Efficient Skeleton-Based Action Recognition for Real-Time Embedded Systems

Nadhira Noor†, Fabianaugie Jametoni†, Jinbeom Kim‡, Hyunsu Hong‡, and In Kyu Park†

†Dept. of Electrical and Computer Engineering, Inha University, Incheon 22212, Korea
‡Finedigital Inc., Gyeonggi-do 13496, Korea
{nadhirannoor, akmalogi}@gmail.com, {jbkim, phil}@difine.co.kr, pik@inha.ac.kr

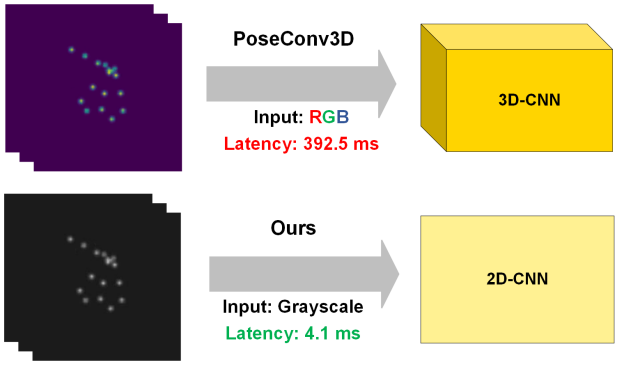
Abstract

Action recognition is vital for various real-world applications, yet its implementation on embedded systems or edge devices faces challenges due to limited computing and memory resources. Our goal is to facilitate lightweight action recognition on embedded systems by utilizing skeleton-based techniques, which naturally require less computing and memory resources. To achieve this, we propose innovative methodologies and optimizations tailored for embedded deployment, including post-training quantization, optimized model architectures, and efficient resource utilization. By enabling real-time and lightweight action recognition on resource-constrained embedded systems, our research opens up new possibilities for applications in areas like autonomous surveillance, driving, and indoor safety monitoring.

1. Introduction

Human action recognition, a fundamental task in video processing, aims to classify human action in video sequences. Over the years, action recognition has received substantial attention within the computer vision community owing to its possible wide-ranging practical applications, including surveillance, robotics, and autonomous driving. Despite its importance, implementing action recognition systems in real-world scenarios presents significant challenges, primarily due to their computational complexity.

Existing action recognition methods based on deep learning and using RGB as input [3, 7] often exhibit heavy computational demands, these methods rely on processing large amounts of pixel data from RGB video frames. Hence, 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. Recognizing these limitations, researchers are increasingly turning to skeleton-based approaches, which prioritize capturing



	Acc	Params	GPU/CPU	Embedded Device
PoseConv3D [6]	98.26%	2.0M	✓	✗
Ours	**98.35%**	**0.1M**	✓	✓

Figure 1. The illustration of our method compared to PoseConv3D [6] on 5 action combined dataset [20]. Our method is 96% faster while demonstrating competitive performance when compared to the state-of-the-art approach.

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. Previous works in this domain, such as PoseConv3D [6], 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 [9]. Conversely, several approaches [10, 18] 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, with highly accurate models and less than 1 million of total parameters. In our methodology, we represent human motion by generating joint heatmaps from detected 2D coordinates of human keypoints across the time frame to represent human motion. We propose a method that utilizes a 2D-CNN network, offering a lighter alternative that remains viable for deployment on embedded devices while preserving accuracy. By adopting this approach, we aim to design a more efficient action recognition system that can operate in real-time on devices with limited resources, addressing the challenges of deploying in embedded environments. Additionally, we incorporate compression techniques such as quantization to reduce the model's footprint further. As illustrated in Figure 1 our method achieves higher accuracy compare to the baseline [6] while being smaller in size and faster processing speed. The contributions of this paper are as follows,

- We propose a light-weight deep learning model utilizing a skeleton-based method that are suitable for action recognition on embedded system
- We develop a real-time action recognition system on embedded device

2. Related works

Skeleton-Based Action Recognition Extensive research has been conducted on recognizing human action from video for many years. Skeleton-based action recognition has arisen as a promising approach due to its effective capture of human motion dynamics. Unlike RGB-based methods, skeleton-based approaches are resilient to various noise sources, such as changes in background color and lighting conditions.

One prominent approach in skeleton-based action recognition involves representing human poses as graphs, where joints correspond to nodes and limb connections represent edges. Graph convolutional networks (GCNs) [4, 13, 14, 24], have been successfully applied to extract spatial and temporal features from skeletal data for action classification. For instance, HD-GCN [13] proposed a hierarchical graph-based framework that leverages both spatial and temporal dependencies among body joints for improved action recognition performance.

Convolutional Neural Networks (CNNs) [6] and Recurrent Neural Networks (RNNs) [5, 16], have shown promising results in capturing complex motion patterns and recognizing actions accurately when applied to skeletal data. For example, Duan *et al.* [6] introduced a novel spatial-temporal convolutional network for skeleton-based action recognition that effectively integrates spatial and temporal information through 3D-CNN.

Efforts for real-time implementation have 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. Noor *et al.* [20] developed a real-time skeleton-based action recognition system optimized for deployment on GPU.

Action Recognition on Embedded Devices Meng *et al*. [18] introduced a human action recognition system for embedded systems. [18] utilized Hierarchical Motion History Histogram (HMHH) feature to represent the motion information and employed a Support Vector Machine (SVM) as the classifier. Monisha *et al*. [19] utilized template matching algorithm for action recognition, where the template images were generated by using edge detection to find the boundary of the human posture and the hand gesture, making it not suitable in real-world scenario where the background scene is variable. Jain *et al*. [19] developed a federated learning framework that allows a large number of devices to jointly learn action recognition models without sharing data.

3. Methodology

3.1. Pose Estimator

The pose estimator receives input of RGB frames and outputs bounding boxes and coordinates of each keypoint (x, y, score). We have a total of 17 coordinates for each person based on COCO-keypoints [15]. To achieve efficient pose estimation suitable for deployment on embedded device, we leverage YOLOv8s-pose [22], a state-of-the-art architecture renowned for its balance between accuracy and speed in human pose estimation tasks. Additionally, to further optimize this architecture for embedded deployment, we introduce two key modifications: pruning and custom post-processing, which will be explained in Section 3.4.1.

3.2. Heatmap Generation

The pseudo joint heatmap serves as the input representation for the action recognition model. It encapsulates 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. Heatmap Generation: As implemented in PoseConv3D [6], given the location of each joint, we generate the heatmap at every joint detected using gaussian maps.

2. Center Cropping: We perform center cropping on the input frames to focus on the subject of interest. This cropping step ensures the subject remains centered within the frame, facilitating more accurate action recognition.

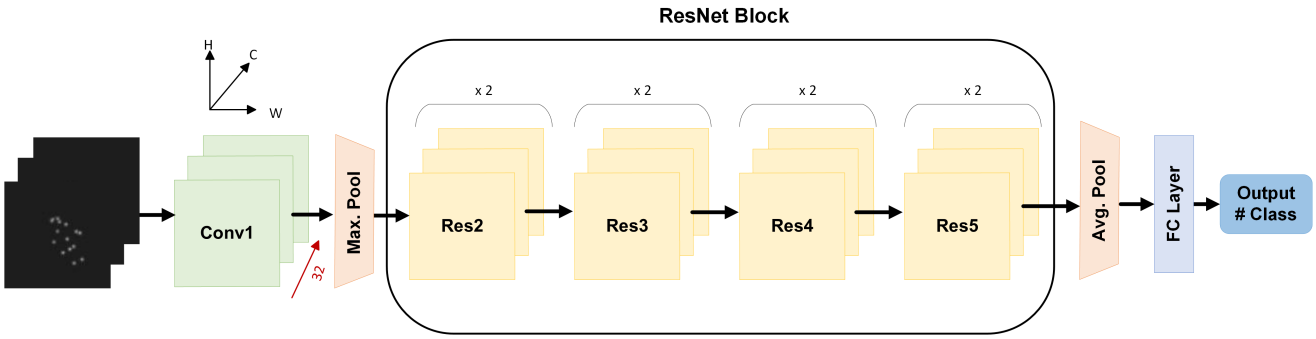


Figure 2. Network architecture of our model. Given an input of stacked generated joint heatmap, we employ a number of #T channels in line with the number of frames for the convolution.

3. Resizing: To conform to the spatial size requirements of the action recognition module, we resize the pseudo joint heatmap to a fixed spatial size of 112×112 . This resizing step ensures that the input data are consistent across all frames and facilitates efficient processing within the action recognition module.

4. Conversion to Grayscale: As a further preprocessing step, we convert the resized pseudo joint heatmap into grayscale. This conversion simplifies the input data representation, reducing computational complexity and memory requirements for subsequent processing in the action recognition module.

In addition to generating pseudo joint heatmaps, our system optimizes frame sampling. To reduce computational load and temporal redundancy while preserving temporal information, we uniformly sample 32 frames out of every 64 frames.

3.3. Action Recogniton

In the action recognition module, we present a novel approach designed to efficiently utilize spatial and temporal information while minimizing computational complexity. Our method is based on a modified ResNet18 architecture that serves as the backbone for our action recognition model. Figure 2 shows our overall architecture.

To begin, the action recognition module receives the generated heatmap as grayscale, a single-channel image as input. This preprocessing step simplifies the input representation while retaining essential information. Next, we stack 32 frames of the sampled generated pseudo heatmap to form a multi-channel input tensor. By stacking the heatmaps over time, we preserve both spatial and temporal features within the input data.

In Table 1 we show our proposed network, we reduce the number of channels from 32 (number of input frames) to 8, reducing the number of channels is done intentionally to decrease the computational complexity of the network and to extract more abstract features from the input data. Next, we employ ResNet blocks to extract hierarchical features from the input tensor. These blocks consist of convolutional layers followed by batch normalization and ReLU activation functions, facilitating feature extraction while mitigating the risk of vanishing gradients. Subsequently, we employ a linear layer to map the extracted features and obtain the logits as output. During training, we utilize the cross-entropy loss function to quantify the disparity between predicted and ground truth labels. Additionally, we manually apply softmax activation outside the model to obtain the final class probabilities. This approach allows for greater flexibility in model deployment and facilitates compatibility for downstream applications. Our action recognition module offers a balance between computational efficiency and recognition accuracy, making it suitable for real-time deployment on resource-constrained embedded device.

Stage	AR	Output Sizes ($C \times H \times W$)
Data Layer	uniform 32, 112×112	$32 \times 112 \times 112$
conv_1	7×7, 8, stride 2	$8 \times 56 \times 56$
max pool	7×7, stride 2	$8 \times 28 \times 28$
Res_2	$\begin{bmatrix} 3 \times 3, 8 \\ 3 \times 3, 8 \end{bmatrix} \times 2$	$8 \times 28 \times 28$
Res_3	$\begin{bmatrix} 3 \times 3, 16 \\ 3 \times 3, 16 \end{bmatrix} \times 2$	$16 \times 14 \times 14$
Res_4	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix} \times 2$	$32 \times 7 \times 7$
Res_5	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$64 \times 7 \times 7$
	GAP, FC	# of Classes

Table 1. The proposed network. The dimensions of the kernels are denoted by $H \times W, C$ for height, width, and channel size. GAP and FC denote the global average pooling and fully connected, respectively.

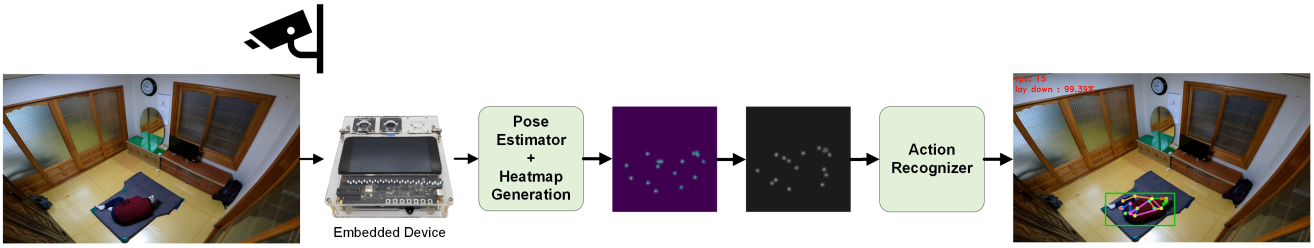


Figure 3. Inference pipeline on embedded device.

3.4. Optimization

We employ several optimization techniques to further optimize our model for speed. In our pose estimator model, we employ both model pruning and quantization. In our action recognition model, we employ only quantization.

3.4.1 Pruning and Post-processing

Pruning To further optimize our pose-estimator network performance, we prune some unnecessary operations from the original model that do not have any model weight. In particular, we prune the last few operations, which consist of transformation and normalization operations. In the ultralytics ONNX implementation of YOLOv8s-pose [22], we take the outputs from node ids 388, 403, 418, 336, 350, and 364, which contain the raw features of the detection and keypoints, and trim the rest of the node below it, effectively, increasing the speed of the model inference.

Post-Processing In addition to pruning, we implement a custom post-processing to extract the bounding box and keypoints coordinates. This post-processing step receives input from the output (6 sets of arrays) of the pruned pose estimator model. The first 3 sets from the output nodes [388, 403, 418] are for detection, we process these outputs by calculating the coordinates of the bounding boxes based on the grid cell offsets and box dimensions, then apply the sigmoid function to the scores, and then process the detection from multiple scales (output0, output1, and output2) into a single array of bounding box proposals. The second 3 sets from the output nodes [336, 350, 364] are for keypoints, we process the outputs to extract keypoint proposals and confidence scores. It calculates the coordinates of the keypoints based on the grid cell offsets, applies the sigmoid function to the confidence scores, and scales the coordinates. Then, we convert the keypoints from multiple scales (output0, output1, output2) into a single array with 51 values of keypoint proposals, because we employ COCO-keypoints [15] which have 17 keypoints, hence, each 3 values represent the coordinates (x, y) and score (z). Lastly, it filters out low-confidence detection, applies non-maximum suppression (NMS) to remove redundant bounding boxes, and returns the filtered detection and keypoints.

3.4.2 Quantization

For implementation on the embedded device, 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.

Uniform Affine Quantizer According to [11], this quantization method yields the smallest loss for most models. Assume we have a variable with a range of (X_{max}, X_{min}) where $X \in \mathbb{R}$ and want to quantize it to a range of $(0, N_{precission})$ where $N_{precission}$ is the amount of precision we desire. We use the following formula to quantize the variable.

$$X_{int} = round(\frac{X_{float}}{\Delta}) + z \tag{1}$$

$$X_{Quantize} = clamp(0, N_{precission} - 1, X_{int}) \tag{2}$$

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:

$$X_{float} = (X_{Quantize} - z)\Delta \tag{3}$$

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

Dynamic Fixed Point The second quantization method that we use is Dynamic Fixed Point [9]. This method follows the formula below to quantize a given value

$$X_{int} = round(X_{float} * 2^{fl}) \tag{4}$$

$$X_{Quantize} = clamp(N_{min}, N_{max}, X_{int}) \tag{5}$$

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

$$N_{max} = 2^{N_{precission}-1} - 1 \tag{6}$$

Rockchip RV1126 EVB

CPU	Quad-core Arm Cortex-A7 @ 1.5 GHz + RISC-V MCU @ 400 MHz
NPU	2.0 TOPS
Memory	32-bit DDR3

Figure 4. Rockchip RV1126 EVB specifications

$$N_{min} = -(2^{N_{precission}-1} - 1) \tag{7}$$

We set N_{min} as -127 and N_{max} as 127 for 8-bit precision (int8) and N_{min} as -255 and N_{max} as 255 for 16-bit precision (int16). We use this method to quantize our model to int8 and int16.

3.5. Model Conversion

To utilize the full capability of the Rockchip NPU module, we need to convert the model to Rockchip's file format called RKNN. First, we must convert our PyTorch model to a torchscript module or ONNX. In this experiment, we choose to convert our model to ONNX with an offset of 11 because of the capability and ease of use of the ONNX format. After converting our model to ONNX, we can convert it to RKNN using Rockchip's [9] built-in conversion tool.

3.6. Real-time Inference on Embedded System

The real-time inference pipeline, illustrated in Figure 3, integrates our method into embedded systems for efficient action recognition. We utilize the converted model (RKNN model) of the pose estimator and action recognition model for implementation on the embedded device. Connecting a webcam or CCTV camera to our embedded device initiates the process. Upon capturing frames, the embedded device simultaneously processes pose estimation for individuals within the scene. Utilizing every 64 frames, the system generates joint heatmaps and to optimize temporal information, we then uniformly sample 32 frames, ensuring a comprehensive understanding of the motion dynamics. We then convert the generated heatmap from RGB to grayscale as detailed in Section 3.2, which serves as the input for our action recognition module, which will classify the person's action within the frame.

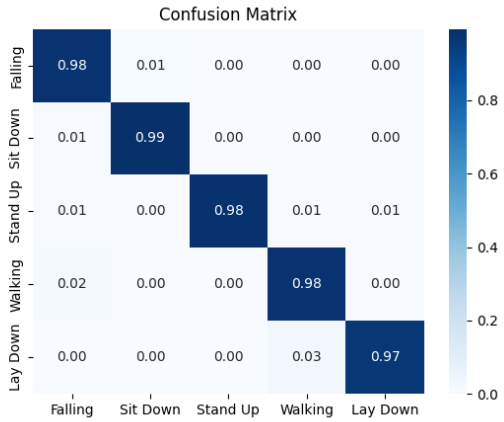


Figure 5. Confusion matrix illustrating the classification performance of the proposed model on [20] dataset, with rows corresponding to actual classes and columns corresponding to predicted classes.

Model	Accuracy (%)	Parameters (M)	Model Disk Size (MB)	Action Proc. Time
Duan *et al.* [6]	98.26	2.0	16.2	392.50
Noor *et al.* [20]	97.93	0.5	4.3	26.40
Ours	**98.35**	**0.1**	**0.8**	**4.10**

Table 2. Comparison with the existing method.

4. Experiments

4.1. Implementation Details

We train our model with stochastic gradient descent as the optimizer with a momentum of 0.9 and weight decay of 0.001. The learning rate is set to decay with cosine annealing [17], the initial learning rate is 0.01 and the minimum learning rate is 0.0001. We use the first 10 epochs for warm-up [8] in our training. Training is done in 150 epochs.

Training and testing are performed on an Intel i7-11700K CPU with 64GB RAM and NVIDIA RTX 4090 GPU. Following the model evaluation on the GPU, we proceed to implement the real-time system on an embedded device. Specifically, we deploy the trained model onto the EVB Rockchip RV1126 embedded device, which is characterized by limited computing and memory resources suitable for edge computing applications. Figure 4 shows the device's specifications.

As discussed in Section 3.4, this deployment process involves optimizing the model for inference on the embedded device. By implementing the real-time system on the embedded device, we demonstrate the feasibility and practicality of our approach in real-world scenarios with resource-constrained environments.

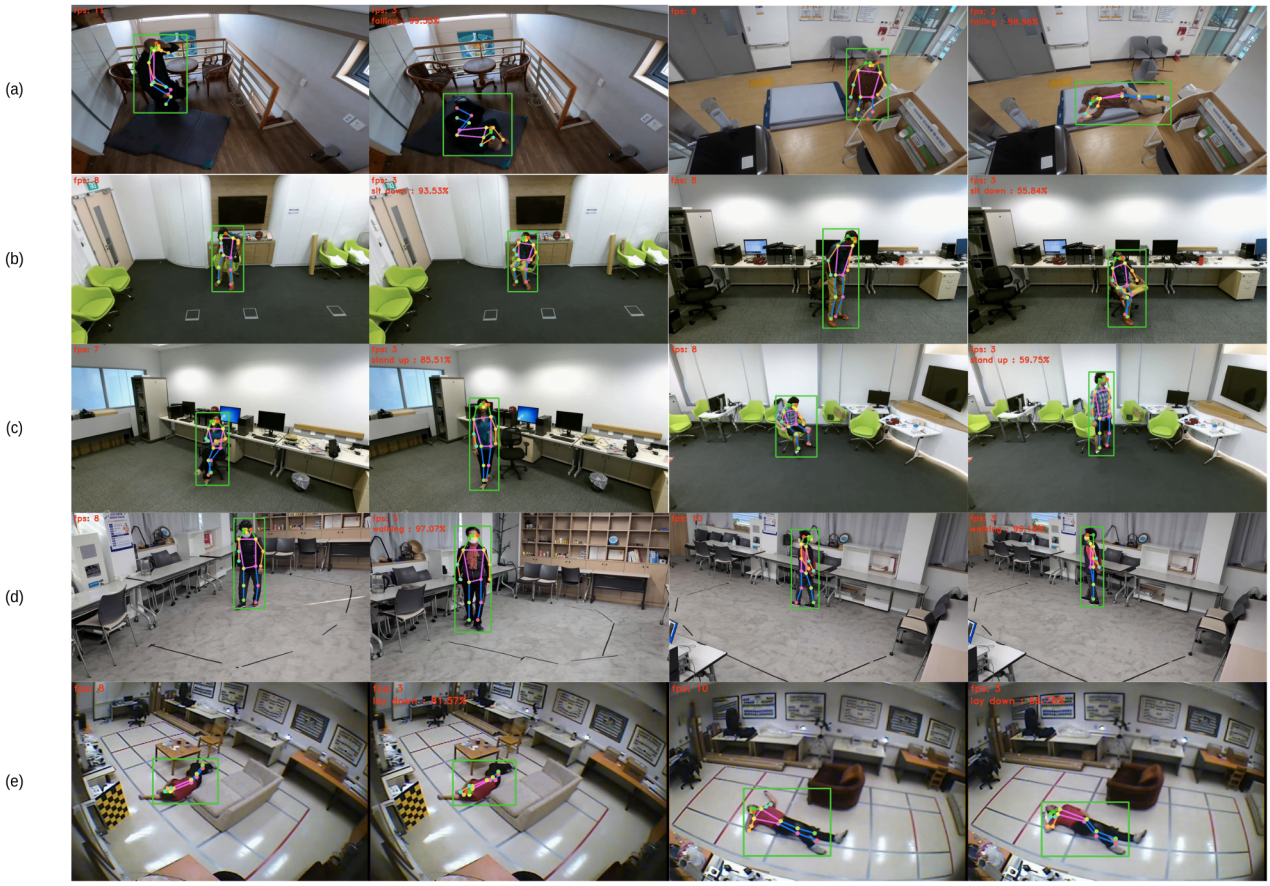


Figure 6. Example of predictions from the proposed model implemented on the embedded devices. (a) Falling, (b) Sit Down, (c) Stand Up, (d) Walking, (e) Lay Down.

Evaluation metrics. To evaluate our proposed model's computation efficiency, we calculate the number of parameters (params), its disk size, processing speed of each module, and FPS. Note that the FPS performance is calculated during inference by streaming the input video and then running all the processes in real-time. Therefore, as explained in Section 3.6, action recognition is not performed on every frame, only the pose estimator runs on every frame.

Dataset. We train and test our method by using the 5 action combined dataset (NTU RGB+D [21], NW-UCLA [23], URFD [12], Multiple Cameras Fall Dataset [2], and AI Hub [1]) proposed in [20]. 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.

4.2. Experimental Result

First, we validate our trained model accuracy on the 5 action combined dataset [20] on GPU. Second, we evaluate our model implementation performance on the embedded device.

4.2.1 Performance of Action Recognition on GPU

We assess the performance of our model on GPU before deploying it on the embedded device. We evaluate the accuracy of our proposed models across different classes by examining the confusion matrix shown in Figure 5. Here, each row represents the instances of the ground truth label, while each column represents the instances in a predicted class. The diagonal elements of the matrix indicate the number of correctly classified instances for each class. Off-diagonal elements represent miss-classification, where the row class is the actual class, and the column class is the predicted class. The confusion matrix shows that our model demonstrates robust performance across all classes, indicating its generalization capability and effectiveness.

Action Recognition Model	Pose Estimator Model	Accuracy (%)	Pose Proc. Time (ms)	Action Proc. Time (ms)	Average FPS
Quantized i16	Quantized i16	89.00	204.00	166.00	4.31
	Quantized i8	85.10	84.20	133.40	10.13
	Quantized u8	72.60	83.20	132.40	10.20
Quantized i8	Quantized i16	90.60	211.0	103.60	4.46
	Quantized i8	69.80	85.80	76.40	10.75
	Quantized u8	74.60	81.70	61.80	10.80
Quantized u8	Quantized i16	91.10	214.90	108.40	4.28
	Quantized i8	88.50	84.90	66.40	10.70
	Quantized u8	82.10	83.00	68.30	10.90

Table 3. Comparison of the proposed quantized model on an embedded device (Rockchip's RV1126). i16, i8, and u8 refers to int16, int8, and uint8, respectively. The cells with orange color represent the best performance. The row with green color is the model we choose for our real-time action recognition system on an embedded device.

Device	Accuracy (%)	Pose Estimator Latency (ms)	Action Recognition Latency (ms)	FPS
GPU	98.35	15.35	4.10	19.96
CPU	98.35	55.73	25.95	11.65
Embedded Device	88.5	84.9	66.4	10.70

Table 4. Speed performance comparison between devices. Note that FPS is computed on a set of 64 frames.

In Table 2, we compare our results with other methods, our model is 20 times smaller, 28 times faster, and 0.09% increase in accuracy compared to the baseline model [6]. Moreover, compared with [20], our proposed model is 5 times smaller yet achieves a higher accuracy by 0.42%.

4.2.2 Performance Result on Embedded Device

Performance of the quantized model on an embedded device. We evaluate the performance of the different quantized models. We infer the model 50 times on Rockchip's RV1126, followed by averaging the results obtained from each algorithm. As presented in Table 3, we observe that the performance of our proposed models is influenced by the pose estimator's performance. Notably, our models quantized to int16 exhibit lower accuracy and slower processing speeds. Conversely, our models quantized to uint8 show improved performance in both accuracy and speed. However, selecting a pose estimator model quantized to uint8 results in decreased performance in action recognition, with only a marginal decrease in FPS. Therefore, we opt for the proposed action recognition model quantized to uint8 and pose estimator models quantized to int8.

Speed performance between devices. As shown in Table 4, The model we use in the embedded device is the quantized model of our proposed model, with the pose estimator quantize from fp32 to int8, and action recognition quantize from fp32 to uint8. As expected, implementation on the embedded device is slower, with a 0.95 FPS decrease compared to CPU and a 9.26 FPS decrease compared to GPU performance.

Model	System Memory	NPU Memory	Total Memory	Maximum Allocation	Total Allocation
Not quantize	29.06	14.70	43.76	87.52	112.94
Quantize i16	11.48	6.48	17.97	35.93	47.98
Quantize i8	11.04	6.12	17.16	34.32	43.90
Quantize u8	11.00	5.92	16.91	33.83	43.39

Table 5. Comparison of memory usage (MB) of each proposed quantized model.

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
- Total memory: The sum of system memory and NPU memory
- Maximum allocation: The peak memory usage, it indicates the maximum allocation value of memory from the beginning to the end of the model inference
- Total allocation: It represents the sum of all memory allocated during the operation of the RKNN model

The findings presented in Table 5 indicate that quantized models require only half the memory compared to the non-quantized model.

4.3. Ablation Studies

The effect of channel size. In Table 6, we present the results of our ablation study, focusing on the effects of varying

Channel Sizes	Parameters (M)	Accuracy (%)	Latency (ms)
$\begin{bmatrix} Conv_1, 32 \\ Res_2, 32 \\ Res_3, 64 \\ Res_4, 128 \\ Res_5, 128 \end{bmatrix}$	1.3	**98.70**	7.69
$\begin{bmatrix} Conv_1, 16 \\ Res_2, 16 \\ Res_3, 64 \\ Res_4, 128 \\ Res_5, 256 \end{bmatrix}$	0.7	98.47	6.53
$\begin{bmatrix} Conv_1, 8 \\ Res_2, 8 \\ Res_3, 16 \\ Res_4, 32 \\ Res_5, 64 \end{bmatrix}$	**0.1**	98.35	**4.10**

Table 6. Comparison of performance across different channel sizes in the proposed network.

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 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.

The effect of input size. We investigate the impact of different input resolutions on the performance of the action recognition module. From the results presented in Table 7, we observe a trade-off between input resolution and model performance. While utilizing an input resolution of 56×56 yields a commendable accuracy of 97.29% with a relatively low latency of 3.75ms, opting for a higher resolution of 112×112 enhances accuracy to 98.35% at the expense of a slightly increased latency of 4.10ms. Considering this trade-off, we opt for the latter configuration 112×112 to strike a balance between accuracy and latency, thereby ensuring optimal performance for our application.

5. Discussion and Future Research

Our experimental results (Section 4.2) demonstrate the effectiveness of our proposed method in achieving accurate action recognition on GPU, CPU, and embedded devices. The validation of our model accuracy on the GPU showcases its ability to accurately classify actions across various scenarios. Furthermore, evaluating our model implementation on the embedded device confirms its suitability for real-world deployment, even with limited computational resources. The qualitative results[1] depicted in Figure 6 further reinforce the robustness of our method in correctly classifying each action class. These findings show the potential of our approach to contribute to applications such as surveillance, robotics, and autonomous driving, where real-time action recognition is crucial.

Input Resolution	Accuracy (%)	Latency (ms)
56×56	97.29	3.75
112×112	98.35	4.10

Table 7. Comparison of performance across different generated heatmap resolutions in the proposed network.

While our method shows promising performance, some areas require further improvement and exploration. Specifically, future research could focus on enhancing the efficiency and speed of our model implementation on embedded devices, potentially through optimization techniques, such as knowledge-distillation. Moreover, it would be beneficial to expand the scope of our experiments by utilizing diverse datasets to evaluate the generalization capabilities of our model across different action recognition scenarios.

6. Conclusion

In this paper, we proposed a novel approach to address the challenges of action recognition on embedded devices. We developed a real-time action recognition system capable of operating efficiently on resource-constrained devices by leveraging skeleton-based action recognition and adopting a lightweight 2D-CNN network architecture. Our proposed methodology offered a balance between computational efficiency and accuracy, making it suitable for real-world applications, such as surveillance systems. The experimental results demonstrated the effectiveness of our approach in achieving high recognition accuracy while maintaining real-time performance.

Acknowledgement

This work was supported by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No.RS-2022-00155915, Artificial Intelligence Convergence Innovation Human Resources Development (Inha University) and No.2022-0-00981, Foreground and Background Matching 3D Object Streaming Technology Development and No.2021-0-02068, Artificial Intelligence Innovation Hub).

[1] Video results available at https://youtu.be/c6EpnVlucWY

References

[1] AI Hub Senior Abnormal Behavior Video Dataset. https://aihub.or.kr/aihubdata/data/view.do?currMenu=115&topMenu=100&aihubDataSe=realm&dataSetSn=167, 2020. 6

[2] E. Auvinet, C. Rougier, J. Meunier, A. St-Arnaud, and J. Rousseau. Multiple cameras fall data set. *Technical report 1350, DIRO - Université de Montréal*, 2010. 6

[3] João Carreira and Andrew Zisserman. Quo vadis, action recognition? A new model and the kinetics dataset. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 4724–4733, 2017. 1

[4] Ke Cheng, Yifan Zhang, Xiangyu He, Weihan Chen, Jian Cheng, and Hanqing Lu. Skeleton-based action recognition with shift graph convolutional network. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 180–189, 2020. 2

[5] Yong Du, Wei Wang, and Liang Wang. Hierarchical recurrent neural network for skeleton based action recognition. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1110–1118, 2015. 2

[6] Haodong Duan, Yue Zhao, Kai Chen, Dahua Lin, and Bo Dai. Revisiting skeleton-based action recognition. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2959–2968, 2022. 1, 2, 5, 7

[7] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. In *Proc. IEEE/CVF International Conference on Computer Vision*, pages 6201–6210, 2019. 1

[8] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. 5

[9] Rao Hong. RKNN documentations. https://github.com/rockchip-linux/rknn-toolkit, 2023. 1, 4, 5

[10] Pranjal Jain, Shreyas Goenka, Saurabh Bagchi, Biplab Banerjee, and Somali Chaterji. Federated action recognition on heterogeneous embedded devices. *IEEE Trans. on Artificial Intelligence*, abs/2107.12147, 2021. 1

[11] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper, 2018. 4

[12] Bogdan Kwolek and Michal Kepski. Human fall detection on embedded platform using depth maps and wireless accelerometer. *Computer Methods and Programs in Biomedicine*, 117(3):489–501, 2014. 6

[13] Jungho Lee, Minhyeok Lee, Dogyoon Lee, and Sangyoun Lee. Hierarchically decomposed graph convolutional networks for skeleton-based action recognition. In *Proc. IEEE/CVF International Conference on Computer Vision*, pages 10410–10419, 2023. 2

[14] Maosen Li, Siheng Chen, Xu Chen, Ya Zhang, Yanfeng Wang, and Qi Tian. Actional-structural graph convolutional networks for skeleton-based action recognition. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3595–3603, 2019. 2

[15] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In *Proc. European Conference on Computer Vision*, pages 740–755. Springer, 2014. 2, 4

[16] Jun Liu, Gang Wang, Ping Hu, Ling-Yu Duan, and Alex C. Kot. Global context-aware attention LSTM networks for 3D action recognition. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 3671–3680, 2017. 2

[17] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *Proc. International Conference on Learning Representations*, 2017. 5

[18] Hongying Meng, Nick E. Pears, and Chris Bailey. A human action recognition system for embedded computer vision application. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2007. 1, 2

[19] M Monisha and Pooja S Mohan. A real-time embedded system for human action recognition using template matching. In *Proc. IEEE International Conference on Electrical, Instrumentation and Communication Engineering*, pages 1–5, 2017. 2

[20] Nadhira Noor and In Kyu Park. A lightweight skeleton-based 3D-CNN for real-time fall detection and action recognition. In *Proc. IEEE/CVF International Conference on Computer Vision Workshop*, pages 2171–2180, 2023. 1, 2, 5, 6, 7

[21] Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang Wang. NTU RGB+D: A large scale dataset for 3D human activity analysis. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1010–1019, 2016. 6

[22] Ultralytics. Pose - ultralytics YOLOv8 documentations. https://docs.ultralytics.com/tasks/pose, 2024. 2, 4

[23] Jiang Wang, Xiaohan Nie, Yin Xia, Ying Wu, and Song-Chun Zhu. Cross-view action modeling, learning, and recognition. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 2649–2656, 2014. 6

[24] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Proc. AAAI Conference on Artificial Intelligence*, pages 7444–7452, 2018. 2