Received 22 July 2024, accepted 11 August 2024, date of publication 14 August 2024, date of current version 23 August 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3443618

APPLIED RESEARCH

Factorized 3D-CNN for Real-Time Fall Detection and Action Recognition on Embedded System

NADHIRA NOOR, (Student Member, IEEE), AND IN KYU PARK, (Senior Member, IEEE)
Department of Electrical and Computer Engineering, Inha University, Incheon 22212, South Korea

Corresponding author: In Kyu Park (pik@inha.ac.kr)

This work was supported in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) grant funded by Korean Government (MSIT) [Artificial Intelligence Convergence Innovation Human Resources Development (Inha University), Foreground and Background Matching 3D Object Streaming Technology Development, and Artificial Intelligence Innovation Hub] under Grant RS-2022-00155915, Grant 2022-0-00981, and Grant 2021-0-02068; and in part by the Inha University Research Grant.

ABSTRACT We present a novel approach for skeleton-based action recognition and fall detection, optimized for real-time performance on embedded devices. Our method employs a factorized 3D convolutional neural network (3D-CNN) to efficiently extract spatiotemporal features from skeletal data. Initially, a 2D convolution layer is applied to capture spatial features from the input skeleton frames. Subsequently, a 1D convolution layer processes these spatial features to model temporal dynamics, effectively reducing the computational complexity compared to traditional 3D-CNN approaches. This factorization enables the creation of a lightweight model that maintains high accuracy while being suitable for deployment on resource-constrained embedded systems. Our approach is particularly advantageous for surveillance applications, such as autonomous driving or monitoring in elderly homes, where real-time action recognition and fall detection are critical for ensuring safety. Experimental results demonstrate that our model achieves high performance in recognizing various actions and detecting falls, highlighting its potential for practical real-world applications.

INDEX TERMS Real-time action recognition, fall detection, skeleton-based action recognition.

I. INTRODUCTION

In real-world scenarios, human action recognition is crucial for applications such as autonomous driving and elderly surveillance, including fall detection. However, this field faces significant challenges due to the computational complexity involved in processing video data, which requires attention to both spatial and temporal features. Existing action recognition methods based on deep learning and RGB frames as input [1], [2] often exhibit heavy computational demands. These methods rely on processing large amounts of pixel data from RGB video frames and utilizing 3D-CNNs to capture the spatiotemporal features. Consequently, current methods are impractical for deployment on embedded devices with limited processing power and memory. Moreover, RGB-based methods can be affected by factors such as background colors and occlusions, particularly during fall detection, where fast movement and occlusion often occur.

Recognizing these limitations, researchers are increasingly turning to skeleton-based approaches, which prioritize capturing the underlying motion dynamics rather than pixel-level details. Skeleton-based methods offer a promising alternative to RGB-based approaches by abstracting away background clutter and focusing solely on the spatial relationships between body joints and human motion. Previous works in this domain, such as PoseConv3D [3], have explored 3D convolutional architectures as the backbone for skeleton-based action recognition. However, such methods remain unsuitable for certain embedded devices, such as those utilizing Rockchip processors, due to their computational requirements and limitations [4]. The challenges of implementing efficient action recognition on embedded devices, particularly for multi-person scenarios, are part of a broader issue in integrating deep learning with IoT systems. A recent

The associate editor coordinating the review of this manuscript and approving it for publication was Joao Neves.

comprehensive survey by Elhanashi et al. [5] highlights both the potential and challenges of deploying deep learning models on resource-constrained IoT devices across various applications, including human activity recognition. Their work underscores the need for efficient, lightweight solutions that can handle complex, real-world scenarios. In line with these challenges, several approaches [6], [7] tailored for embedded deployment tend to sacrifice complexity in favor of efficiency, resulting in reduced recognition accuracy.

To bridge this gap, we propose a novel approach to an efficient skeleton-based action recognition model that can be embedded in real-world devices with limited computational resources and memory footprint, while maintaining high accuracy and total trainable parameters count of less than 1 million. In our methodology, we represent human motion by generating joint heatmaps from detected 2D coordinates of human keypoints across the time frame. Inspired by [8], we propose a method that decomposes the 3D-CNN into a 2D-CNN followed by a 1D-CNN, offering an alternative that works efficiently on embedded devices while effectively capturing both spatial and temporal features. This factorization significantly reduces computational complexity and memory requirements compared to standard 3D-CNNs, as it decreases the number of parameters and operations required. The 2D convolution efficiently captures spatial features and relationships between body joints in each frame, while the subsequent 1D convolution models temporal dynamics across frames, allowing for effective spatiotemporal analysis with lower computational demands. This approach not only makes the model more suitable for embedded devices with limited resources but also enhances real-time performance by reducing the processing time required for each frame. Additionally, this decomposition enables our model to run on embedded devices that may not support full 3D convolutions, further broadening its applicability in resource-constrained environments. Consequently, we can achieve efficient real-time action recognition on embedded devices without sacrificing the accuracy of the model.

This study is an extended version of our preliminary work [9]. While our preliminary work [9] introduced a 2D-CNN approach for single-person action recognition on embedded devices, this extended work makes several significant advancements. Firstly, we transition from a 2D-CNN to a 3D-CNN architecture, allowing for better capture of spatiotemporal features. Secondly, we expand our model's capability to handle multi-person scenarios, making it more applicable to real-world situations. These enhancements, along with more comprehensive experiments and analyses, represent substantial improvements over our initial work.

The contributions of this paper are as follows:

- We propose a lightweight deep learning model utilizing a skeleton-based method suitable for action recognition on embedded systems.
- We develop a real-time multi-person action recognition system.
- We integrate compression techniques to reduce model size, enabling efficient deployment on resource-constrained devices.

II. RELATED WORKS

A. VISION-BASED FALL DETECTION

Fall detection is a critical application in elderly care, aiming to provide timely assistance and prevent serious injuries. Vision-based methods for fall detection leverage video data to analyze human posture and movement patterns. Early approaches [10], [11], [12] employed handcrafted features, such as segmenting the subject from the background, which often struggled with robustness and accuracy in complex environments. Recent advancements have applied different representations of human motion. For example, [13] transforms the video into multiple dynamic images, whereas [14] converts RGB input into optical flow images. However, these methods typically rely on RGB video frames, which can be computationally intensive and prone to errors due to lighting variations and occlusions.

To mitigate these issues, some studies have explored skeleton-based approaches, focusing on the movement of body joints rather than pixel-level details [15], [16], [17], [18], [19], [20]. For instance, [19] proposed a hybrid CNN-LSTM model with 2D keypoints as input, the model that captures temporal dynamics for more accurate fall detection. Reference [17] utilized a ToF camera to extract 3D keypoints and then applied a handcrafted algorithm to calculate the distance between the floor and the joints to detect falls. Reference [20] also focused on the distance between joints, enhancing their approach by using a LSTM network trained on calculated angle and distance sequences to classify falls and non-fall activities. These methods demonstrate improved robustness but still face challenges due to their dependency on the accuracy of the pose estimator. Consequently, several methods have attempted to enhance pose estimator accuracy. For example, [21] fine-tuned the pose estimator on a falling motion pose dataset to improve accuracy, while [15] used an additional CNN to generate sequences of keypoints that the pose estimator failed to extract.

B. SKELETON-BASED ACTION RECOGNITION

Action recognition is fundamentally a time series problem, hence, early research predominantly employed RNN models [22], [23]. In [22], inputs are represented as 3D poses divided into five segments, whereas [23] utilized the joint locations of 3D poses directly as input. Another line of research leveraged CNN models [24], [25], which directly transform the coordinates in a skeleton sequence into a pseudo-image, typically a 2D input of shape $K \times T$, where K represents the number of joints and T is the temporal length.

While both RNN and CNN approaches utilize joint coordinates, they do not explicitly capture the structural topology of the joints, leading to the introduction of Graph Convolutional Networks (GCNs) [26], [27], [28], [29], [30].

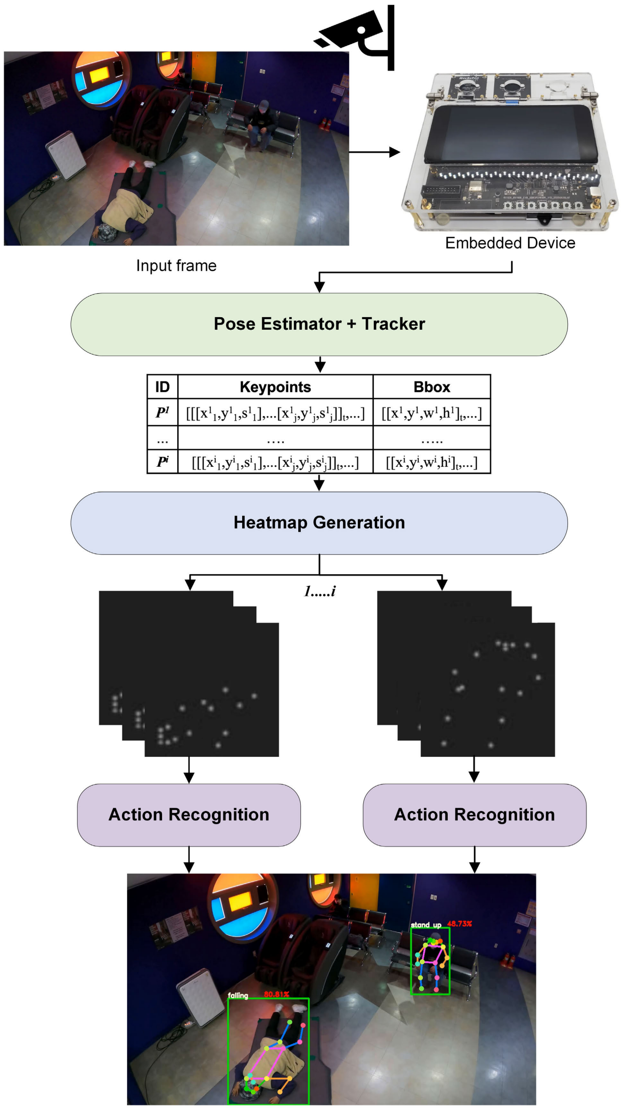
GCN models, such as ST-GCN [30], utilize extracted keypoint sequences as spatiotemporal graphs. AS-GCN [28] employs multi-scale modeling and additionally predicts the human pose. G3D [29] proposes a novel graph convolution operator for capturing long-range joint relationships. Shift-GCN [26] aims to reduce computational cost by proposing a shift graph operation. For instance, HD-GCN [27] proposed a hierarchical graph-based framework that leverages both spatial and temporal dependencies among body joints for improved action recognition performance.

However, for multi-person scenarios, all GCN methods become computationally heavy as they multiply the input for each detected person. To overcome this, a new approach using 3D-CNN, called PoseConv3D [3], was developed to handle multi-person scenarios without additional computation costs. Initially, 3D-CNNs for action recognition were applied to RGB input, as seen in SlowFast [2]. The SlowFast network captures spatial semantics and motion separately by using different spatiotemporal resolutions for the two networks. However, using RGB input requires many channels, making the network computationally intensive. In contrast, PoseConv3D [3] represents the RGB input as stacks of joint heatmaps generated from extracted 2D human poses, forming 3D heatmap volumes as input for the 3D-CNN. This approach efficiently handles multi-person scenarios without imposing excessive computational burdens.

Moreover, 3D-CNNs can learn spatiotemporal features more effectively than 2D-CNNs, making skeleton-based approaches using 3D-CNNs outperform GCN-based methods. This makes them a more effective and efficient choice for action recognition tasks. Efforts have also been made to optimize skeleton-based action recognition models for real-time implementation on embedded devices. Lightweight network architectures and efficient inference techniques are employed to reduce computational complexity and memory footprint while maintaining recognition accuracy. For instance, Noor and Park [21] developed a real-time skeleton-based action recognition system optimized for deployment on GPUs.

C. ACTION RECOGNITION ON EMBEDDED DEVICES

The integration of deep learning models, particularly for action recognition, into IoT and embedded systems presents unique challenges and opportunities. Elhanashi et al. [5] provide a comprehensive overview of this field, emphasizing the need for efficient implementations on resource-constrained devices. Their survey not only highlights the general challenges but also discusses specific techniques and optimizations relevant to deploying deep learning models in IoT contexts. In parallel with these developments, researchers have been exploring various approaches for implementing action recognition on embedded devices. Meng et al. [7] is the first one to proposed an action recognition for embedded device, the method utilized the Hierarchical Motion History Histogram (HMHH) feature to capture motion data and uses a Support Vector Machine (SVM) for classification.



ID	Keypoints	Bbox
P^1	$[[[x^1_1,y^1_1,s^1_1],\dots[x^1_j,y^1_j,s^1_j]]_t,\dots]$	$[[x^1,y^1,w^1,h^1]_t,\dots]$
...	....	.....
P^i	$[[[x^i_1,y^i_1,s^i_1],\dots[x^i_j,y^i_j,s^i_j]]_t,\dots]$	$[[x^i,y^i,w^i,h^i]_t,\dots]$

FIGURE 1. Overall pipeline of the proposed method for real-time action recognition on embedded devices. The pipeline includes connecting a camera to the device, capturing frames, performing simultaneous pose estimation and tracking, generating joint heatmaps and finally classifying actions of each person detected. *i* refers to number of person detected. Details explained in Section IV-B.

Jain et al. [6] designed a federated learning framework that enables numerous devices to collaboratively learn action recognition models without the need to share data. Monisha and Mohan [31] implemented an action recognition approach based on a template matching algorithm. The templates were created using edge detection to identify the contours of human postures and hand gestures. However, this method may not be robust in real-world situations where the background can vary.

III. PROPOSED METHOD

In this section, we present the proposed method, an efficient real-time action recognition system, as illustrated in Figure 1. Our framework consists of three main modules: pose estimation, tracking, and action recognition. The details of the proposed method are discussed in the following sections.

Firstly, we fine-tune a pose estimator model to accurately predict falling motion poses. Next, we implement a tracking system to track each person appearing in the frame, which will give a unique identity for each person. Then, we generate the corresponding pseudo joint heatmap for each person. Finally, we utilize an efficient action recognition module, which employs a factorized 3D-CNN for skeleton-based action recognition.

A. POSE ESTIMATION

We opt for 2D pose estimators due to their significantly lower processing time compared to 3D pose estimators [32], [33], [34]. For instance, [34] reported a processing time of 300.0 ms, while [35] reported a processing time of 233.2 ms. Moreover, according to the findings in [3], the estimated 2D keypoints consistently outperform both sensor-collected (NTU RGB+D [36]) and estimated 3D keypoints in action recognition tasks. Therefore, to balance accuracy and computational efficiency, we use 2D keypoints for our method.

We employ a one-stage pose estimator [35] to optimize time efficiency, aiming to construct an efficient pipeline for real-world applications and embedded device implementation. The pose estimator receives input of RGB frames and outputs bounding boxes $BBox \in R^{x,y,w,h}$ and coordinate triplets (x, y, s) for each joint in every person $\boldsymbol{P}$ detected. Based on COCO keypoints [37] each person has 17 associated keypoints, $Pose \in R^{17\times3}$.

$$\boldsymbol{P} = \{Pose, BBox\}. \tag{1}$$

Additionally, to ensure accuracy, we finetune the pose estimator with a fall person dataset [21].

B. TRACKING

We utilize bounding boxes, RGB frames, and pose data as input for tracking. Unlike the original implementation of DeepSORT [38], which relies solely on bounding boxes and RGB frames, we incorporate additional pose data to enhance the tracking accuracy.

Therefore, the input for the tracker module includes both $\boldsymbol{P}$ and the RGB frame. The tracker will output a unique identity for each person. The output will represent each person $i \in [1, 2, 3, ..n]$ at time $t \in [1, 2, 3, ..T]$, where n and T refer to the number of person and frames, respectively. The tracker output can be expressed as:

$$\boldsymbol{P}_t^i = \{Pose_t^i, BBox_t^i\}. \tag{2}$$

C. PSEUDO JOINT HEATMAP GENERATION

The pseudo joint heatmap serves as the input representation for the action recognition model. It represents spatial information about the human body's keypoints extracted from the estimated poses. We employ the following steps to generate the pseudo joint heatmap:

1) SUBJECT CENTER CROPPING

We perform a subject center cropping to ensure consistent and focused regions. This method calculates the dimensions of the input image and defines a centered square crop and adjusting the joint coordinates to align them within the new dimensions. This preprocessing step standardizes the input frames, enhancing the accuracy and consistency of the subsequent joint heatmap generation.

2) HEATMAP GENERATION

We generate heatmaps for each detected joint using Gaussian maps as implemented in [3]. Given the location of each joint (x_k, y_k) and its confidence score s_k, the heatmap for the k-th joint is defined as:

$$\mathbf{J_{kuv}} = s_k \cdot exp\{-\frac{(u - x_k)^2 + (v - y_k)^2}{2\sigma^2}\}, \tag{3}$$

where (u, v) represents the coordinates in the heatmap grid, and σ controls the variance of Gaussian maps. The confidence score s_k scales the peak of the Gaussian, ensuring that joints with higher confidence have more pronounced peaks in the heatmap. This method allows for precise localization of joints in the generated heatmaps, which is crucial for accurate action recognition. The result of this heatmap generation can be represented as $\boldsymbol{H} \in R^{H\times W\times C}$. Thus, each individual at every frame has their unique heatmap, which can be represented as:

$$\boldsymbol{P_t^i} = \{Pose_t^i, BBox_t^i, H_t^i\}. \tag{4}$$

3) RESIZING DIMENSION

In order to comply with the spatial size prerequisites of the action recognition module, we adjust the spatial dimension of the pseudo joint heatmap to a uniform size of 112×112, which results in $\boldsymbol{H} \in R^{112\times112\times C}$. The rationale for choosing this fixed size will be explain in Section VI. This resizing step normalizes the input data across all frames, thereby ensuring uniformity and promoting efficient computation within the action recognition module.

4) CONVERSION TO GRAYSCALE

To simplify the input data representation and reduce computational complexity and memory requirements, we convert the resized pseudo joint heatmap into grayscale, $\boldsymbol{H} \in R^{112\times112\times1}$.

5) FRAME SAMPLING

Given the temporal nature of action recognition tasks, we exploit temporal redundancy in the input data by uniformly sampling a subset of frames for processing, as validated by experiments in [3]. Specifically, we uniformly sample 32 frames, this frame sampling strategy effectively reduces the computational load while preserving temporal information necessary for accurate action recognition. Therefore, we can represent the output of heatmap generation and

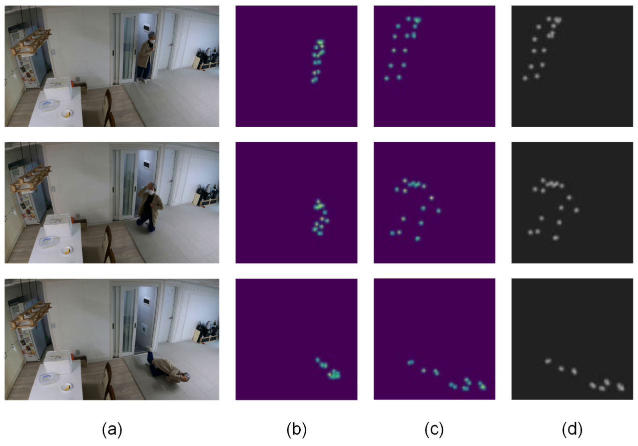


FIGURE 2. Examples of generated heatmaps. (a) The original RGB frame. (b) Generated joint heatmap before subject center cropping. (c) Joint heatmap after applying subject center cropping, ensuring focused regions around the subject. (d) The final grayscale joint heatmap, resized to 112 × 112, used for action recognition.

the input for action recognition as:

$$\mathbf{\Phi}_{\boldsymbol{i}} = \{H_1^i, H_2^i, H_3^i, \ldots, H_{32}^i\}. \tag{5}$$

Examples of the generated joint heatmaps are shown in Figure 2.

TABLE 1. The proposed network architecture. The dimensions of the kernels are denoted by $H \times W$ for height and width, and C for channels. In each layer, the first convolution operation is a 2D convolution applied spatially, followed by a 1D convolution applied temporally on the intermediate channels (M_i). The intermediate channels are calculated as described in Section III-D. The output size format is ($T \times H \times W, C$), where T refers to the number of frames.

Stage	Layer	Output Sizes
Data Layer	32, 112 × 112	32 × 112 × 112, 1
conv$_1$	7 × 7, 8, stride 2	32 × 56 × 56, 8
max pool	3 × 3, stride 2	32 × 28 × 28, 8
Layer$_0$	$\begin{bmatrix} 3 \times 3, 8 \\ 3, M_i \end{bmatrix} \times 2$	32 × 28 × 28, 8
Layer$_1$	$\begin{bmatrix} 3 \times 3, 16 \\ 3, M_i \end{bmatrix} \times 2$	32 × 14 × 14, 16
Layer$_2$	$\begin{bmatrix} 3 \times 3, 32 \\ 3, M_i \end{bmatrix} \times 2$	32 × 7 × 7, 32
Layer$_3$	$\begin{bmatrix} 3 \times 3, 64 \\ 3, M_i \end{bmatrix} \times 2$	32 × 7 × 7, 64
	GAP, FC	# of Classes

D. ACTION RECOGNITION

Our action recognition module leverages a factorized 3D-CNN architecture inspired by [8], balancing computational efficiency and recognition accuracy. As shown in Figure 3, we use the generated heatmap $\mathbf{\Phi}_{\boldsymbol{i}}$ as input, formatted as $T \times H \times W \times C$, where T, H, W, and C denote the number of frames, height, width, and channels, respectively. The backbone of our model is ResNet18 [39]. Unlike the usual 3D-CNN used in [3], our approach decomposes 3D convolutions into separate 2D convolution as spatial convolution and 1D convolution as temporal convolution within each layer. This decomposition is complemented by non-linear rectification and batch normalization steps between the convolutions, which is crucial for enhancing model performance and stability. The approach serves two main purposes: it increases the number of non-linearities in the network, potentially improving the model's learning capacity, and it enables our model to run efficiently on embedded devices like the RV1126 EVB that have limitations in computing 3D convolutions.

In our architecture shown in Table 1, the spatial convolution filters size are $N_{i-1} \times 1 \times d \times d$, where N_{i-1} represents the number of input channels. The output channel of this convolution is M_i, which serves as the input channel for the subsequent temporal convolution. Thus, the temporal convolution filters size is $M_i \times t \times 1 \times 1$. The hyperparameter M_i determines the dimensionality of the intermediate subspace where the signal is projected between the spatial and temporal convolutions. Following the approach in the R(2+1)D model [8], M_i is calculated as,

$$\boldsymbol{M_i} = \frac{td^2 N_{i-1} N_i}{d^2 N_{i-1} + tN_i} \tag{6}$$

to ensure that the number of parameters in the decompose convolution (2D+1D) is approximately equal to that of a full 3D convolution.

The intuition behind this model design is twofold. Firstly, most embedded devices do not support 3D convolution, making it impractical for real-time applications on such platforms. Secondly, according to [8], decomposing the 3D convolution into separate spatial and temporal convolutions effectively doubles the number of nonlinearities in the network. This is due to the additional ReLU activation introduced between the spatial and temporal convolutions, as illustrated in Figure 3. This increased nonlinearity enhances the model's capacity to learn complex features, improving action recognition performance.

For training, we use the cross-entropy loss function to measure the difference between predicted and ground truth labels. Softmax activation is manually applied outside the model to derive final class probabilities, providing greater deployment flexibility and compatibility with various applications. This design ensures that our action recognition module is both accurate and efficient enough for real-time deployment on resource-constrained embedded devices.

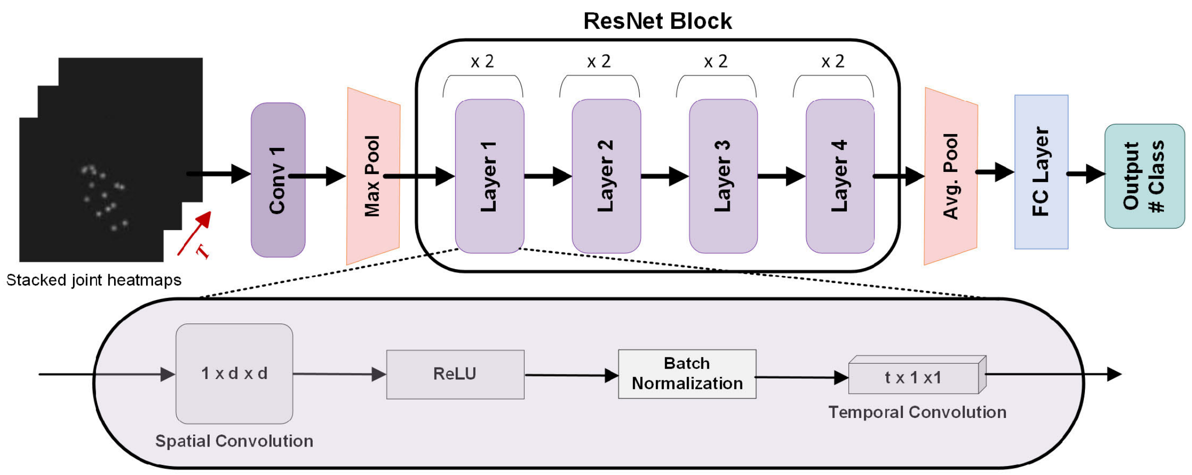


FIGURE 3. Architecture of the proposed action recognition model. Each layer in the ResNet block utilizes a factorized 3D-CNN, decomposing the convolutions into separate spatial (2D convolution) and temporal (1D convolution) convolutions to enhance computational efficiency and recognition accuracy, making it suitable for embedded device implementation. *T* refers the number of frames.

IV. REAL-TIME ACTION RECOGNITION PIPELINE ON EMBEDDED DEVICE

A. OPTIMIZATION

We apply several optimization techniques to ensure our model can run efficiently on embedded devices. Specifically, we employ quantization for pose estimation and tracker modules to reduce the precision of the model weights, thereby decreasing memory usage and improving inference speed. Additionally, we apply pruning and a post-processing algorithm in our pose estimation model to streamline computations. In this section, we provide a detailed explanation of these optimization techniques.

1) QUANTIZATION

We employ "Post-Training Quantization" to optimize the model, which allows direct deployment of the quantized model without additional training. We test 2 different quantization methods that will be explained in this section.

a: UNIFORM AFFINE QUANTIZER

According to [40], this quantization method yields the smallest loss for most networks. We use the following formula to quantize the variable.

$$X_{int} = round(\frac{X_{float}}{\Delta}) + z,$$
$$X_{quant} = clamp(0, N_{prec} - 1, X_{int}) \tag{7}$$

where Δ is the scale and z is the zero-point that maps the floating point to integers. We use the following formula to de-quantize the results:

$$X_{float} = (X_{quant} - z)\Delta \tag{8}$$

We use this method to quantize our model to unsigned integer 8 (uint8).

b: DYNAMIC FIXED POINT

The second quantization method that we use is Dynamic Fixed Point [4]. This method follows the formula below to quantize a given value

$$X_{int} = round(X_{float} * 2^{fl}),$$
$$X_{quant} = clamp(N_{min}, N_{max}, X_{int}) \tag{9}$$

where fl is how much a digit is shifted to the left and

$$\text{clamp}(a, b, x) = \begin{cases} a & \text{if } x \leq a \\ x & \text{if } a < x \leq b \\ b & \text{if } x > b \end{cases}$$
$$N_{max} = 2^{N_{prec}-1} - 1,$$
$$N_{min} = -(2^{N_{prec}-1} - 1) \tag{10}$$

We set $(N_{min}, N_{max}) = (-127, 127)$ for 8-bit precision (int8) and $(N_{min}, N_{max}) = (-32768, 32767)$ for 16-bit precision (int16). We use this method to quantize our model to int8 and int16.

2) POSE ESTIMATOR OPTIMIZATION

a: PRUNING

To enhance the performance of our pose estimator network, we employ a pruning strategy. This involves removing certain operations from the original model that are not associated with any model weights. Specifically, we have pruned the terminal operations, which include transformation and normalization processes. In the ONNX model of YOLOv8s-pose [35], we extract the outputs from node IDs 388, 403, 418, 336, 350, and 364. These nodes provide the raw features for detection and keypoints. By eliminating subsequent nodes, we have effectively increase the inference speed of the model.

b: POST-PROCESSING

Building upon the pruning strategy, we implement custom post-processing to extract the bounding box and keypoint coordinates from the pose estimator's output. This step processes the model's output, which consists of 6 sets of arrays. For detection, we merge the outputs into a single array of bounding box coordinates ($x_{\text{top}}, y_{\text{top}}, x_{\text{bottom}}, y_{\text{bottom}}$). For keypoints, we combine the outputs into a single array of 51 keypoint proposals, following the COCO-keypoints format [37], which includes 17 keypoints, each represented by 3 values (coordinates (x, y) and score (z)). Finally, we filter out low-confidence detections and apply non-maximum suppression (NMS) to remove redundant bounding boxes, ensuring the accuracy and efficiency of the final detection and keypoint outputs.

B. DEPLOYMENT ON EMBEDDED DEVICE

To fully utilize the capabilities of the Rockchip NPU module, we need to convert the model to Rockchip's file format, called RKNN. First, we convert our PyTorch model to either a TorchScript or ONNX format. In our method, we choose to convert our model to ONNX with an opset of 11 due to the capabilities and ease of use of the ONNX format. After converting our models to ONNX, we can then convert them to RKNN format using Rockchip's built-in conversion tool [4]. The real-time inference pipeline, illustrated in Figure 1, integrates our method into embedded systems for efficient action recognition. We leverage the RKNN models of the pose estimator, tracker, and action recognition module for implementation on the embedded device.

The process starts by connecting webcam or CCTV camera to the embedded device. Once frames are captured, the device simultaneously performs pose estimation and tracks individuals within the scene. Every 64 frames, the system generates joint heatmaps, as described in Section III-C. To optimize the temporal information, we uniformly sample 32 frames, ensuring a comprehensive understanding of the motion dynamics. These joint heatmaps are then fed into our action recognition module, which classifies the actions of individuals within the frame. This streamlined process enables efficient, real-time action recognition on embedded devices, demonstrating the practical applicability of our method in various surveillance and monitoring scenarios.

V. EXPERIMENTAL RESULTS

In this section, we present the experimental results of our proposed method. First, we validate the ability of our model to effectively distinguish falling motions from other daily activities. Second, we conduct a comprehensive comparison between our action recognition model and pipeline against baseline methods. Third, we evaluate the performance of our model when implemented on an embedded device.

CPU	Quad-core Arm Cortex-A7 @ 1.5 GHz + RISC-V MCU @ 400 MHz
NPU	2.0 TOPS
RAM	1GB DDR3

FIGURE 4. Rockchip RV1126 EVB specifications.

A. IMPLEMENTATION DETAILS

We train our action recognition model using stochastic gradient descent (SGD) with a momentum of 0.9 and weight decay set to 0.001. The learning rate is set to decay with cosine annealing (SGDR) [41], starting from an initial rate of 0.01 and decreasing to a minimum of 0.0001. To stabilize training, we include a warm-up phase for the first 10 epochs [42], with the entire training process spanning 240 epochs. Both training and testing were conducted on an Intel i7-11700K CPU with 64GB of RAM, and NVIDIA RTX 4090 GPU.

Following model evaluation on the GPU, we deployed the trained model onto EVB Rockchip RV1126 embedded device. This device is optimized for edge computing applications and operates under limited computational and memory resources. The specifications of the embedded device are detailed in Figure 4. As discussed in Section IV, deploying the model on the embedded device involved optimizing for inference, taking into account constraints such as computational efficiency and memory footprint. This process demonstrates the feasibility and practicality of our approach in real-world scenarios with resource-constrained environments.

B. EVALUATION ON FALL DETECTION

We evaluate our proposed model, trained as a binary classifier for fall detection, on two benchmark datasets and compare its performance with previous fall detection methods. The evaluation metrics for fall detection [43] are accuracy, sensitivity, and specificity. *Accuracy* is the proportion of correctly detected falls and non-fall behaviors. *Sensitivity* is the proportion of correctly detected falls among all fall events. *Specificity* is the proportion of correctly detected non-fall behaviors among all non-fall events.

1) DATASETS

a: UR FALL DETECTION (URFD) DATASET [44]

contains 70 videos: 30 videos of falling motion from 2 different camera angles and 40 videos of activities of daily living. For training and testing, we use a random dataset split with an 80:20 ratio for each class.

b: AI HUB SENIOR ABNORMAL BEHAVIOR DATASET [45]

contains a total of 9,400 videos of 3 human actions. The actor is assumed to be elderly and the action classes are threefold:

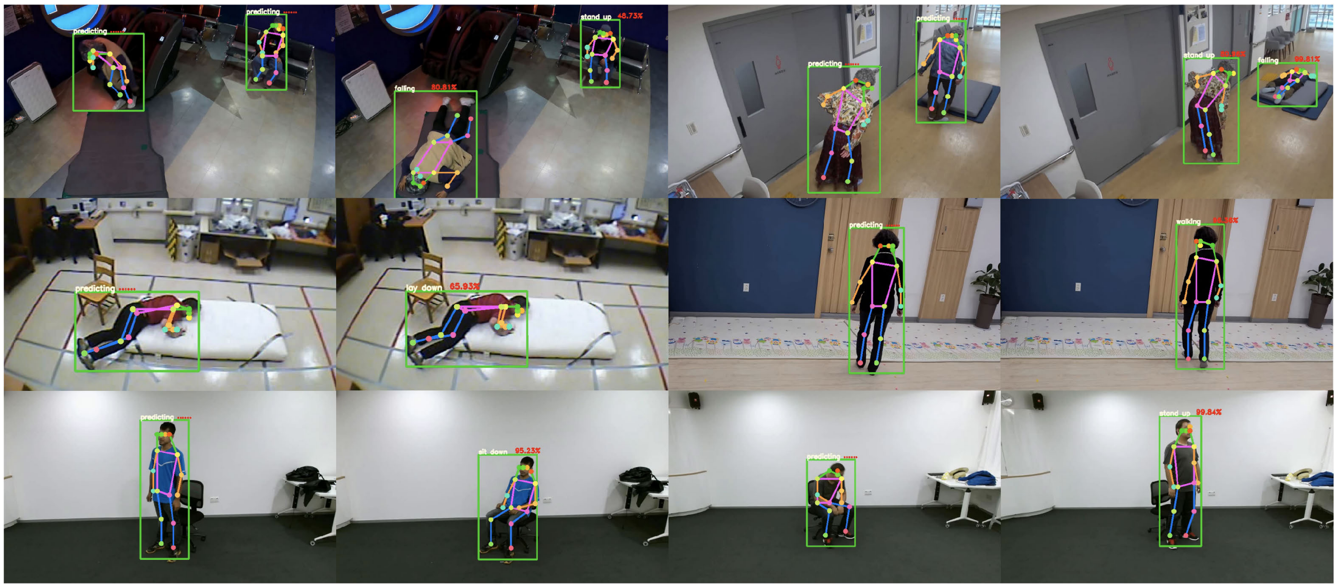


FIGURE 5. Example frames of the proposed multi-person fall detection and action recognition system.

TABLE 2. Comparison with the state-of-the-art on the URFD dataset [44]. The † symbol denotes models trained with the pose estimator fine-tuned using the fall person dataset [21].

Model	Accuracy	Specificity	Sensitivity
Lin *et al.* [18]	92.00	-	-
Hasan *et al.* [16]	-	96.00	99.00
Marcos *et al.* [14]	95.00	92.00	100.00
Salimi *et al.* [19]	98.90	-	100.00
Noor *et al.*† [21]	**100.00**	**100.00**	**100.00**
Ours	90.00	87.50	91.67
Ours†	**100.00**	**100.00**	**100.00**

TABLE 3. Comparison with the state-of-the-art on the AI Hub dataset [45]. The † symbol denotes models trained with the pose estimator fine-tuned using the fall person dataset [21].

Model	Accuracy	Specificity	Sensitivity
Duan *et al.* [3]	97.56	97.72	97.26
Noor *et al.*† [21]	**99.02**	**100.00**	97.26
Ours	97.29	96.22	97.72
Ours†	98.37	96.26	**99.24**

fall down, wander, and dementia. We discard the dementia class, and only use the falling and wander data.

2) COMPARISON WITH THE STATE-OF-THE-ART

Our fall detection model demonstrates strong performance on both benchmark datasets, URFD [44] and AI Hub [45], as shown in Table 2 and Table 3. On the URFD dataset [44], our model achieves perfect accuracy, specificity, and sensitivity, outperforming all previous methods. Notably, models fine-tuned with the fall person dataset [21], denoted by †, achieve exceptional performance, achieving 100.00% on all metrics. Similarly, on the AI Hub dataset [45], our model demonstrates competitive performance with prior works. Our model achieves an accuracy of 98.37%, specificity of 96.26%, and sensitivity of 99.24%. These results indicate the robustness and effectiveness of our approach in distinguishing between falling and non-falling activities. Notably, our model's performance is competitive with state-of-the-art methods, demonstrating its potential for real-world fall detection applications.

C. EVALUATION ON ACTION RECOGNITION

1) DATASET

We train and test our method by using the 5 action combined dataset (NTU RGB+D [36], NW-UCLA [46], URFD [44], Multiple Cameras Fall Dataset [47], and AI Hub [45]) proposed in [21]. 5 action combined dataset is a compilation of various datasets merged together, it is specifically curated to represent the most prevalent indoor activities encountered in real-world scenarios. The 5 classes are: Falling, Sit Down, Stand up, Walking, and Laydown. In total, this dataset provides 6,231 videos and also provides annotated sequences of human keypoints for each video. For training and testing, we use a random dataset split with an 80:20 ratio for each class.

2) PERFORMANCE OF ACTION RECOGNITION ON GPU

Before deploying our proposed model on the embedded device, we evaluate its performance on GPU. We assess the computational efficiency of our model by calculating

TABLE 4. Comparison with the state-of-the-art on 5 action combined dataset [21].

Model	Accuracy (%)	Parameters (M)	Model Disk Size (MB)	Proc. Time (ms)
Duan *et al.* [3]	**98.26**	2.0	16.2	392.50
Noor *et al.* [21]	97.93	0.5	4.3	40.00
Ours	95.62	**0.2**	**0.9**	**15.44**

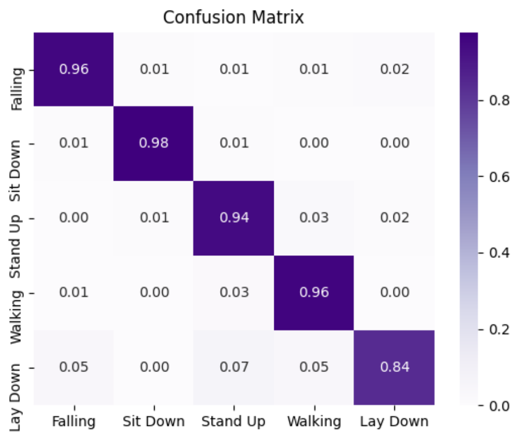


FIGURE 6. Confusion matrix of the action recognition model evaluated on the 5 action combined dataset [21]. Rows corresponding to actual classes and columns corresponding to predicted classes.

the number of parameters (params), its disk size, and the processing speed. Additionally, we evaluate the accuracy of our proposed model across different classes by analyzing the confusion matrix depicted in Figure 6, where each row represents instances of the ground truth label and each column represents instances predicted in a class. The diagonal elements indicate the number of correctly classified instances for each class, while off-diagonal elements represent misclassifications.

In Table 4, we compare our results with prior methods. Our model achieves a competitive accuracy of 95.62%, with a significantly smaller size and faster processing time. Specifically, our model is 20 times smaller and 25 times faster than the baseline model by Duan et al. [3], which has an accuracy of 98.26%. Moreover, compared to Noor and Park [21], our proposed model is 24.56 ms faster. This demonstrates the efficiency and effectiveness of our approach for real-time action recognition tasks.

3) PERFORMANCE RESULT ON EMBEDDED DEVICE

a: PERFORMANCE OF THE QUANTIZED MODEL ON AN EMBEDDED DEVICE

We evaluate different quantized models for action recognition on an embedded device by inferring 50 videos, with 10 videos per class in the test set. Table 5 shows the accuracy and processing time for each combination of quantized pose estimator and tracker, while the action recognition model remained unquantized (float32). Due to excessive processing time (2.15 seconds per frame) of the unquantized pose estimator, we excluded it from our evaluation. However, we tested the unquantized tracker with various quantized pose estimators. The results demonstrate that the performance of our models is significantly influenced by the quantization level of the pose estimator and tracker. Models quantized to int16 show lower accuracy and slower processing speeds, while int8 or uint8 quantization improves both accuracy and speed. The optimal configuration, achieving 90.00% accuracy with the fastest processing times, includes a uint8-quantized pose estimator and an int8-quantized tracker. This setup is ideal for real-time action recognition on embedded devices. Qualitative results are shown in Figure 5, demonstrating effective detection of falling motion and accurate classification of other action classes.

b: SPEED PERFORMANCE BETWEEN DEVICES

In Table 6 we compare action recognition accuracy and processing times (in ms) across different devices. On GPU our model achieves 95.62% accuracy with faster processing times: 18.939 ms for pose estimation, 4.37 ms for tracking, and 15.44 ms for action recognition. On CPU resulting in longer processing times: 250.05 ms for pose estimation, 19.30 ms for tracking, and 24.07 ms for action recognition. On the embedded device, accuracy drops to 90.00%, with slower processing times: 94.14 ms for pose estimation, 3.56 ms for tracking, and 179.47 ms for action recognition. We achieve frame rate of 7.9 Hz on the embbeded device and 16.3 Hz on GPU.

c: MEMORY USAGE EVALUATION

The details of the evaluation metrics of the memory usage of the RKNN model during inference on Rockchip NPU are as follows.

- System memory: System memory allocated by non-NPU drivers, including memory allocated in the system for models and input data
- NPU memory: Indicates the memory allocated by the NPU driver during model inference

The findings presented in Table 7 show that the pose estimator is the most memory-intensive module, using the highest system and NPU memory at both maximum and total allocations. In contrast, the tracker and action recognition modules have significantly lower memory usage.

VI. ABLATION STUDY

We evaluate the effect of different channel sizes in the proposed network. In Table 8, we present the results of our ablation study, focusing on the effects of varying channel sizes in the initial convolutional layer and the subsequent ResNet layers. While larger channel sizes appear to yield slightly higher accuracy in the trained model, the accompanying increase in total parameters presents a notable trade-off. Our analysis reveals that the accuracy

TABLE 5. Comparison of proposed modules on an embedded device (Rockchip's RV1126). The table shows action recognition accuracy along with the processing time (in milliseconds) for each module and model type. The best results are presented in **bold**. The row with bold cells indicates the selected model for our real-time action recognition system on the embedded device.

Pose Estimator	Tracker	Accuracy (%)	Pose Proc. Time	Track Proc. Time	Action Proc. Time
int16	float32	84.00	277.18	377.09	273.29
	int16	84.00	238.63	7.37	227.76
	int8	90.00	232.92	4.45	227.18
	uint8	88.00	232.97	4.44	215.69
int8	float32	44.00	128.19	411.21	376.87
	int16	46.00	106.79	8.41	281.71
	int8	52.00	98.36	3.84	273.48
	uint8	44.00	98.16	3.71	262.34
uint8	float32	84.00	121.99	424.43	244.89
	int16	88.00	103.12	8.51	202.89
	int8	**90.00**	**94.14**	**3.56**	**179.47**
	uint8	86.00	94.65	3.95	179.72

TABLE 6. Action recognition accuracy and processing time of each module on different computing devices.

Device	Accuracy (%)	Pose Proc. Time	Track Proc. Time	Action Proc. Time
GPU	95.62	18.939	4.37	15.44
CPU	95.62	250.05	19.30	24.07
Embedded Device	90.00	94.14	3.56	179.47

TABLE 7. Memory usage (MB) of each module on the embedded device. *Max* refers to the maximum memory allocated at any point during execution, and *Total* refers to the total memory allocated over the entire runtime.

Module	System Memory		NPU Memory	
	Max	*Total*	*Max*	*Total*
Pose Estimator	35.26	342.11	35.56	35.57
Tracker	25.21	198.69	23.05	23.58
Action Recognition	45.59	61.61	19.48	19.57

TABLE 8. Comparison of performance across different channel sizes in the proposed network.

Channel Sizes	Parameters (M)	Accuracy (%)	Proc. Time (ms)
$\begin{bmatrix} Conv_1, 16 \\ Res_2, 16 \\ Res_3, 32 \\ Res_4, 64 \\ Res_5, 128 \end{bmatrix}$	0.9	**95.70**	16.84
$\begin{bmatrix} Conv_1, 8 \\ Res_2, 8 \\ Res_3, 16 \\ Res_4, 32 \\ Res_5, 64 \end{bmatrix}$	**0.2**	95.62	**15.44**

decrease associated with smaller channel sizes is minimal, suggesting that the benefits of larger channels may not justify the additional computational cost. Consequently, we opt for the smallest model configuration, utilizing a first channel output of 8, as our baseline model. This decision balances model complexity and accuracy, ensuring optimal performance while minimizing computational resources.

VII. CONCLUSION

In this paper, we introduced an innovative approach to tackle the challenges associated with real-time implementation of action recognition especially on embedded devices. Our method leveraged skeleton-based action recognition and employs Factorized 3D-CNN network architecture, enabling real-time operation on resource-constrained devices. By focusing on computational efficiency and maintaining high accuracy, our proposed methodology was well-suited for practical applications, particularly in surveillance systems. The experimental results substantiated the effectiveness of our approach. We achieved high recognition accuracy while ensuring real-time performance, demonstrating the practical viability of our system. Overall, our work contributed to advancing action recognition capabilities on embedded devices, paving the way for enhanced applications in various real-world scenarios. While our current implementation utilizes hardware with NPU support, we believe our model's lightweight architecture makes it adaptable to a range of embedded devices, including those without NPU support. However, this may involve trade-offs in latency, power consumption, and frame rate. Future work could explore additional optimization techniques such as pruning or knowledge distillation to further reduce computational requirements. Additionally, investigating the integration of other sensor data and adapting our approach to different application domains could broaden the impact of this research.

REFERENCES

[1] J. Carreira and A. Zisserman, "Quo vadis, action recognition? A new model and the kinetics dataset," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 4724–4733.

[2] C. Feichtenhofer, H. Fan, J. Malik, and K. He, "SlowFast networks for video recognition," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, Oct. 2019, pp. 6201–6210.

[3] H. Duan, Y. Zhao, K. Chen, D. Lin, and B. Dai, "Revisiting skeleton-based action recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2022, pp. 2959–2968.

[4] R. Hong. (2023). *RKNN Documentations*. [Online]. Available: https://github.com/rockchip-linux/rknn-toolkit

[5] A. Elhanashi, P. Dini, S. Saponara, and Q. Zheng, "Integration of deep learning into the IoT: A survey of techniques and challenges for real-world applications," *Electronics*, vol. 12, no. 24, p. 4925, Dec. 2023.

[6] P. Jain, S. Goenka, S. Bagchi, B. Banerjee, and S. Chaterji, "Federated action recognition on heterogeneous embedded devices," 2021, *arXiv:2107.12147*.

[7] H. Meng, N. Pears, and C. Bailey, "A human action recognition system for embedded computer vision application," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2007, pp. 1–6.

[8] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri, "A closer look at spatiotemporal convolutions for action recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 6450–6459.

[9] N. Noor, F. Jametoni, J. Kim, H. Hong, and I. K. Park, "Efficient skeleton-based action recognition for real-time embedded systems," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshop*, Jun. 2024, pp. 5889–5897.

[10] Y. Chen, W. Li, L. Wang, J. Hu, and M. Ye, "Vision-based fall event detection in complex background using attention guided bi-directional LSTM," *IEEE Access*, vol. 8, pp. 161337–161348, 2020.

[11] P. Feng, M. Yu, S. M. Naqvi, and J. A. Chambers, "Deep learning for posture analysis in fall detection," in *Proc. 19th Int. Conf. Digit. Signal Process.*, Aug. 2014, pp. 12–17.

[12] Q. Feng, C. Gao, L. Wang, Y. Zhao, T. Song, and Q. Li, "Spatio-temporal fall event detection in complex scenes using attention guided LSTM," *Pattern Recognit. Lett.*, vol. 130, pp. 242–249, Feb. 2020.

[13] Y. Fan, M. D. Levine, G. Wen, and S. Qiu, "A deep neural network for real-time detection of falling humans in naturally occurring scenes," *Neurocomputing*, vol. 260, pp. 43–58, Oct. 2017.

[14] A. Nuñez-Marcos, G. Azkune, and I. Arganda-Carreras, "Vision-based fall detection with convolutional neural networks," *Wireless Commun. Mobile Comput.*, vol. 2017, no. 1, pp. 1–16, 2017.

[15] A. Apicella and L. Snidaro, "Deep neural networks for real-time remote fall detection," in *Proc. ICPR Int. Workshops Challenges*, vol. 12662, 2021, pp. 188–201.

[16] M. M. Hasan, M. S. Islam, and S. Abdullah, "Robust pose-based human fall detection using recurrent neural network," in *Proc. IEEE Int. Conf. Robot., Autom., Artif.-Intell. Internet-of-Things*, Nov. 2019, pp. 48–51.

[17] X. Kong, T. Kumaki, L. Meng, and H. Tomiyama, "A skeleton analysis based fall detection method using ToF camera," *Proc. Comput. Sci.*, vol. 187, pp. 252–257, Aug. 2021.

[18] C.-B. Lin, Z. Dong, W.-K. Kuan, and Y.-F. Huang, "A framework for fall detection based on OpenPose skeleton and LSTM/GRU models," *Appl. Sci.*, vol. 11, no. 1, p. 329, Dec. 2020.

[19] M. Salimi, J. J. M. Machado, and J. M. R. S. Tavares, "Using deep neural networks for human fall detection based on pose estimation," *Sensors*, vol. 22, no. 12, p. 4544, Jun. 2022.

[20] Y. Zheng, D. Zhang, L. Yang, and Z. Zhou, "Fall detection and recognition based on GCN and 2D pose," in *Proc. 6th Int. Conf. Syst. Informat. (ICSAI)*, Nov. 2019, pp. 558–562.

[21] N. Noor and I. Kyu Park, "A lightweight skeleton-based 3D-CNN for real-time fall detection and action recognition," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshops*, Oct. 2023, pp. 2171–2180.

[22] Y. Du, W. Wang, and L. Wang, "Hierarchical recurrent neural network for skeleton based action recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 1110–1118.

[23] J. Liu, G. Wang, P. Hu, L.-Y. Duan, and A. C. Kot, "Global context-aware attention LSTM networks for 3D action recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 3671–3680.

[24] G. Chéron, I. Laptev, and C. Schmid, "P-CNN: Pose-based CNN features for action recognition," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2015, pp. 3218–3226.

[25] D. C. Luvizon, D. Picard, and H. Tabia, "2D/3D pose estimation and action recognition using multitask deep learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 5137–5146.

[26] K. Cheng, Y. Zhang, X. He, W. Chen, J. Cheng, and H. Lu, "Skeleton-based action recognition with shift graph convolutional network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2020, pp. 180–189.

[27] J. Lee, M. Lee, D. Lee, and S. Lee, "Hierarchically decomposed graph convolutional networks for skeleton-based action recognition," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, Oct. 2023, pp. 10410–10419.

[28] M. Li, S. Chen, X. Chen, Y. Zhang, Y. Wang, and Q. Tian, "Actional-structural graph convolutional networks for skeleton-based action recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2019, pp. 3590–3598.

[29] Z. Liu, H. Zhang, Z. Chen, Z. Wang, and W. Ouyang, "Disentangling and unifying graph convolutions for skeleton-based action recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2020, pp. 140–149.

[30] S. Yan, Y. Xiong, and D. Lin, "Spatial temporal graph convolutional networks for skeleton-based action recognition," in *Proc. AAAI Conf. Artif. Intell.*, 2018, vol. 32, no. 1, pp. 7444–7452.

[31] M. Monisha and P. S. Mohan, "A real-time embedded system for human action recognition using template matching," in *Proc. IEEE Int. Conf. Electr., Instrum. Commun. Eng.*, Apr. 2017, pp. 1–5.

[32] M. Kocabas, N. Athanasiou, and M. J. Black, "VIBE: Video inference for human body pose and shape estimation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2020, pp. 5252–5262.

[33] D. Mehta, S. Sridhar, O. Sotnychenko, H. Rhodin, M. Shafiei, H.-P. Seidel, W. Xu, D. Casas, and C. Theobalt, "VNect: Real-time 3D human pose estimation with a single RGB camera," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 1–14, Aug. 2017.

[34] H. Tu, C. Wang, and W. Zeng, "VoxelPose: Towards multi-camera 3D human pose estimation in wild environment," in *Proc. Eur. Conf. Comput. Vis.*, Aug. 2020, pp. 197–212.

[35] (2024). *Pose—Ultralytics YOLOv8 Documentations*. [Online]. Available: https://docs.ultralytics.com/tasks/pose

[36] A. Shahroudy, J. Liu, T.-T. Ng, and G. Wang, "NTU RGB+D: A large scale dataset for 3D human activity analysis," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 1010–1019.

[37] T. Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 740–755.

[38] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in *Proc. IEEE Int. Conf. Image Process.*, Sep. 2017, pp. 3645–3649.

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.

[40] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018, *arXiv:1806.08342*.

[41] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–24.

[42] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch SGD: Training ImageNet in 1 hour," 2017, *arXiv:1706.02677*.

[43] E. Alam, A. Sufian, P. Dutta, and M. Leo, "Vision-based human fall detection systems using deep learning: A review," 2022, *arXiv:2207.10952*.

[44] B. Kwolek and M. Kepski, "Human fall detection on embedded platform using depth maps and wireless accelerometer," *Comput. Methods Programs Biomed.*, vol. 117, no. 3, pp. 489–501, Dec. 2014.

[45] (2020). *AI Hub Senior Abnormal Behavior Video Dataset*. [Online]. Available: https://aihub.or.kr/aihubdata/data/view.do?currMenu=115&topMenu=100&aihubDataSe=realm&dataSetSn=167

[46] J. Wang, X. Nie, Y. Xia, Y. Wu, and S.-C. Zhu, "Cross-view action modeling, learning, and recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 2649–2656.

[47] E. Auvinet, C. Rougier, J. Meunier, A. St-Arnaud, and J. Rousseau, "Multiple cameras fall data set," Universite de Montreal, Montreal, QC, Canada, Tech. Tech. Rep. 1350, 2010.

NADHIRA NOOR (Student Member, IEEE) received the B.Comp.Sc. degree in computer science from Bina Nusantara University, Indonesia, in 2020. She is currently pursuing the Ph.D. degree with Inha University. Her research interests include computer vision and deep learning, with a focus on human action recognition.

IN KYU PARK (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer science from Seoul National University, in 1995, 1997, and 2001, respectively. From September 2001 to March 2004, he was a Member of the Technical Staff with the Samsung Advanced Institute of Technology. Since March 2004, he has been with the School of Information and Communication Engineering, Inha University, where he is currently a Full Professor. From January 2007 to February 2008, he was an Exchange Scholar with Mitsubishi Electric Research Laboratories. From September 2014 to August 2015, he was a Visiting Associate Professor with MIT Media Lab. From July 2018 to June 2019, he was a Visiting Scholar with the Center for Visual Computing, University of California at San Diego. His research interests include the joint area of computer vision and graphics, including 3D shape reconstruction from multiple views, image-based rendering, computational photography, deep learning, and GPGPU for image processing and computer vision. He is a member of ACM.