recognition (HAR)**, where the objective is to predict specific actions captured in over 400 video clips. Each action or activity has an associated video clip, and the dataset includes these labelled video samples for training and evaluation.

Here are the key points of the data:

1. **Video Clips**: Each clip represents a different human activity, providing visual data for each action.
2. **Action Labels**: For each video, there is a corresponding label representing the type of activity performed in that clip. These labels are essential for training the model to recognize and predict actions.
3. **Text File with Class Labels**: A text file listing all possible action labels, one per line. This is used by the model to match and display the predicted action label for each clip.
4. **Model Requirements**: A pre-trained deep learning model is required to recognize and predict activities in new, unseen videos by comparing their features to the labelled examples in the dataset.

3.DATA PREPARATION

1. **Loading Class Labels**:

- The class labels, representing possible activities, are read from a text file provided via the --classes argument.
- Each line in the file contains a unique action label, and these labels are stored in a list (ACT). This list allows the model to later map its output to human-readable activity names.

2. **Frame Sampling**:

- The code captures frames from the video stream in batches of 16 frames (defined by SAMPLE_DURATION). This batch of frames represents a short clip from the video and serves as the input unit for the model.
- Each frame is resized to 112x112 pixels (set by SAMPLE_SIZE) to match the input size expected by the model, reducing computational load while preserving enough detail for action recognition.

3. **Blob Construction**:

- After sampling, the frames are combined into a "blob" using OpenCV's cv2.dnn.blobFromImages function. The blob creation involves:
- **Normalization**: Each frame is scaled to standardize pixel intensity values, which improves model accuracy.
- **Colour Adjustment**: The function swaps the colour channels to the order expected by the model.
- **Reshaping**: The frames are reshaped and organized as a single blob with the required batch size and dimensions, making it ready for input.

4. **Preparing Model Input**:

- The blob is then transposed and expanded in dimensions as needed to match the model's input shape. This ensures compatibility with the deep learning model's architecture, so it can process the entire sequence of frames as a single input.

5. **Prediction Mapping**:

- Once the model outputs its prediction scores for each activity, the code uses np.argmax to find the index of the highest probability. This index corresponds to an action label in the ACT list, mapping the model's numerical output to a readable action label.

6. **Displaying the Predicted Action**:

- The predicted action label is drawn onto each original frame, overlaying a text box with the identified action, ready for display or saving to an output file.

4.EXPLORATORY DATA ANALYSIS (EDA)

For a Human Activity Recognition (HAR) project with video data, **Exploratory Data Analysis (EDA)** focuses on understanding the data's structure, distribution, and characteristics. Since HAR involves video sequences, EDA extends beyond traditional techniques for tabular data. Here's a breakdown of effective EDA steps for such a dataset:

1. Class Distribution Analysis

- **Objective**: Check the distribution of activity labels to understand if the dataset is balanced.
- **Method**: Plot a bar chart showing the frequency of each activity label. This can help identify whether some activities are overrepresented or underrepresented, which may affect model performance.
- **Outcome**: If there's a significant imbalance, consider strategies like data augmentation for underrepresented classes.

2. Duration and Frame Rate Analysis

- **Objective**: Understand the duration of each video and frame rate consistency, as they impact model training and feature extraction.
- **Method**: Calculate and visualize the distribution of video lengths and frames per second (FPS) across the dataset. This can reveal potential anomalies, such as unusually long or short videos, that may need adjustment.
- **Outcome**: Based on these insights, standardize video lengths and FPS if necessary, to improve model consistency.

3 .Frame Quality and Resolution Analysis

- **Objective**: Check the visual quality and resolution of video frames to ensure they are clear enough for model interpretation.
- **Method**: Sample frames from random videos across different classes to examine brightness, contrast, and resolution. Check for any low-quality frames or variability in resolution that might degrade model performance.
- **Outcome**: If low-quality frames are prevalent, consider using filters or enhancement techniques to improve frame clarity.

4 . Motion Patterns and Action Complexity

- **Objective**: Understand the complexity of actions and their motion patterns to inform model architecture choices.
- **Method**: Use optical flow analysis to visualize the motion across frames for different activities. Plot trajectories of keypoints or use heatmaps to assess movement dynamics in each video.
- **Outcome**: Activities with complex, fast motion may require a model that captures temporal dependencies effectively, like an LSTM or 3D CNN.

5. Temporal Consistency and Sampling Effectiveness

- **Objective**: Check if sampled frames capture the action effectively and if temporal consistency is maintained.
- **Method**: For a subset of videos, visualize the sampled frames in sequence to verify that they represent the complete action. This can also involve examining the variance between consecutive frames.
- **Outcome**: Adjust SAMPLE_DURATION (number of frames per batch) if needed to capture actions more accurately.

6. Activity Transition and Overlap Analysis

- **Objective**: Identify if some activities have overlapping characteristics, which might cause misclassification.
- **Method**: Compare key frames of similar activities (e.g., “walking” vs. “running”) and calculate pixel-level similarity or feature-based similarity to gauge overlap.
- **Outcome**: Activities with high overlap may benefit from additional differentiating features or more specialized processing.

5.RESULTS AND DISCUSSION

Here are two images:

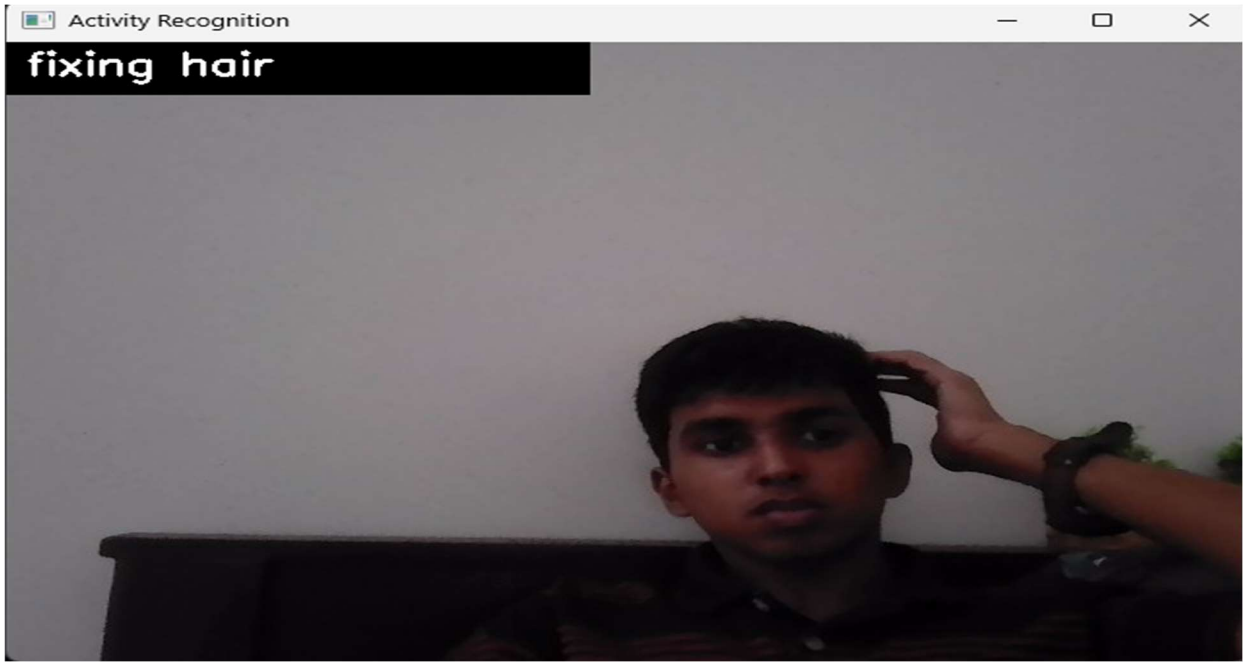

Fig 1

In Fig 1, the user is fixing their hair, with a label overlay from the model identifying the action as "fixing hair."

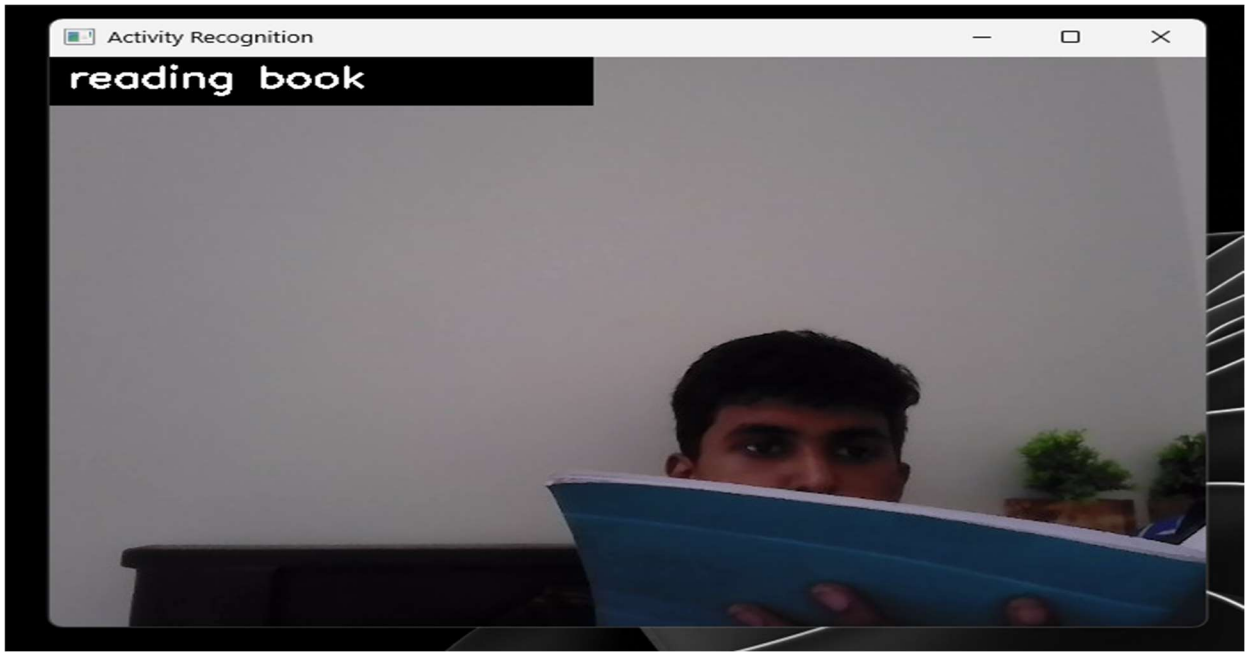

Fig 2

In Fig 2, the user is reading a book, with a label overlay from the model identifying the action as "reading book"

6.CONCLUSION

The Human Activity Recognition project successfully demonstrates the application of deep learning techniques to identify and classify human activities in real-time video streams. By leveraging a pre-trained model and OpenCV's DNN module, the system efficiently processes video input to recognize and display activities with minimal latency.

Future enhancements could include improving the accuracy of activity predictions through additional training data or optimizing the model for faster processing speeds. Additionally, expanding the list of recognized activities and integrating advanced features, such as multi-person recognition or environmental context analysis, could further enhance the system's applicability.

Overall, this project serves as a foundational step towards building more sophisticated human activity recognition systems, paving the way for advancements in automated monitoring and analysis in diverse applications.

7.REFERENCES

https://opencv.org/
https://onnx.ai/
https://openai.com/index/chatgpt
https://www.v7labs.com/blog/human-activity-recognition
https://www.geeksforgeeks.org/human-activity-recognition-using-deep-learning-model
https://www.geeksforgeeks.org/human-activity-recognition-with-opencv/?ref=asr1

8.APPENDIX

A. Technical Specifications

- **Programming Language:** Python
- **Libraries Used:**
  - NumPy: For numerical computations.
  - OpenCV: For image processing and computer vision tasks, including DNN functionality.
  - imutils: For functions related to image processing.
- **Model Specifications:**
  - The project utilizes a pre-trained deep learning model compatible with OpenCV's DNN module for human activity recognition.
  - Input dimensions for the model are set to a sample size of 112x112 pixels.
- **System Requirements:**
  - Python 3.x
  - OpenCV version compatible with DNN module
  - GPU support for accelerated processing

B. Command-Line Arguments

The program accepts the following command-line arguments:

-m or --model	Path to the pre-trained model file.
-c or --classes	Path to the class labels file containing activity labels.
-i or --input	Path to the input video file (can be left blank for webcam input).
-o or --output	Path for saving the output video file (optional).
-d or --display	Integer flag to display output frames (1 to display, 0 to not display).
-g or --gpu	Integer flag to use GPU (1 for yes, 0 for no).

C. Example Usage

To run the program, users can execute the following command in the terminal:

Bash

python activity_recognition.py -m <model_path> -c <classes_file_path> -i <input_video_path> -o <output_video_path> -d 1 -g 0

Replace <model_path>, <classes_file_path>, <input_video_path>, and <output_video_path> with the appropriate file paths.

D. Code Snippet

```
#HUMAN ACTIVITY RECOGNITION

import numpy as np
import argparse
import imutils
import sys
import cv2

# we will pass argument using argumetn parser so construct argument parser.
argv = argparse.ArgumentParser()
argv.add_argument("-m", "--model", required=True, help="specify path to pre-trained model")
argv.add_argument("-c", "--classes", required=True, help="specify path to class labels file")
argv.add_argument("-i", "--input", type=str, default="", help="specify path to video file")
argv.add_argument("-o", "--output", type=str, default="",   help="path to output video file")
argv.add_argument("-d", "--display", type=int, default=1,   help="to display output frmae or not")
argv.add_argument("-g", "--gpu", type=int, default=0,   help="whether or not it should use GPU")
args = vars(argv.parse_args())

# declare an variable to open and load contents of labels of activity .
# specify size here for the frames.
ACT = open(args["classes"]).read().strip().split("\n")
SAMPLE_DURATION = 16
SAMPLE_SIZE = 112



# Load the Deep Learning model.
print("Loading The Deep Learning Model For Human Activity Recognition")
gp = cv2.dnn.readNet(args["model"])


#Check if GPU will be used here

if args["gpu"] > 0:
    print("setting preferable backend and target to CUDA...")
    gp.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
    gp.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)
```

```
# Grab the pointer to the input video stream
print(" Accessing the video stream...")
vs = cv2.VideoCapture(args["input"] if args["input"] else 0)
writer = None
fps = vs.get(cv2.CAP_PROP_FPS)
print("Original FPS:", fps)


# Detect continoulsy till terminal is expilicitly closed
while True:
    # Frame intilasation
    frames    = []  # frames for processing
    originals = []  # original frames


    # Use sample frames
    for i in range(0, SAMPLE_DURATION):
        # Read a frame from the video stream
        (grabbed, frame) = vs.read()
        # to exit video stream
        if not grabbed:
            print("[INFO] No frame read from the stream - Exiting...")
            sys.exit(0)
        # or else it read
        originals.append(frame) # save
        frame = imutils.resize(frame, width=400)
        frames.append(frame)

    #  frames array is filled we can construct our blob
    blob = cv2.dnn.blobFromImages(frames, 1.0, (SAMPLE_SIZE, SAMPLE_SIZE), (114.7748, 107.7354,
99.4750),
                                  swapRB=True, crop=True)
    blob = np.transpose(blob, (1, 0, 2, 3))
    blob = np.expand_dims(blob, axis=0)


    # Predict activity using blob

    gp.setInput(blob)
    outputs = gp.forward()
    label = ACT[np.argmax(outputs)]

    # for adding lables

    for frame in originals:
        # append predicted activity

        cv2.rectangle(frame, (0, 0), (300, 40), (0, 0, 0), -1)
        cv2.putText(frame, label, (10, 25), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2)

        # if displayed is yes

        if args["display"] > 0:
            cv2.imshow("Activity Recognition", frame)
            key = cv2.waitKey(1) & 0xFF
```

```
            # to exit
            if key == ord("q"):
                break

        # for output video boing already given
        # intialise the witer variable
        if args["output"] != "" and writer is None:
            fourcc = cv2.VideoWriter_fourcc(*'mp4v')# *'MJPG' for .avi format
            writer = cv2.VideoWriter(args["output"], fourcc, fps,
                (frame.shape[1], frame.shape[0]), True)

        # write frame to ouput
        if writer is not None:
            writer.write(frame)

# ------------------------------
#   USAGE
# ------------------------------
# python human_activity_recognition_deque.py --model resnet-34_kinetics.onnx --classes
action_recognition_kinetics.txt --input videos/example_activities.mp4
# python human_activity_recognition_deque.py --model resnet-34_kinetics.onnx --classes
action_recognition_kinetics.txt

# ------------------------------
#   IMPORTS
# ------------------------------
# Import the necessary packages
from collections import deque
import numpy as np
import argparse
import imutils
import cv2

# Construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-m", "--model", required=True, help="path to trained human activity recognition
model")
ap.add_argument("-c", "--classes", required=True, help="path to class labels file")
ap.add_argument("-i", "--input", type=str, default="", help="optional path to video file")
args = vars(ap.parse_args())

# Load the contents of the class labels file, then define the sample duration (i.e., # of frames for
classification) and
# sample size (i.e., the spatial dimensions of the frame)
CLASSES = open(args["classes"]).read().strip().split("\n")
SAMPLE_DURATION = 16
SAMPLE_SIZE = 112

# Initialize the frames queue used to store a rolling sample duration of frames -- this queue will
automatically pop out
# old frames and accept new ones
frames = deque(maxlen=SAMPLE_DURATION)

# Load the human activity recognition model
print("[INFO] Loading the human activity recognition model...")
net = cv2.dnn.readNet(args["model"])
```

```
# Grab the pointer to the input video stream
print("[INFO] Accessing the video stream...")
vs = cv2.VideoCapture(args["input"] if args["input"] else 0)

# Loop over the frames from the video stream
while True:
    # Read the frame from the video stream
    (grabbed, frame) = vs.read()
    # If the frame was not grabbed then we have reached the end of the video stream
    if not grabbed:
        print("[INFO] No frame read from the video stream - Exiting...")
        break
    # Resize the frame (to ensure faster processing) and add the frame to our queue
    frame = imutils.resize(frame, width=400)
    frames.append(frame)
    # If the queue is not filled to sample size, continue back to the top of the loop and continue
    # pooling/processing frames
    if len(frames) < SAMPLE_DURATION:
        continue
    # Now the frames array is filled, we can construct the blob
    blob = cv2.dnn.blobFromImages(frames, 1.0, (SAMPLE_SIZE, SAMPLE_SIZE), (114.7748, 107.7354,
99.4750),
                                  swapRB=True, crop=True)
    blob = np.transpose(blob, (1, 0, 2, 3))
    blob = np.expand_dims(blob, axis=0)
    # Pass the blob through the network to obtain the human activity recognition predictions
    net.setInput(blob)
    outputs = net.forward()
    label = CLASSES[np.argmax(outputs)]
    # Draw the predicted activity on the frame
    cv2.rectangle(frame, (0, 0), (300, 40), (0, 0, 0), -1)
    cv2.putText(frame, label, (10, 25), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2)
    # Display the frame to the screen
    cv2.imshow("Activity Recognition", frame)
    key = cv2.waitKey(1) & 0xFF
    # If the 'q' was pressed, break from the loop
    if key == ord("q"):
        break
```

E. Future Work

- Model Optimization: Explore options for model quantization and pruning to enhance speed and efficiency.
- User Interface Enhancements: Consider developing a graphical user interface (GUI) for easier interaction.
- Multi-Person Recognition: Implement functionality to recognize multiple individuals in a single frame, enhancing usability in crowded environments.
- Integration with Other Systems: Develop APIs for integration with other applications, allowing for expanded functionality and data sharing.