Fall Detection from video footage

55-708252- AI Research and Development Project

Code Description

Supervisor: Dr. Jing Wang

Students:

Mohammed Alsiraji – 33064634

Shah Sawar Jan - 33085681

Preety Batta - 33079741

Afaq Ahmed Javed - 33075619

This document details the fall detection prototype developed through computer vision, deep learning, and machine learning techniques. It covers feature extraction, model training, prototype testing, and results analysis.

Related Files

- “Dataset_of_FEATURES_MoveNet&MediaPipe.ipynb”
- “Models_Training.ipynb”
- “PrototypesDemo.ipynb”
- “Charts&Graphs.ipynb”

Feature Extraction: Related to "Dataset_of_FEATURES_MoveNet&MediaPipe.ipynb" file

The project involves computer vision, deep learning, and machine learning components. We start by importing the necessary libraries: TensorFlow (for deep learning), OpenCV (for image processing), Pandas (for data handling), and Matplotlib (for data visualization). We also use the TensorFlow Hub to load MoveNet pre-trained model and MediaPipe package for The MediaPipe Pose Landmarker.

Model Setup:

We load and prepare two pre-trained pose estimation models: MoveNet and MediaPipe. The MoveNet model is loaded using TensorFlow Hub, with two model variants "MoveNet_lightning" and "MoveNet_thunder," which differ in speed and accuracy we use Lightning version. For MediaPipe, we use the Pose Landmarker task, which aids in detecting landmarks in human bodies in images or videos. These models allow us to identify key locations on the human skeleton and analyse posture.

Data Handling and Visualization:

We utilize OpenCV for processing images and videos, while Pandas helps manage data in DataFrames. The results, including annotated frames, are visualized using Matplotlib, employing tools like "`LineCollection`" and "`patches`" for detailed illustrations.

Dataset and Processing Pipeline:

The Le2i Fall Detection dataset was created by the Le2i Laboratory at the University of Burgundy in France to develop and test algorithms for fall detection, particularly in elderly monitoring systems. It includes a variety of scenarios simulating falls and non-fall activities, recorded under different conditions and with various actors (Charfi et al., 2013).

Our dataset is 40 fall sequences (falls) filtered visually by inspecting all the videos from the publicly available Le2i dataset, the video files are then saved in the designated directory to serve as the training dataset. Each of the 40 videos in the dataset is processed sequentially.

The processing involves two feature extraction functions, one for MoveNet (`extract_keypoints_MV`) and one for MediaPipe (`extract_keypoints`). These functions extract keypoints from each video and output annotated videos along with CSV files containing the data.

Pose estimation models MediaPipe and MoveNet use Convolutional Neural Networks (CNNs) to extract features the process involves input processing, keypoint detection and pose estimation. The input image or video frame is processed to detect human figures then CNNs are used to identify and locate key points of the human body (e.g., joints like elbows, knees). The detected key points are used to estimate the pose by connecting them to form a skeleton-like structure. After processing, we generate annotated videos and corresponding CSV files containing data extracted from the videos.

MoveNet Processing:

The function "`extract_keypoints_MV`" processes videos using the MoveNet model. Keypoints are extracted from each frame as normalized (x, y) pairs. We start by reading the video with `cv2.VideoCapture(VIDEO_IN)`. If the video cannot be opened, an error message is displayed, and the process is terminated. We then retrieve video properties, such as frames per second, width, and height. Frames are cropped to focus on the area of interest (where the person is present). The keypoints are predicted for each frame, resulting in 17 normalized (x, y) pairs representing different body joints. These keypoints are annotated on the frames, and the cropped region is adjusted accordingly. The keypoints data is collected per frame, including metadata like video ID, and frame index, and is eventually stored in a CSV file. This file is saved in a designated output directory and have these columns.

- **Video id**
- **Label:** initial default value is 0
- **Frame index**
- **Features columns** [Nose, Left Eye, Right Eye, Left Ear, Right Ear, Left Shoulder, Right Shoulder, Left Elbow, Right Elbow, Left Wrist, Right Wrist, Left Hip, Right Hip, Left Knee, Right Knee, Left Ankle, Right Ankle]

MediaPipe Processing:

Similarly, the function "`extract_keypoints`" uses MediaPipe to extract keypoints. The setup involves "`baseoptions`" and "`PoseLandmarkerOptions`"to configure the model with specific assets and options, including handling segmentation masks for input and output videos. We read the video frames, retrieve frames per second, and determine the video properties such as height, width, and total frames. The frames are converted to RGB format for MediaPipe processing. We focus on detecting a single person, with default keypoints set to 17 pairs of (0, 0) if no person is detected. Keypoints are extracted and annotated on the frames. Similar to the MoveNet process, the data is collected and saved as CSV files in the output directory.

Models Training: Related to "Models_Training.ipynb" file

Both functions generate detailed CSV files, each containing keypoint data along with metadata such as video number and frame index. These files are crucial for further analysis and are organized into separate directories based on the model used (MoveNet or MediaPipe).

Preprocessing:

The feature extraction steps produce to the dataset that will be used to training. In this section we need to do some data preparation, we have 40 csv file for MediaPipe and 40 csv files for MoveNet the process is the same for both.
To prepare our dataset for training a fall detection model, we begin by merging 40 CSV files—each representing a video—into a single dataset. The combined file is sorted in

ascending order by video number and frame index. This sorted order is crucial for accurately labelling the data.
The training videos depict two main sequences: an initial phase where the person is standing or walking, followed by a fall, after which the person remains on the ground. For precise labelling, we manually review each frame. The frames before the fall are labelled as (0) for "non-fall," and those after the fall are labelled as (1) for "fall." The most challenging aspect of this process is pinpointing the exact frame where the body first contacts the ground. We chose a manual labelling approach to ensure the highest accuracy, carefully observing all 40 videos, which include a total of 11,723 frames.
Once the dataset is labelled, the original dataset file is duplicated and processed according to the specific model being trained.
We prepare the data for three types of models: Support Vector Machine (SVM), Multi-Layer Perceptron (MLP), and Long Short-Term Memory (LSTM) networks. Each model has unique strengths. SVMs are effective for high-dimensional spaces, MLPs are good for complex patterns, and LSTMs excel in temporal sequence analysis.

1. **SVM Preprocessing:**
    - For SVM, preprocessing involves selecting the feature columns (which consist of (x, y) pairs) and the label column. Each feature originally has a shape of (number of samples, 17), corresponding to the 17 keypoints. These are flattened into a shape of (number of samples, 34), where each new feature column is split into two separate columns for the x and y coordinates.
2. **Sequential Models Preprocessing (MLP & LSTM):**
    - For these models, we maintain the original structure of the data as (number of samples, 17, 2) this format is suitable for models that handle temporal sequences and can process the spatial dimensions of the data.

Before training the models, the dataset is split into training and testing sets. For SVM, we allocate 95% of the data for training and 5% for testing. For sequential models, the split is 80% for training and 20% for LSTM and MLP was overfitted with all the inputs. This division ensures that the models are trained on a comprehensive dataset while retaining a sufficient portion for evaluating model performance.

Training results:

We have two datasets and three binary classification models; therefore, we get six different types of prediction models. Both SVM models map input data into a high-dimensional space using a linear kernel function it finds the optimal hyperplane that maximizes the margin between the classes and data points are classified based on which side of the hyperplane they fall on.
MLP is a type of feedforward artificial neural network. The features (number of samples, 17 keypoints, 2 coordinates) are flattened using a Flatten layer followed by two hidden Dense layers with ReLU activation and Dropout for regularization. The model is trained using backpropagation to minimize the “binary cross entropy” loss function by adjusting weights using Adam optimizer. The binary output layer assigns probabilities to two classes.

LSTM is a type of recurrent neural network (RNN) designed to handle sequence prediction problems. The network processes input sequences, which consist of (the number of samples, 17 keypoints, and 2 coordinates). Each LSTM layer has 50 units, where each unit represents an LSTM cell that helps in maintaining and updating the memory state over long sequences. The cells have input, output, and forget gates controlling the flow of information, enabling the network to capture temporal dependencies. To prevent overfitting, a dropout rate of 20% is used. The model utilizes the Adam optimizer and binary cross-entropy loss function, with classification done via a sigmoid activation for binary output.

Both neural networks are trained using TensorFlow/Keras with batch sizes of 32 and 50 epochs. The models cross-validation accuracy scores after training are shown in the below table:

	SVM	MLP	LSTM
MediaPipe	97%	90%	96%
MoveNet	98%	94%	97%

Prototypes: Related to "PrototypesDemo.ipynb" file

Testing dataset:

The URFD dataset is used in our testing process of the prototypes created. This dataset contains 70 (30 falls + 40 activities of daily living) recorded with 2 Microsoft Kinect cameras camera 0 and camera 1 (parallel to the floor and ceiling mounted, respectively) both cameras record RGB and depth. Corresponding accelerometric data was collected using two PS Move devices, the dataset is organized as follows. Each row contains sequence of depth and RGB images for camera 0 and 1, synchronization data and raw accelerometer data. Each video stream is stored in separate zip archive in form of png image sequence (Kwolek & Kepski, 2014). We have used camera (0) RGB of image sequences of 30 fall events to be predicted and their corresponding synchronised sensor data to serve as a ground truth.
The testing dataset will be first prepared for prediction, the image sequences are retrieved as 30 files of images each image is RGB frame of the sequence. We converted each file to video format using OpenCV functions and by maintaining the order of the sequence we generated 30 videos in mp4 format from the 30 RGB image sequences files present in the URFD and saved into a new directory to serve as our validation set.

Prediction Methodology:

In this section we discuss how a single video is predicted and processed, and we simply iterate over our validation set to be analysed by the six prediction model variations, each model produces 30 annotated videos and 30 CSV files.

- Six trained models are loaded.
- Predict and process single video.
    - Pass the video to six different feature extraction functions, one for each model, to be processed frame by frame.
        - Get the frame keypoints pass them to be predicted. The feature extraction functions from before are restructured to handle a new element of prediction instead of saving the 17 keypoints, we flatten them the same way explained in the preprocessing section and predict the result of the specified model.
            - If it is 1 for (SVM) predictions or greater than 0.7 (the threshold for sequential model):
                - Fall detected.
                - Register the first time of getting a fall prediction.
            - Register the frame execution time.
            - The prediction is annotated on the frame.
            - The image written out to the output video as a new frame.
            - Register the following as frame data 'Video_num', 'frame index','predictions', 'frame_proccessing_duration' 'fall_detection_duration_perVideo', 'frame_width', 'frame_height'
        - Save all frames data as a single video data CSV file, save new annotated video.
    - The result of each model is saved separately, which means 6 videos and 6 CSV files.

Testing Results: Related "Charts&Graphs.ipynb" file

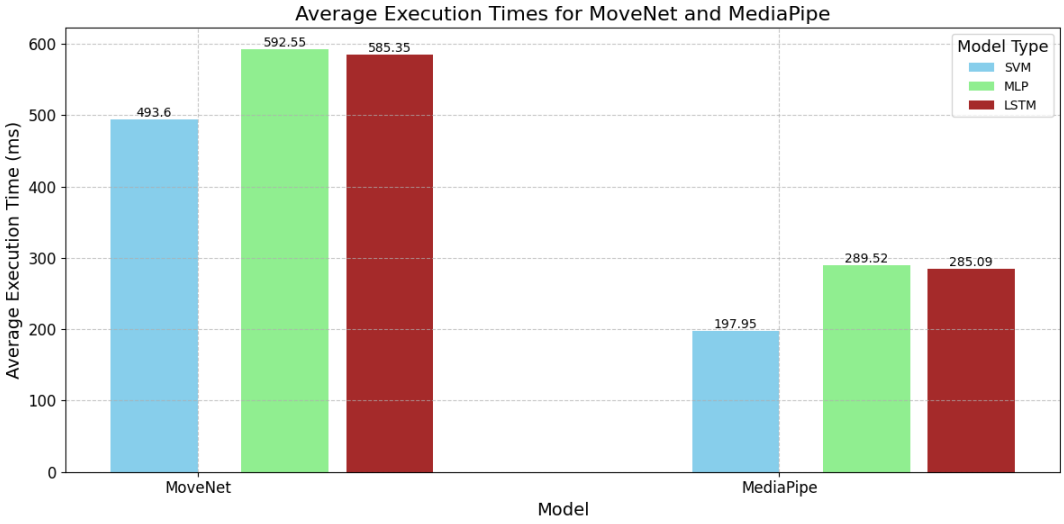
To understand our approach of the preparation of the synchronised sensor data, we explain the structure and analysis of one of the files. The file has 3 columns frame number, timestep and synchronised sensor value. The interpretation of using accelerometer in our understanding is the highest the value is the faster the person is going down, for this reason we decided to choose the frame with the highest value as our fall frame just as the image below, to separate the video into two sequences non-fall (0) before the fall frame and fall (1) after that specific frame, following a similar way of our training data preprocessing but this time a sensor data instead of visual inspection to generate a binary ground truth labels for the testing dataset.

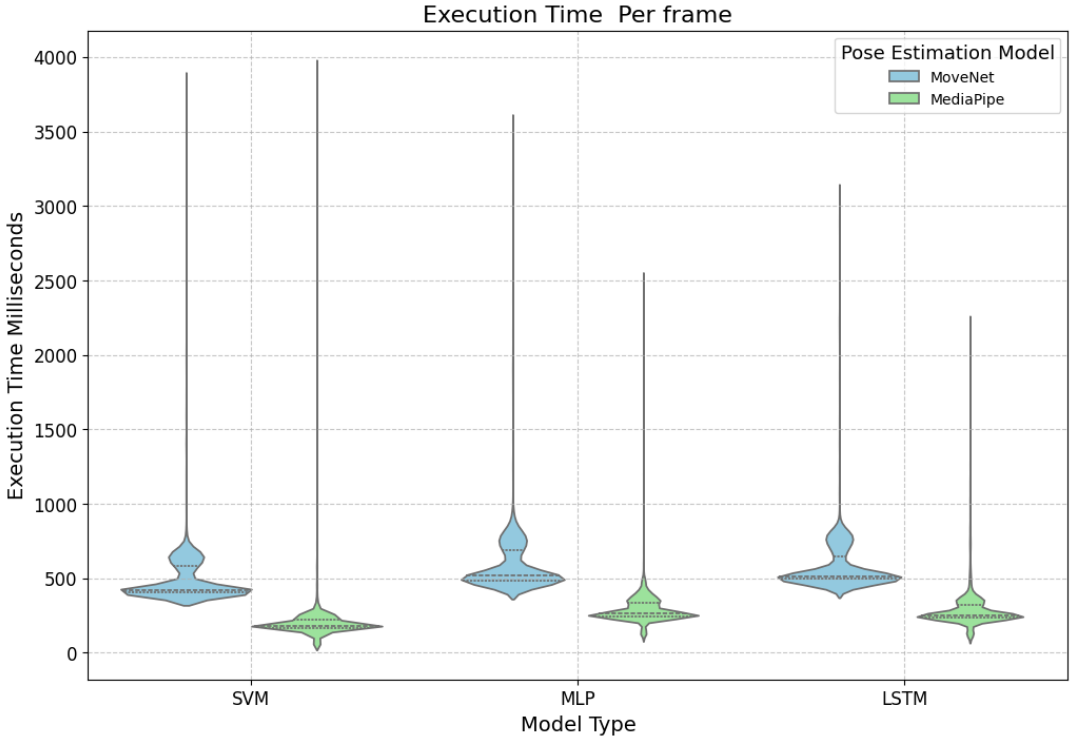
To get the results of our predictions against our ground truth label we did a simple data manipulation to the predictions generated by the sequential models (MLP and LSTM) because they give the prediction as float number from 0 to 1, we used the same threshold of determining a fall (0.7) to produce two new list of predictions, if the prediction less than the threshold assign 0 if greater assign 1. Therefore, we got 3 binary lists of predictions to do the comparison straightforward.

THE RESULTS OF THE MODEL'S PREDICTIONS PER FRAME COMPARED TO THE GROUND TRUTH LABELS

	SVM	MLP	LSTM
MEDIAPIPE	83.34%	83.11%	78.13%
MOVENET	84.07%	83.51%	78.56%

To evaluate the performance of pose estimation and classification models execution time for each frame processed by the models. We recorded the time taken from the start to the end of processing each frame individually. The results, shown in the following images, reveal that MediaPipe generally has a lower average execution time compared to MoveNet. Additionally, SVM models consistently demonstrate a slight but noticeable advantage in execution time.


Average Execution Times for MoveNet and MediaPipe
Average Execution Time (ms)
Model
MoveNet
MediaPipe
493.6
592.55
585.35
197.95
289.52
285.09
Model Type
SVM
MLP
LSTM


Execution Time Per frame
Execution Time Milliseconds
Model Type
SVM
MLP
LSTM
Pose Estimation Model
MoveNet
MediaPipe

Reference:

Charfi, I., Miteran, J., Dubois, J., Atri, M., & Tourki, R. (2013). Optimized spatio-temporal descriptors for real-time fall detection: comparison of support vector machine and Adaboost-based classification. *Journal of Electronic Imaging*, *22*(4), 041106-041106.

Kwolek, B., & Kepski, M. (2014). Human fall detection on embedded platform using depth maps and wireless accelerometer. *Computer methods and programs in biomedicine*, *117*(3), 489-501.