Sensors and Materials, Vol. 37, No. 4 (2025) 1639–1656
MYU Tokyo

S & M 4010

Real-time Fall Detection and Reporting System Using the AlphaPose Model of Artificial Intelligence

Yuh-Shihng Chang* and Guan-Yu Lin

Department of Information Management, National Chin-Yi University of Technology,
No. 57, Sec. 2, Zhongshan Rd., Taiping Dist., Taichung 411030, Taiwan (R.O.C.)

(Received October 28, 2024; accepted April 9, 2025)

Keywords: fall detection, artificial intelligence, human activity recognition and behavior understanding, reporting system, AlphaPose model

Falling is a prevalent and hazardous event that can lead to severe injuries, such as limb fractures or spinal damage, especially for elderly individuals in hospital care. In this study, we aim to develop a machine-learning-based system for effective fall detection and prompt intervention. We applied deep learning techniques, particularly the AlphaPose + Spatial Temporal Graph Convolutional Network (ST-GCN) model, to enhance human activity recognition and behavior analysis. These advanced machine learning models allow for the real-time monitoring of fall events by accurately identifying abnormal movements and behaviors associated with falls. In this study, we employed a web camera as a sensor to capture the human pose, and the AI-powered system achieved an accuracy rate exceeding 96% in training results, showcasing its robustness in detecting falls. Upon detection, the system sends immediate alerts via communication software, ensuring timely notifications to healthcare providers or family members. This machine learning approach significantly improves the safety of elderly individuals by reducing response time and minimizing the risk of fall-related injuries.

1. Introduction

Falls are a common and serious problem among the elderly population. When older adults are waiting for treatment or receiving care in hospitals, falls can lead to severe consequences, including fractures, internal bleeding, trauma, and even life-threatening situations. Falls are the second leading cause of unintentional injury-related deaths worldwide. According to the World Health Organization, an estimated 684000 people die from falls each year, with more than 80% of these fatalities occurring in low- and middle-income countries. Among fatal falls, adults aged 60 and above represent the largest proportion. Each year, there are approximately 37.3 million falls that are severe enough to require medical attention. Globally, falls result in more than 38 million disability-adjusted life years lost, contributing to life loss due to disabilities.(1) When nurses or family members are unable to constantly supervise elderly patients, the risk of falls

*Corresponding author: e-mail: eric_chang@ncut.edu.tw
https://doi.org/10.18494/SAM5419

increases significantly. Many elderly individuals may be unable to call for help in time, potentially worsening their injuries.

As the history of computer vision illustrates, the availability of larger datasets, combined with advancements in computing power, frequently leads to paradigm shifts. In this context, convolutional neural networks (CNNs) have emerged as the de facto standard in the field. Owing to their exceptional feature extraction capabilities, CNNs excel in various visual tasks, particularly in fall detection applications. In recent years, numerous studies have focused on the application of CNNs within AI technology for fall detection. Key literature includes the following:

Li *et al.* proposed a fall detection method based on CNNs, utilizing video sequences to identify fall events by sorting images of various actions.(2) They applied CNNs to each frame of the video to extract human poses. As part of the data preprocessing, the average image of all training and test images is calculated, and then this average image is subtracted from each training and test image to achieve uniform brightness. The advantage of this study is its ability to capture time series characteristics with high accuracy. However, it has limited adaptability to different environments or lighting conditions and is highly dependent on specific datasets.(2)

Butt *et al.* employed deep learning technology by combining CNNs and long short-term memory (LSTM) networks for fall detection.(3) For comparative analysis, the performance characteristics of two deep learning architectures—LSTM and CNN-based transfer learning—were evaluated. Notably, they found that CNN transfer learning achieved the highest quantitative performance, reaching an accuracy of 98%. The advantage of this study lies in its consideration of both LSTM and CNN architectures, which enhances detection accuracy and improves the understanding of action continuity. However, a notable drawback is that deep learning models require substantial computing resources and lengthy training times.(3)

Benoit *et al.* conducted a comparative study of nine neural network models applied to fall detection in the elderly.(4) The models examined included Dense, CNN, LSTM, Gated Recurrent Unit (GRU), BiLSTM, Bidirectional GRU (BiGRU), CNN Dense, CNN LSTM, and CNN GRU, all aimed at predicting falls prior to impact with the ground using accelerometer data. The models were optimized using Keras Tuner and the TensorFlow backend, a leading deep learning framework. The results indicate that deep learning approaches demonstrate a higher classification accuracy and a more streamlined software architecture, although this comes at the cost of energy efficiency and reduced inference speed.(4)

Wang *et al.* demonstrated that the back-propagation neural network can effectively avoid local optimal solutions and converges more rapidly.(5) The fall detection accuracies achieved with the German Aerospace Center, Smart Fall, and University of Rzeszów Fall (URFall) datasets were 98.3, 92.0, and 96.1%, respectively. This research provides technical support for portable, low-power wearable fall detection systems that can adapt to various environmental conditions. The integration of multiple data sources enhances the accuracy and robustness of detection. The high accuracy in recognizing fall behaviors paves the way for portable, multi-application scenarios with low power consumption for future fall detection systems.(5)

Overall, CNN-based fall detection techniques show strong potential but also face several challenges. Our research aims to enhance the adaptability and generalization ability of the model through machine learning technology and integrate web cameras as sensors to detect human body movements, thereby improving the accuracy and reliability of fall detection.

The main goal of this study is to develop an instant fall detection and notification system based on the AlphaPose artificial intelligence model to address the urgent need for fall detection among the elderly. This system is designed to build upon previous research and achieve the following objectives:

1. accurately detect fall events using web cameras, with an accuracy of more than 96%,
2. establish a notification system that can promptly transmit fall incident information to medical staff or family members within 0.5 s, and
3. enable real-time sensing to ensure that the elderly can receive timely assistance after a fall, thereby minimizing potential injuries.

2. Theoretical Foundation

In this section, the image recognition technology of AI open sources and human motion classification methods will be discussed. This discussion aims to facilitate a better understanding for readers of the Lightweight AlphaPose(6) model proposed in this study and the human motion classification model. These are crucial for real-time detection capabilities, as the aforementioned AI algorithms and models help ensure that fall events can immediately trigger notification processes, providing prompt responses and reducing potential harm.

2.1 Skeleton pose detection

Skeleton pose detection is a crucial task in the field of computer vision, focusing on accurately identifying the posture and movements of human bodies in images. This technology finds applications in various domains, including but not limited to medical image processing,(7) motion analysis,(8) virtual reality,(9) and security surveillance.(10) Skeleton pose detection methods are primarily categorized into bottom-up and top-down approaches. In the top-down approach, algorithms first identify the potential locations of humans in the input image using object detection techniques. These locations are then marked using bounding boxes. Subsequently, the image regions within these bounding boxes are fed into networks for human pose estimation, aiming to detect joint points and body segments of humans. Methods such as DeepPose(11) and AlphaPose(12) adopt this approach.

Another bottom-up approach involves first identifying possible joint locations in the image and obtaining feature maps for each key point through a bottom-up network. Subsequently, on the basis of relationships and positions of each key point, the human pose is gradually assembled. Methods such as OpenPose(13) and DensePose(14) operate similarly in this manner. AlphaPose's skeletal pose detection system focuses on real-time multiperson pose estimation. It consists of four components: a Spatial Transformer Network (STN) responsible for extracting each person's image and optimizing bounding box positions, two Single-Person Pose Estimation (SPPE) modules for individual keypoint prediction, cross-matching to obtain possible joint positions, a Spatial Deformer Transformer Network (SDTN) for transforming predicted keypoint locations back to original image coordinates, and Pose Non-maximum Suppression (Pose NMS) to address duplicate predictions caused by multiple bounding boxes. AlphaPose achieves 75 mAP on the Common Objects in Context dataset (COCO dataset) and 82.1 mAP on the MPii dataset. Compared with OpenPose and the method of Newell *et al.*,(15) AlphaPose's skeletal pose detection effect demonstrates superior accuracy and operates at 23 frames per second.(16) Therefore, we selected AlphaPose as the skeletal pose detection method.

AlphaPose is based on deep learning techniques, specifically leveraging CNNs. CNNs are particularly suited for tasks in image processing and computer vision, as they excel at learning hierarchical representations of visual data. Key algorithms and components associated with AlphaPose include the following:

1. Residual Network (ResNet): This is a deep CNN architecture renowned for its ability to effectively train very deep networks using residual blocks. AlphaPose utilizes ResNet as part of its backbone to extract features from input images.
2. Region-based CNN (R-CNN): R-CNN is a family of object detection algorithms, including Faster R-CNN, which is used by AlphaPose. Faster R-CNN integrates a Region Proposal Network (RPN) with CNN to efficiently generate region proposals and classify objects within these proposals.
3. Multiperson Pose Estimation: AlphaPose focuses on estimating poses for multiple individuals, involving the detection of multiple persons in images or video frames and accurately localizing their body joints (keypoints) such as shoulders, elbows, wrists, hips, knees, and ankles.
4. Keypoint Localization: A specific task in AlphaPose involves predicting the coordinates of these keypoints for each person detected in the image. This is achieved through a combination of CNN-based feature extraction followed by keypoint regression or classification.

AlphaPose integrates these deep learning technologies, including CNNs (such as ResNet for feature extraction) and object detection frameworks (such as Faster R-CNN for person detection), to perform real-time and accurate multiperson pose estimation. These algorithms and frameworks collectively enable AlphaPose to achieve robust pose estimation from visual data.

2.2 Classifier

A classifier is a model or algorithm in machine learning used to categorize data into different classes on the basis of their features. This classification process spans various fields, from image recognition, natural language processing,(17) financial forecasting,(18) and medical diagnosis (19) to social media analysis(20) and product recommendations.(21) The core objective of classifiers is to automatically identify data features through analysis and learning patterns in the data, assigning them to appropriate categories for practical applications. Classifiers commonly include Support Vector Machines,(22) Random Forests,(23) CNN,(24) and Recurrent Neural Networks,(25) although these methods may not necessarily discern motion features in human pose.

Traditional CNNs are widely applied in processing image data with regular structures, such as static pictures. However, their performance may be limited for nonregular image data. In contrast, Graph Convolutional Networks (GCNs)(26) excel in handling data with non-Euclidean structures, especially irregular image data. GCNs effectively capture complex relationships in data, making them an ideal choice for processing irregular image data. Additionally, GCNs demonstrate stronger generalization capabilities, which contribute to improving model training efficiency. GCNs operate on graph-structured data and involve a convolution operation aimed at learning node representations. The basic theoretical equation describing GCN is as follows:

Given a graph $G = (V, E)$, V is the set of vertices (nodes) representing entities or feature points in the graph. E is the set of edges, representing connections between nodes. We assume that each node $v_i \in V$ has a feature representation x_i, which denotes the feature vector of the node.

1. Adjacency Matrix: Each node aggregates information from its neighboring nodes for graph representation learning. In typical usage, matrix A is employed to represent a $|V| \times |V|$ matrix, where A_{ij} indicates whether there is a connection from node v_i to node v_j (usually represented as 1 or 0).
2. Node Feature Matrix: In notation with X, it represents a $|V| \times d|V|$ matrix, where d is the dimension of the node feature vectors. X_i denotes the feature vector x_i of node v_i.
3. Graph Convolution Operation: The convolutional operation on node v_i takes into account the features of its neighbors and possibly its own features to compute an updated representation. The core operation of GCNs convolves the node feature matrix X with the adjacency matrix A.

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right) \tag{1}$$

Here, $H^{(l)}$ is the node representation matrix at layer l. $W^{(l)}$ is the weight matrix at layer l. $\tilde{A} = A + I$ is the adjacency matrix A with added self-loops to ensure each node has at least one connection. $\tilde{D}$ is the degree matrix of $\tilde{A}$. σ is a nonlinear activation function, such as ReLU. This equation describes how GCNs leverage the structural information of the graph (via the adjacency matrix A) to update node feature representations. By stacking multiple layers of such graph convolution operations, high-level node representations can be progressively extracted for various tasks involving graph-structured data, including node classification, link prediction, and graph classification.(26)

Yan *et al.* proposed the Spatial Temporal GCNs (ST-GCNs) for Skeleton-Based Action Recognition in 2018.(27) One of the main reasons for choosing ST-GCNs in their study is their capability in action recognition, which is commonly applied to recognize human body poses during motion; considering the need for real-time action detection, especially in dynamic environments where the human body is typically in motion rather than stationary, it is essential to develop efficient and accurate detection methods. Spatial graph convolution is a technique that extends traditional convolution operations to handle graph-structured data. This method generalizes convolution from regular 2D grid images to irregular graph structures, where nodes represent entities (such as human joints) and edges represent relationships between nodes. This allows for effective feature learning on graph data, capturing complex relationships between nodes in a graph.

In the 2D convolution described by Yan *et al.*,(27) the sampling function $p(h, w)$ defines neighboring pixels relative to a center pixel, with a fixed spatial order on a regular grid. This fixed spatial arrangement allows the weight function to be implemented by indexing a tensor of fixed dimensions. In contrast, graphs do not have such a fixed grid structure for nodes and edges, necessitating a redefinition of sampling and weight functions to fit the graph structure.

2.2.1 Sampling function

In graph convolution, the sampling function p is used to define the neighbor set of a node. For a node v_{ti}, its neighbor set $B(v_{ti})$ includes all nodes connected to v_{ti}. Specifically, the sampling function $p(v_{ti}, v_{tj})$ can be expressed as $p(v_{ti}, v_{tj}) = v_{tj}$. This means that for each neighbor v_{tj} of v_{ti}, the sampling function directly maps to these neighboring nodes.

2.2.2 Weight function

The weight function in graph convolution is challenging to design because nodes in a graph lack a fixed spatial order, unlike grid-structured data such as images, where convolutional filters operate on a consistent neighborhood structure. To define the weight function, we typically partition the neighbor set $B(v_{ti})$ into several subsets, each with a numerical label. This allows the weight function to be implemented by indexing a (c, K) dimensional tensor, where K is the number of subsets and c is the feature dimension. Specifically, the weight function w can be expressed as $w(v_{ti};v_{tj}) = w_0(l_{ti}(v_{tj}))$, where l_{ti} is a function that maps node v_{tj} to its subset label, and w_0 is the corresponding weight tensor.

2.2.3 Graph convolution equation

Combining the above sampling and weight functions, Yan *et al.* proposed the graph convolution operation,(27) which can be rewritten as

$$f_{out}(v_{ti}) = \sum\nolimits_{v_{tj} \in B(v_{ti})} \frac{1}{Z_{ti}(v_{tj})} f_{in}(p(v_{ti}, v_{tj})) \cdot w(v_{ti}, v_{tj}), \tag{2}$$

where $B(v_{ti})$ is the neighbor set of node v_{ti}, $Z_{ti}(v_{tj})$ is a normalization factor used to handle the weight of node v_{tj}, which is typically the reciprocal of the number of neighbors, $fin(p(v_{ti};v_{tj}))$ is the input feature obtained from node v_{tj}, and $w(v_{ti};v_{tj})$ is the computed weight.

The spatial graph convolution extends traditional convolution operations to graph structures. This method redefines the sampling and weight functions to accommodate the unique characteristics of graphs. The sampling function defines the neighbor set of nodes, whereas the weight function determines the weights by mapping to subset labels. The resulting graph convolution equation combines these elements to perform the weighted aggregation of features for each node, capturing relational and structural information in the graph.

The ST-GCN is particularly effective in dynamic environments, meeting the demands for efficiency and accuracy in action recognition. The integration of the aforementioned models and methods enables this research to achieve superior performance when handling irregular action image data, while ensuring that human body movements, particularly in fall scenarios, are successfully detected. These considerations contribute to the applicability of our system, especially in dynamic and variable scenes. Therefore, we adopted ST-GCN as the theoretical foundation for the system.

3. System Construction Planning

3.1 System development

As previously mentioned, AlphaPose is a powerful multiperson pose detection system that focuses on the real-time detection and recognition of human poses in complex scenes. Its main steps and technologies include the following:

STN: The STN is a module capable of learning spatial transformations on input images. Its purpose is to automatically learn appropriate transformations to standardize the space so that the subsequent pose estimation network can learn more effectively. In practical applications, STN extracts the regions of each person from the image and optimizes each person's bounding box position by performing spatial transformations such as cropping, scaling, and rotating. This helps to reduce background noise and focus on the areas of interest, thereby improving the accuracy of subsequent pose detection.

SPPE: The SPPE module in AlphaPose is responsible for predicting the pose of each person, generating keypoint heatmaps, and returning the keypoint positions for each person. These networks typically use CNNs to predict the keypoints. AlphaPose usually employs two single-person pose estimation networks to predict the keypoint positions of each person separately. These two networks provide redundancy and cross-validation to enhance the accuracy of keypoint prediction.

SDTN: SDTN converts the predicted keypoint positions from the transformed coordinate system back to the original image coordinate system. This allows the single-person pose estimation results to be mapped back onto the original image, accurately displaying the keypoints' positions in the original image.

Pose NMS: Pose NMS is a postprocessing technique used to address the issue of duplicate predictions due to overlapping bounding boxes. This step helps to remove redundant bounding boxes and ensures that each person's pose is detected correctly and uniquely, avoiding repeated detections caused by multiple overlapping bounding boxes.

In this study, we integrated AlphaPose with ST-GCN to further develop a fall detection and real-time notification system. When obtaining the raw image through a computer's webcam, AlphaPose is used to detect each person's pose, which is then processed by the ST-GCN classifier to predict each person's actions (standing, lying down, or falling). If a fall is detected, the system sends a notification to Line Bot via the IFTTT communication service. The Line Bot then informs the user about the fall situation. The system architecture is shown in Fig. 1.

We use AlphaPose, which was developed on the basis of the YOLOv3 model, to detect the positions of individuals within the webcam view and obtain each person's skeletal pose. Then, we employ the ST-GCN model to predict each person's actions (standing, lying down, or falling).

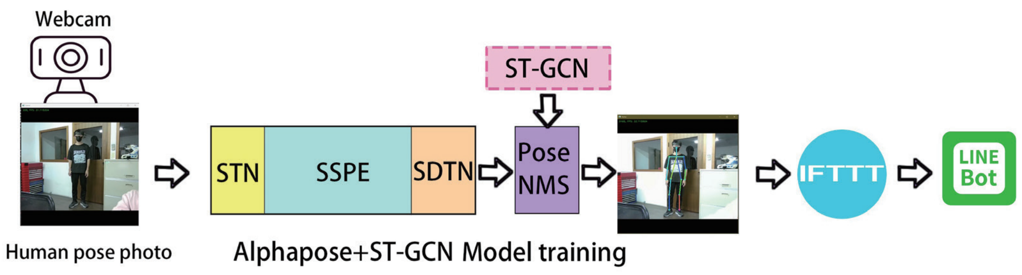


Fig. 1. (Color online) System architecture in this study.

If a fall occurs, the system will notify the user of the fall situation through Line Bot (see the system flow chart in Fig. 2).

3.2 Environment setup

The model training and testing equipment used in this study is a laptop with an Intel Core i7-8565U 1.80 GHz processor and an NVIDIA GeForce MX230 graphics card. Additional information regarding the related hardware and software versions of the development environment is shown in Table 1.

4. Experimental Phase

4.1 Dataset normalization

In this study, we utilized the Le2i Fall Dataset, which is designed for fall detection and prevention research, focusing on using machine learning and computer vision techniques to identify and analyze fall events. This dataset was developed by the Laboratoire d'Informatique en Robotique et Vision laboratory in France, with the primary goal of improving the accuracy and effectiveness of fall detection systems. Its main features include the following:

The dataset primarily consists of video clips captured in a controlled environment. These clips capture various human activities, including falls and normal activities. Each video clip is annotated in detail, including the start and end times of falls, as well as annotations for other activities. This allows researchers to accurately label fall events and conduct analysis. The video clips in the dataset are typically recorded in a laboratory setting, which allows for control over lighting and background conditions, facilitating feature extraction and model training. The Le2i Fall Dataset includes a variety of fall types and human activities, making it useful for training models to recognize different fall patterns and distinguish between falls and non-falls. This dataset is widely used in fall detection system research, particularly in the fields of elderly health care and safety monitoring. By analyzing video data, researchers can develop more effective fall detection and warning systems.

However, using the Le2i Fall Dataset for research involves addressing several challenges, such as how to accurately extract human posture and motion features from the videos and how to handle variations in different environments and backgrounds. The Le2i Fall Dataset is an

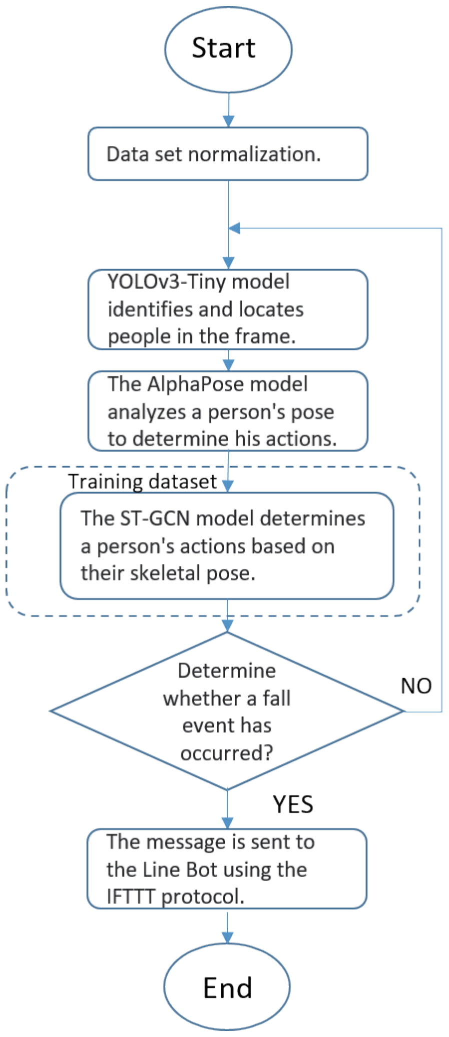

Start
Data set normalization.
YOLOv3-Tiny model identifies and locates people in the frame.
The AlphaPose model analyzes a person's pose to determine his actions.
Training dataset
The ST-GCN model determines a person's actions based on their skeletal pose.
Determine whether a fall event has occurred?
NO
YES
The message is sent to the Line Bot using the IFTTT protocol.
End

Fig. 2. (Color online) System flow chart of this study.

Table 1
Hardware and software version information for AI model training in this study.

	Version, Model, or Specification
Software Information	Integrated Development Environment (IDE): Anaconda 3 + PyCham 2023.1.2
	Python Version: 3.7.2
	PyTorch Version: 1.11.0
	cuDNN Version:8,2,1
Hardware Information	Processor: Intel(R) Core(TM) i7-8565U CPU @ 1.80 GHz
	Memory: 12 GB/ Hard drive: SSD 250 GB + HDD 1 TB
	Graphics Card: NVIDIA GeForce MX230

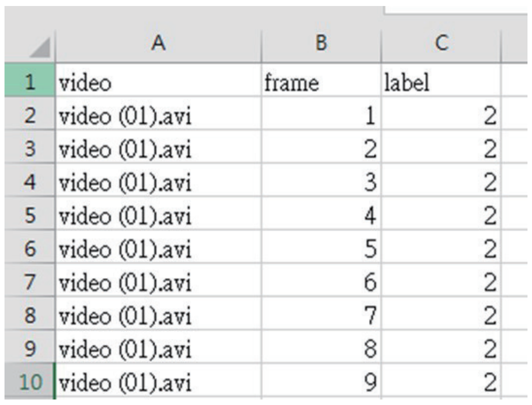
important tool for fall detection and prevention research, helping researchers improve the accuracy and reliability of related systems. The Le2i Fall Dataset may also be stored in academic databases or data sharing platforms such as Kaggle, University of California Machine Learning

Repository, or GitHub. The dataset contains a total of 191 videos, with actions classified into seven categories: Standing, Walking, Sitting, Lying Down, Stand Up, Sit Down, and Fall Down. Initially, the dataset is converted from videos to comma-separated value (CSV) files, with columns for video name, frame number, and action label, representing the video's name, the specific frame number, and the action classification for that frame, respectively, with actions annotated for each frame (see Fig. 3).

Next, the 39 key joints of the person in each frame are labeled and recorded in a CSV file. The CSV file is then converted into a dataset Pickle (PKL) file. Once the dataset is prepared and split into training and validation sets, the ST-GCN model is trained for skeleton-based pose recognition. Upon successful training, the model weights will be saved in a '.pth' file. We use create_dataset_1.py to determine the total number of frames for each video and use this frame count to differentiate each row of data. Then, we add a label column and manually annotate the human actions (based on the array of actions, such as 1 = Standing, 2 = Walking, and so forth). The Python syntax for human action classification is defined as follows:

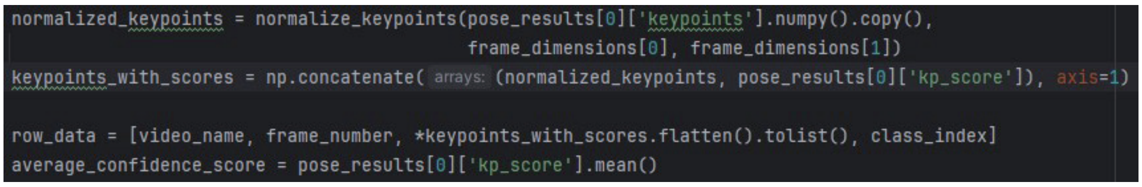
class_name = {'Standing', 'Walking', 'Sitting', 'Lying Down', 'Stand Up', 'Sit Down', 'Fall Down'} # label.

To normalize the Le2i Fall Dataset, we start by using the Python script (create_dataset_1.py) to determine the total number of frames for each video and categorize the data based on frame count. We add a label column and manually annotate human actions, such as 'Standing', 'Walking', and 'Sitting', according to the actions included in the array. Next, we use the Python script (create_dataset_2.py, see Fig. 4) to identify the positions of each human keypoint and manually remove rows where keypoints could not be detected to generate a dataset of human joint coordinates (as illustrated in Fig. 5). Finally, we employ the Python script (create_dataset_3.py; see Fig. 6) to convert the resulting CSV file (Fig. 5) into a PKL file, paying special attention to smoothing functions that might merge label categories (e.g., combining 'Standing' with 'Stand Up').



	A	B	C
1	video	frame	label
2	video (01).avi	1	2
3	video (01).avi	2	2
4	video (01).avi	3	2
5	video (01).avi	4	2
6	video (01).avi	5	2
7	video (01).avi	6	2
8	video (01).avi	7	2
9	video (01).avi	8	2
10	video (01).avi	9	2

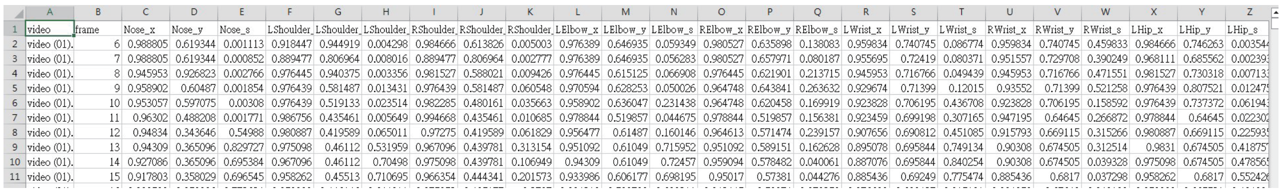
Fig. 3. (Color online) CSV file of dataset.



```
normalized_keypoints = normalize_keypoints(pose_results[0]['keypoints'].numpy().copy(),
                                           frame_dimensions[0], frame_dimensions[1])
keypoints_with_scores = np.concatenate( arrays: (normalized_keypoints, pose_results[0]['kp_score']), axis=1)

row_data = [video_name, frame_number, *keypoints_with_scores.flatten().tolist(), class_index]
average_confidence_score = pose_results[0]['kp_score'].mean()
```

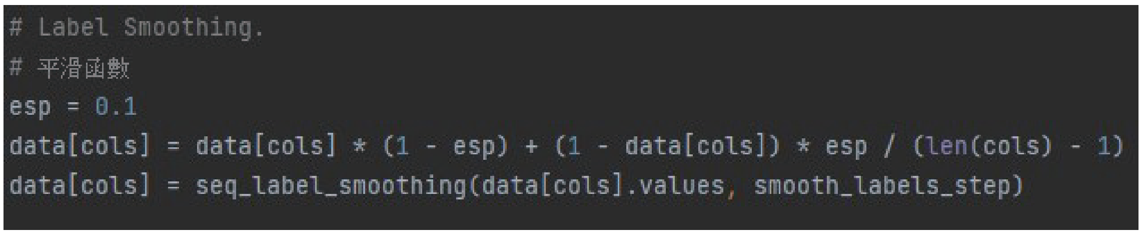
Fig. 4. (Color online) Python script (create_dataset_2.py).



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	video	frame	Nose_x	Nose_y	Nose_s	LShoulder_	LShoulder_	LShoulder_	RShoulder_	RShoulder_	RShoulder_	LElbow_x	LElbow_y
2	video (01).	6	0.988805	0.619344	0.001113	0.918447	0.944919	0.004298	0.984666	0.613826	0.005003	0.976389	0.646935
3	video (01).	7	0.988805	0.619344	0.000852	0.889477	0.806964	0.008016	0.889477	0.806964	0.002777	0.976389	0.646935
4	video (01).	8	0.945953	0.926823	0.002766	0.976445	0.940375	0.003356	0.981527	0.588021	0.009426	0.976445	0.615125
5	video (01).	9	0.958902	0.60487	0.001854	0.976439	0.581487	0.013431	0.976439	0.581487	0.060548	0.970594	0.628253
6	video (01).	10	0.953057	0.597075	0.00308	0.976439	0.519133	0.023514	0.982285	0.480161	0.035663	0.958902	0.636047
7	video (01).	11	0.96302	0.488208	0.001771	0.986756	0.435461	0.005649	0.994668	0.435461	0.010685	0.978844	0.519857
8	video (01).	12	0.94834	0.343646	0.54988	0.980887	0.419589	0.065011	0.97275	0.419589	0.061829	0.956477	0.61487
9	video (01).	13	0.94309	0.365096	0.829727	0.975098	0.46112	0.531959	0.967096	0.439781	0.313154	0.951092	0.61049
10	video (01).	14	0.927086	0.365096	0.695384	0.967096	0.46112	0.70498	0.975098	0.439781	0.106949	0.94309	0.61049
11	video (01).	15	0.917803	0.358029	0.696545	0.958262	0.45513	0.710695	0.966354	0.444341	0.201573	0.933986	0.606177

	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	LElbow_s	RElbow_x	RElbow_y	RElbow_s	LWrist_x	LWrist_y	LWrist_s	RWrist_x	RWrist_y	RWrist_s	LHip_x	LHip_y	LHip_s
2	0.059349	0.980527	0.635898	0.138083	0.959834	0.740745	0.086774	0.959834	0.740745	0.459833	0.984666	0.746263	0.00354
3	0.056283	0.980527	0.657971	0.080187	0.955695	0.72419	0.080371	0.951557	0.729708	0.390249	0.968111	0.685562	0.00239
4	0.066908	0.976445	0.621901	0.213715	0.945953	0.716766	0.049439	0.945953	0.716766	0.471551	0.981527	0.730318	0.00713
5	0.050026	0.964748	0.643841	0.263632	0.929674	0.71399	0.12015	0.93552	0.71399	0.521258	0.976439	0.807521	0.01247
6	0.231438	0.964748	0.620458	0.169919	0.923828	0.706195	0.436708	0.923828	0.706195	0.158592	0.976439	0.737372	0.06194
7	0.044675	0.978844	0.519857	0.156381	0.923459	0.699198	0.307165	0.947195	0.64645	0.266872	0.978844	0.64645	0.02230
8	0.160146	0.964613	0.571474	0.239157	0.907656	0.690812	0.451085	0.915793	0.669115	0.315266	0.980887	0.669115	0.22593
9	0.715952	0.951092	0.589151	0.162628	0.895078	0.695844	0.749134	0.90308	0.674505	0.312514	0.9831	0.674505	0.41875
10	0.72457	0.959094	0.578482	0.040061	0.887076	0.695844	0.840254	0.90308	0.674505	0.039328	0.975098	0.674505	0.47856
11	0.698195	0.95017	0.57381	0.044276	0.885436	0.69249	0.775474	0.885436	0.6817	0.037298	0.958262	0.6817	0.55242

Fig. 5. (Color online) Output of Python script (create_dataset_2.py): human joint coordinates.



```
# Label Smoothing.
# 平滑函數
esp = 0.1
data[cols] = data[cols] * (1 - esp) + (1 - data[cols]) * esp / (len(cols) - 1)
data[cols] = seq_label_smoothing(data[cols].values, smooth_labels_step)
```

Fig. 6. (Color online) Python script (create_dataset_3.py).

4.2 Training dataset

In the AI training, the data normalization phase is crucial to ensure data consistency and accuracy. During this phase, we convert CSV files into dataset PKL files to facilitate subsequent processing and model training.

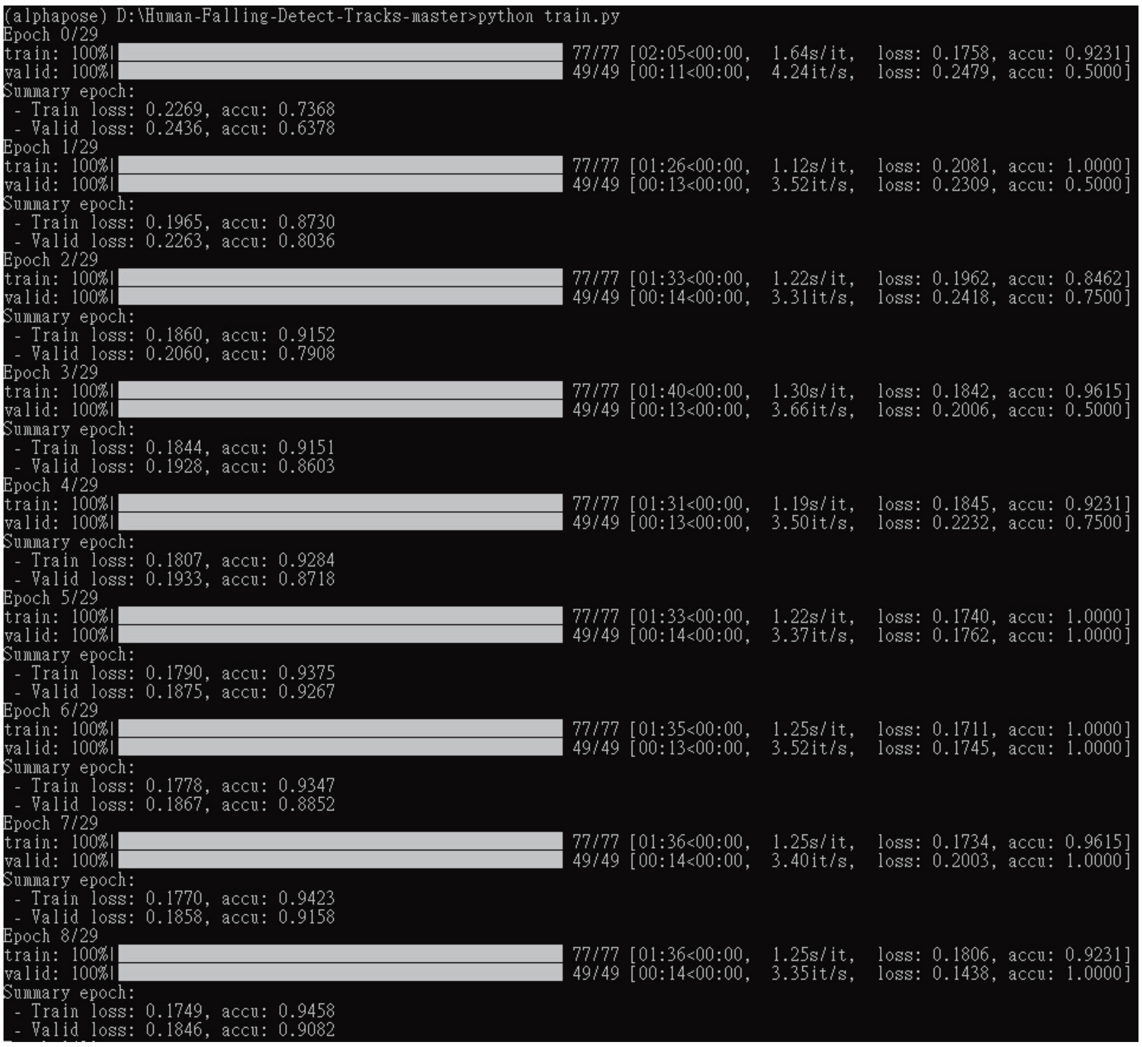
4.2.1 Data normalization phase

1. CSV File Conversion: Initially, we convert the raw CSV files into dataset PKL files. This step involves organizing, cleaning, and transforming CSV-format data into Python's PKL format. The PKL file format offers efficient storage and retrieval methods, making it suitable for handling large datasets. Converting CSV files to PKL files is a common data processing step, especially when dealing with large volumes of data. PKL files are a Python-specific serialization format designed for storing and rapidly retrieving Python objects. This step enhances the efficiency of subsequent data processing.

2. Handling Multiple Datasets: When dealing with multiple datasets, we need to repeatedly perform the CSV-to-PKL file conversion to ensure that each dataset undergoes a consistent processing workflow. This guarantees compatibility between all datasets and maintains overall data consistency.

4.2.2 Entering the training phase

1. Model Selection: After completing data normalization, we proceed to the training phase as shown in Fig. 2. In this stage, we use AlphaPose for training (see Fig. 7). YOLOv3 is a popular real-time object detection model, and AlphaPose extends and modifies it. AlphaPose is an open-source pose estimation tool developed on the basis of the YOLOv3 model, focusing on efficient and accurate human pose estimation. AlphaPose provides an accurate identification of human keypoints, which are the foundation for subsequent action recognition and analysis.



```
(alphapose) D:\Human-Falling-Detect-Tracks-master>python train.py
Epoch 0/29
train: 100%|██████████████████████████████| 77/77 [02:05<00:00,  1.64s/it,  loss: 0.1758, accu: 0.9231]
valid: 100%|██████████████████████████████| 49/49 [00:11<00:00,  4.24it/s,  loss: 0.2479, accu: 0.5000]
Summary epoch:
 - Train loss: 0.2269, accu: 0.7368
 - Valid loss: 0.2436, accu: 0.6378
Epoch 1/29
train: 100%|██████████████████████████████| 77/77 [01:26<00:00,  1.12s/it,  loss: 0.2081, accu: 1.0000]
valid: 100%|██████████████████████████████| 49/49 [00:13<00:00,  3.52it/s,  loss: 0.2309, accu: 0.5000]
Summary epoch:
 - Train loss: 0.1965, accu: 0.8730
 - Valid loss: 0.2263, accu: 0.8036
Epoch 2/29
train: 100%|██████████████████████████████| 77/77 [01:33<00:00,  1.22s/it,  loss: 0.1962, accu: 0.8462]
valid: 100%|██████████████████████████████| 49/49 [00:14<00:00,  3.31it/s,  loss: 0.2418, accu: 0.7500]
Summary epoch:
 - Train loss: 0.1860, accu: 0.9152
 - Valid loss: 0.2060, accu: 0.7908
Epoch 3/29
train: 100%|██████████████████████████████| 77/77 [01:40<00:00,  1.30s/it,  loss: 0.1842, accu: 0.9615]
valid: 100%|██████████████████████████████| 49/49 [00:13<00:00,  3.66it/s,  loss: 0.2006, accu: 0.5000]
Summary epoch:
 - Train loss: 0.1844, accu: 0.9151
 - Valid loss: 0.1928, accu: 0.8603
Epoch 4/29
train: 100%|██████████████████████████████| 77/77 [01:31<00:00,  1.19s/it,  loss: 0.1845, accu: 0.9231]
valid: 100%|██████████████████████████████| 49/49 [00:13<00:00,  3.50it/s,  loss: 0.2232, accu: 0.7500]
Summary epoch:
 - Train loss: 0.1807, accu: 0.9284
 - Valid loss: 0.1933, accu: 0.8718
Epoch 5/29
train: 100%|██████████████████████████████| 77/77 [01:33<00:00,  1.22s/it,  loss: 0.1740, accu: 1.0000]
valid: 100%|██████████████████████████████| 49/49 [00:14<00:00,  3.37it/s,  loss: 0.1762, accu: 1.0000]
Summary epoch:
 - Train loss: 0.1790, accu: 0.9375
 - Valid loss: 0.1875, accu: 0.9267
Epoch 6/29
train: 100%|██████████████████████████████| 77/77 [01:35<00:00,  1.25s/it,  loss: 0.1711, accu: 1.0000]
valid: 100%|██████████████████████████████| 49/49 [00:13<00:00,  3.52it/s,  loss: 0.1745, accu: 1.0000]
Summary epoch:
 - Train loss: 0.1778, accu: 0.9347
 - Valid loss: 0.1867, accu: 0.8852
Epoch 7/29
train: 100%|██████████████████████████████| 77/77 [01:36<00:00,  1.25s/it,  loss: 0.1734, accu: 0.9615]
valid: 100%|██████████████████████████████| 49/49 [00:14<00:00,  3.40it/s,  loss: 0.2003, accu: 1.0000]
Summary epoch:
 - Train loss: 0.1770, accu: 0.9423
 - Valid loss: 0.1858, accu: 0.9158
Epoch 8/29
train: 100%|██████████████████████████████| 77/77 [01:36<00:00,  1.25s/it,  loss: 0.1806, accu: 0.9231]
valid: 100%|██████████████████████████████| 49/49 [00:14<00:00,  3.35it/s,  loss: 0.1438, accu: 1.0000]
Summary epoch:
 - Train loss: 0.1749, accu: 0.9458
 - Valid loss: 0.1846, accu: 0.9082
```

Fig. 7. Dataset training process in this study.

2. ST-GCN Model: ST-GCN is a deep learning model designed to process spatiotemporal graph data, capable of learning spatial and temporal features. It is specialized for extracting spatial and temporal features of human actions from video or sequence data.

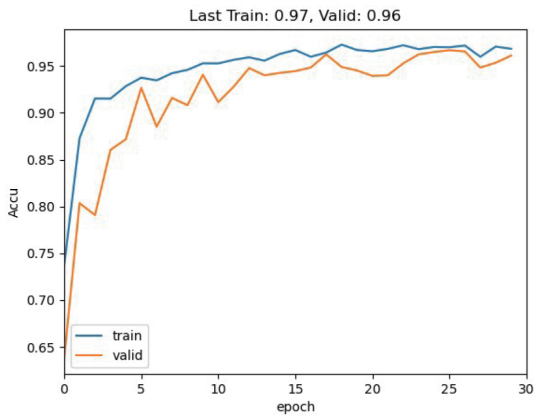
This training process aims to optimize the performance of human pose estimation and action recognition models through the deep learning and analysis of data in PKL files. By using AlphaPose for keypoint estimation and ST-GCN for action classification and analysis, we achieve the precise understanding and prediction of human movements.

4.3 Training results

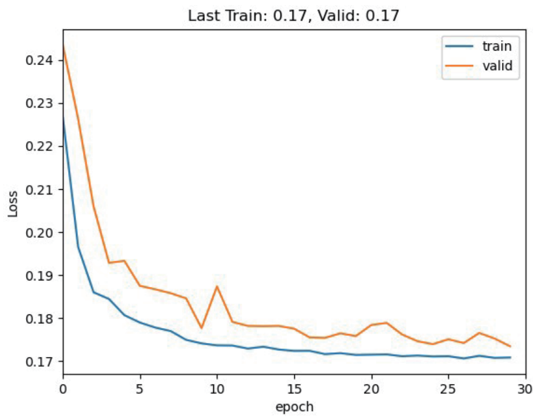
After the training process depicted in Fig. 7, it is evident that accuracy improves significantly with an increasing number of training epochs. Around the 10th training epoch, the accuracy consistently exceeds 96% [see Fig. 8(a)]. Similarly, the loss rate shows a notable decrease with more training epochs, dropping below 0.18 after approximately the 11th training epoch [see Fig. 8(b)]. Analyzing the confusion matrix results (see Fig. 9), we observe that the probability of accurately identifying each category is very high, except for the "walking" category, which sometimes overlaps with the "sitting" category.

4.4 Execution results

After running it in this study, the program begins detecting human bounding boxes and skeletal poses. On the basis of the recognized actions from the skeletal poses, labels are annotated above the bounding boxes. The text information displayed above includes the confidence level of the recognition and the interpreted result. If the action detected is "Lying Down," the action label is displayed in orange [see Fig. 10(a)]. If the action is "Fall Down," the action label is shown in red [see Fig. 10(b)]. Additionally, for "Fall Down" actions, a notification is sent every 30 frames using IFTTT, which forwards the message to Line Bot to alert the relevant personnel.



(a)



(b)

Fig. 8. (Color online) (a) Accuracy obtained from training using Le2i fall dataset in this study. (b) Loss rate obtained from training using Le2i fall dataset in this study.

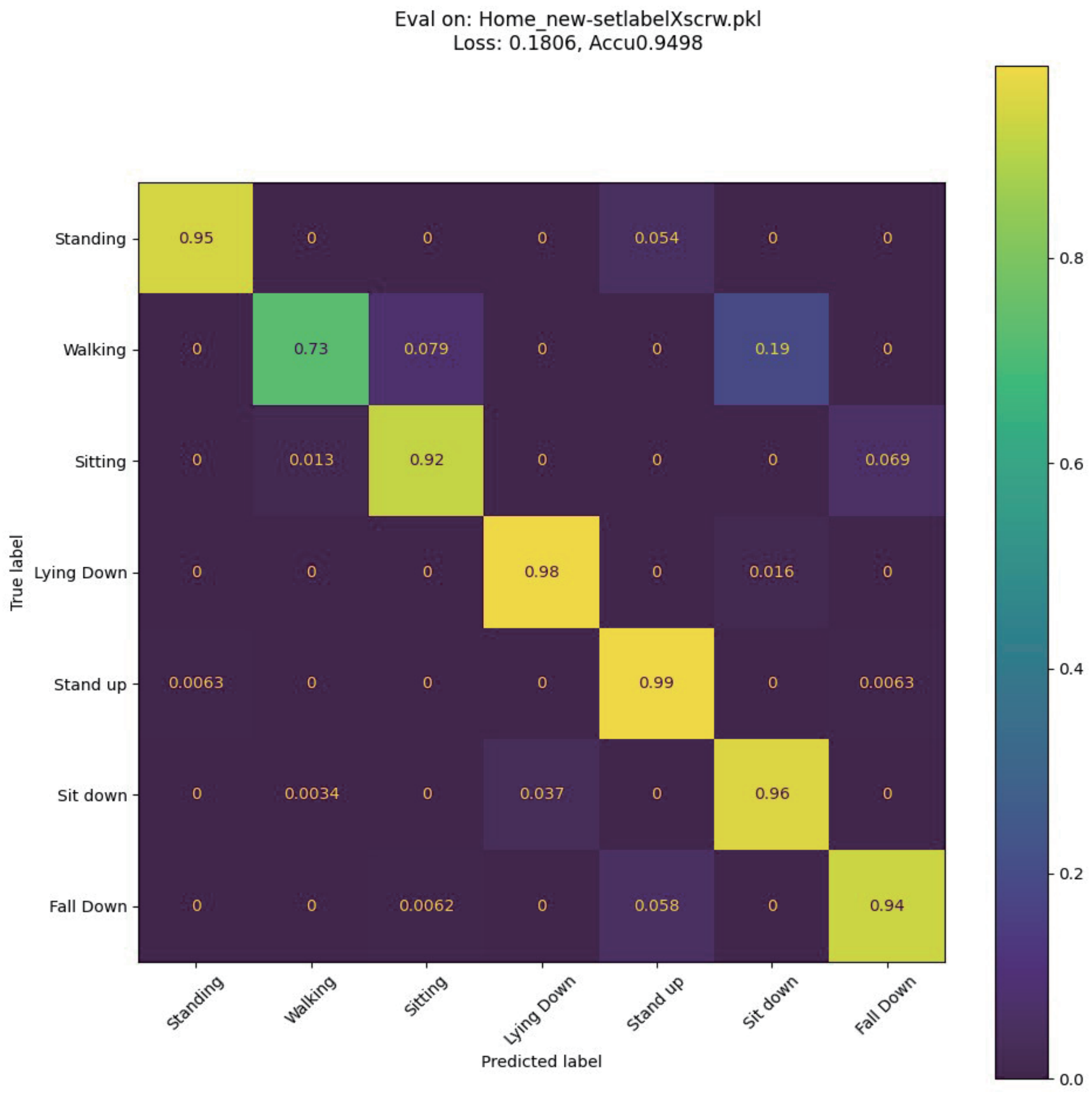
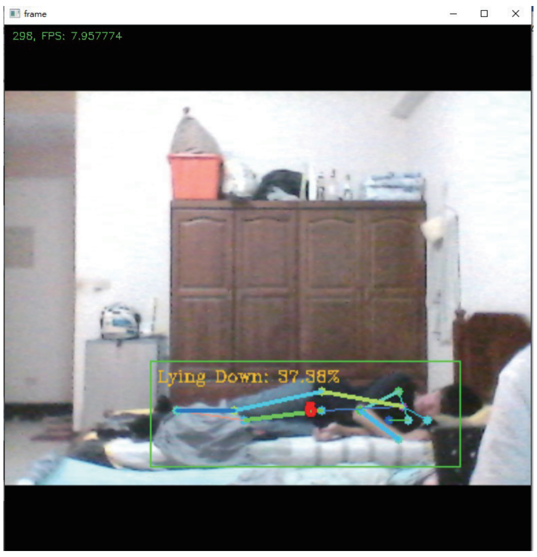
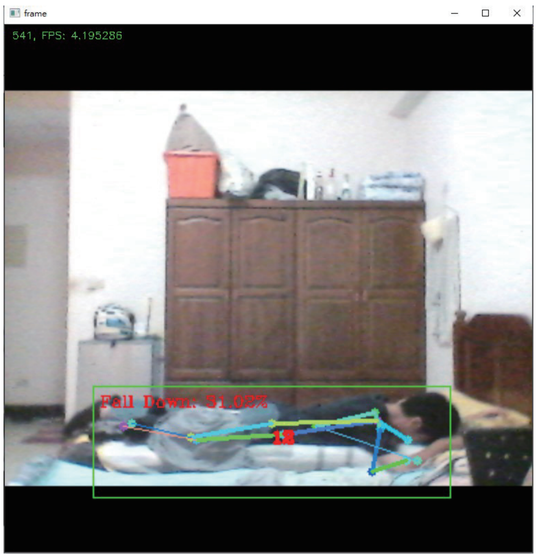


Fig. 9. (Color online) Confusion matrix output from training phase of this study.

In this study, the Real-time Fall Detection and Reporting System developed will, at this time, send notification messages to the relevant family members or caregivers of the elderly individual in the image based on the AI interpretation results (as shown in Fig. 11). The system will provide different notification messages depending on whether the warning is orange or red. Below is the code for sending messages to Line Bot:

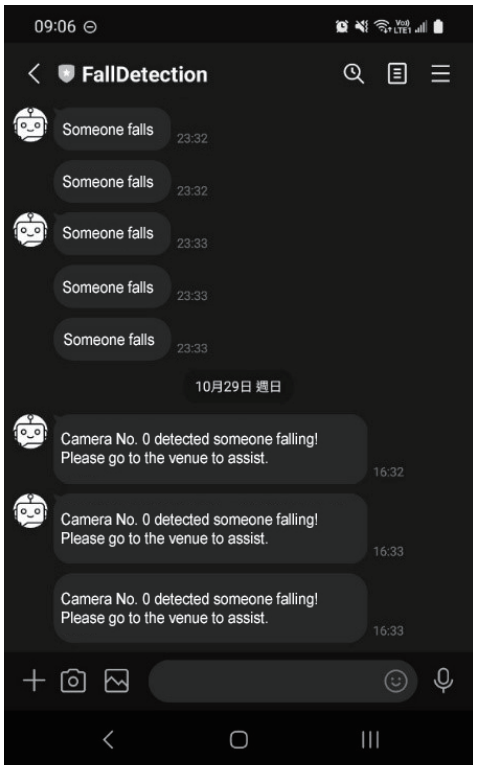


(a)



(b)

Fig. 10. (Color online) (a) Experimental result where AI identifies image as "Lying Down," displaying orange warning and showing accuracy. (b) Experimental result where AI identifies image as "Fall Down," displaying red warning and showing accuracy.



09:06

FallDetection

Someone falls 23:32

Someone falls 23:32

Someone falls 23:33

Someone falls 23:33

Someone falls 23:33

10月29日 週日

Camera No. 0 detected someone falling!
Please go to the venue to assist. 16:32

Camera No. 0 detected someone falling!
Please go to the venue to assist. 16:33

Camera No. 0 detected someone falling!
Please go to the venue to assist. 16:33

Fig. 11. This system sends information about the elderly person falling to Line Bot.

```
camera_source = args.camera
message = TextSendMessage(text=camera_source+' Someone has fallen down at camera No.
#. Please send someone to handle the situation as soon as possible.')

if action_name != 'Fall Down' and action_name != 'Lying Down':
    count = 0
else:
    if action_name == 'Fall Down':
    clr = (255, 0, 0)
    count += 1
    if count >= 30 and count % 30 == 0:
            line_bot_api.broadcast(messages=[message])
elif action_name == 'Lying Down':
    clr = (255, 200, 0)
    count += 1
```

5. Conclusions

The goal of this research is to develop an effective fall detection and instant alert system by integrating web cameras as sensors, machine learning technology, and AI. The system focuses on human skeleton posture recognition, motion detection, and real-time reporting. By employing machine learning techniques, the system can accurately detect falls and promptly notify medical staff or family members, thereby minimizing the potential consequences of a fall.

In this study, machine learning plays a pivotal role in the system's ability to detect, analyze, and report falls. The AI-based models employed include the following:

1. YOLOv3-Tiny: This model is used to localize individuals within the frame, providing the initial detection of human presence.
2. AlphaPose: This model detects the individual's skeletal pose, enabling the system to understand their movements and identify potentially dangerous fall events.
3. ST-GCN: This model recognizes the actions of the individual on the basis of the detected skeletal poses, helping the system distinguish between normal movements and falls.

Machine learning techniques, particularly the integration of these models, ensure that the system's accuracy remains consistently above 96%, demonstrating the reliability of the fall detection and notification process. These models work together to provide instant fall detection and immediate reporting, offering a timely response that can help prevent further injury.

Furthermore, the system ensures effective notification. When a fall is detected, Line Bot sends a message via IFTTT technology to quickly notify relevant personnel. This ensures that falls are addressed promptly, minimizing potential injuries.

The novelty of this study lies in the integration of AlphaPose with ST-GCN for real-time fall detection and notification. The system achieves high accuracy (above 96%) in identifying falls by combining skeletal pose detection and action classification. The key innovative contributions include the following:

1. Real-Time Detection and Notification: The system uses web cameras and advanced AI techniques to detect falls instantly and notify caregivers or family members within 0.5 s, which is critical for timely intervention.
2. Use of Lightweight and Efficient Models: AlphaPose, known for its accuracy in pose estimation, and ST-GCN, which is effective for action recognition in dynamic environments, are combined to ensure robust and efficient fall detection.
3. High Adaptability and Generalization: Unlike many existing systems that rely heavily on specific datasets or controlled environments, in this study, we focused on enhancing the model's adaptability and generalization to various conditions.

References

1 World Health Organization: https://www.who.int/news-room/fact-sheets/detail/falls (accessed April 2024).
2 X. G. Li, T. T. Pang, W. X. Liu, and T. F. Wang: Proc. 2017 10th Int. Congr. Image and Signal Process. BioMed. Eng. Inf. (IEEE, 2017) 1. https://doi.org/10.1109/CISP-BMEI.2017.8302004
3 A. Butt, S. Narejo, M. R. Anjum, M. U. Yonus, M. Memon, and A. A. Samejo: W. Pers. Commun. **126** (2022) 1733. https://doi.org/10.1007/s11277-022-09819-3
4 A. Benoit, C. Escriba, D. Gauchard, A. Esteve, and C. Rossi: IEEE Sens. J. **24** (2024) 11829. https://doi.org/10.1109/JSEN.2024.3364249
5 T. Wang, B. Wang, Y. Shen, Y. Zhao, W. Li, K. Yao, X. Liu, and Y. Luo: Measurement **204** (2022) 112104. https://doi.org/10.1016/j.measurement.2022.112104.
6 H.-S. Fang, J. Li, H. Tang, C. Xu, H. Zhu, Y. Xiu, Y. L. Li, and C. W. Lu: IEEE Trans. P. Anal. Mach. Intell. **45** (2023) 7157. https://doi.org/10.1109/TPAMI.2022.3222784
7 C. Arrowsmith, D. Burns, T. Mak, M. Hardisty, and C. Whyne: Sensors **23** (2023) 363. https://doi.org/10.3390/s23010363
8 C. Chen, K. Liu, and N. Kehtarnavaz: J. Real-T. Image Process. **12** (2013) 155. https://doi.org/10.1007/s11554-013-0370-1
9 D. Mehta, H. Rhodin, D. Casas, P. V. Fua, O. Sotnychenko, W. P. Xu, and C. Theobalt: Proc. 2017 Int. Conf. 3D Vision (IEEE, 2017) 506. https://doi.org/10.1109/3DV.2017.00064.
10 J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake: Proc. Conf. Comput. Vision Pattern Recognition (IEEE, 2011) 1297. https://doi.org/10.1109/CVPR.2011.5995316
11 A. Toshev and C. Szegedy: Proc. 2014 IEEE Conf. Comput. Vision Pattern Recognition (IEEE, 2014) 1653. https://arxiv.org/pdf/1312.4659
12 H. S. Fang, S. Xie, Y. W. Tai, and C. Lu: Proc. 2017 IEEE Int. Conf. Comput. Vision (IEEE, 2017) 2334. https://arxiv.org/pdf/1612.00137
13 Z. Cao, G. Hidalgo, T. Simon, S. Wei, and Y. Sheikh: IEEE Trans. Pattern Anal. Mach. Intell. **43** (2021) 172. https://doi.org/10.1109/TPAMI.2019.2929257
14 R. A. Güler, N. Neverova, and I. Kokkinos: Proc. 2018 IEEE/CVF Conf. Comput. Vision Pattern Recognition (IEEE, 2018) 7297–7306. https://doi.org/10.1109/CVPR.2018.00762
15 A. Newell, K. Yang, and J. Deng: Proc. C. Vision–ECCV 2016. L. Notes in C. Science 2016 (Springer, 2016) 9912. https://doi.org/10.1007/978-3-319-46484-8_29
16 M. Chasmai, N. Das, A. Bhardwaj, and R. Garg: SN Comput. Sci. **3** (2022) 475. https://doi.org/10.1007/s42979-022-01376-7
17 J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova: BERT: Proc. 2019 Conf. North Am. Chapter Assoc. Comput. Ling.: Hum. Lang. Technol. (Association for Computational Linguistics, 2019) 4171. https://doi.org/10.18653/v1/N19-1423
18 J. Y. Campbell, J. D. Hilscher, and J. Szilagyi: J. Invest. Manage. **9** (2011) 14. https://ssrn.com/abstract=1829622
19 S. Jambukia, V. Dabhi, and H. Prajapati: Proc. 2015 Int. Conf. Adv. Comput. Eng. Appl. (IEEE, 2015) 714. https://doi.org/10.1109/ICACEA.2015.7164783
20 Y. Chen, Y. Zhou, S. Zhu, and H. Xu: Proc. IEEE Int. Conf. Social Computing (SocialCom) Privacy, Security, Risk and Trust (PASSAT) (IEEE, 2012) 71–80. https://doi.org/10.1109/SocialCom-PASSAT.2012.55
21 G. Khanvilkarl and D. Vora: Int. J. Eng. Technol. **7** (2018) 87. https://doi.org/10.14419/ijet.v7i3.3.14492
22 V. N. Vapnik: Estimation of Dependences Based on Empirical Data (Springer, Heidelberg, 2006) 2nd ed. https://doi.org/10.1007/0-387-34239-7

23 L. Breiman: Mach. Learn. **45** (2001) 5. https://doi.org/10.1023/A:1010933404324
24 Y. Lecun, L. Bottou, Y. Bengio and P. Haffner: Proc. IEEE. (IEEE, 1998) 2278. https://doi.org/10.1109/5.726791
25 D. Rumelhart, G. Hinton, and R. Williams: Nature **323** (1986) 533. https://doi.org/10.1038/323533a0
26 T. N. Kipf and M. Welling: Proc. Int. Conf. Learn. Representations (Curran Associates, Inc., 2017). https://arxiv.org/abs/1609.02907
27 S. Yan, Y. Xiong, and D. Lin: Proc. Thirty-Second AAAI Conf. Artificial Intelligence (AAAI Press, 2018) 7444–7452.