Human Fall Detection using YOLO: A Real-Time and AI-on-the-Edge Perspective

Ali Raza[1,2], Muhammad Haroon Yousaf[1,2], and Sergio A. Velastin[3]

[1]Department of Computer Engineering, University of Engineering and Technology Taxila Pakistan.
[2]Swarm Robotics Lab, National Centre of Robotics and Automation (NCRA), Pakistan.
[3]School of Electronic Engineering and Computer Science, Queen Mary University of London, London E1 4NS, U.K.,
Email: ali.raza6@students.uettaxila.edu.pk[1], haroon.yousaf@uettaxila.edu.pk[1], sergio.velastin@gmail.com[3]

2022 12th International Conference on Pattern Recognition Systems (ICPRS) | 978-1-6654-6694-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICPRS54038.2022.9854070

***Abstract*—Researchers from all across the world are interested in human fall detection and activity recognition. Fall detection is an exciting topic that may be tackled in several ways. Several alternatives have been suggested in recent years. These applications determine whether a person is average, standing, or falling, among other activities. Elderly fall detection is vital among these activities. This is because it is a pretty typical and dangerous occurrence that affects people of all ages, with the elderly having a disproportionately negative impact. Sensors are typically used in these applications to detect rapid changes in a person's movement. They can be placed in smartphones, necklaces, and smart wristbands to turn them into "wearable" gadgets. These gadgets must be attached to the victim's bodies. This may be unsettling because we must constantly monitor this type of sensor. It is not always possible, and cannot be done in public settings with strangers. In this way, video camera image-based fall detection has several advantages over wearable sensor-based systems. A vision-based solution to fall detection is presented in this research. The suggested method's key component is that it can detect falls automatically on simple images from a typical video camera, eliminating the need for ambient sensors. It performs feature extraction based on the UR Fall self-annotated RGB dataset for fall detection. We used YOLO and its variants. YOLO allows for the detection of falls and a variety of actions for multiple people in the same scenario. As a result, this method using YOLOv1-v4 and tiny YOLOv4 can be employed in real-world situations using Raspberry-pi and OAK-D for edge solutions.**

***Index Terms*—Computer Vision, Fall Detection, YOLO, OAK-D**

I. Introduction

Nowadays, fall detection is one of the main concerns of the healthcare sector, causing physical and mental harm to people, especially the elderly in society. A significant chunk of society has entered into an aging era; people are paying more and more attention to the problem. Nowadays, many technologies are developing to prevent it. Falls are more likely to happen as people get older or have health problems like cardiovascular disease or muscle instability. If you want to maintain a self-regulating life, physical and mental health support will become critical. Therefore, different healthcare ecosystem stakeholders need a reliable fall detection and emergency assistance system.

barometers, inertial sensors, and gyroscopes have been used in some systems to identify falls. Sensors are typically used in these applications to identify rapid changes in a person's activity. These sensors can be integrated into smartphones, intelligent necklaces, or bracelets, turning them into "wearable" devices. The most significant disadvantage is that these devices must be attached to the subject's body, making monitoring sensor-dependent. Thus, people without sensors cannot be observed if they have any fall-like incidents. There are many such products to choose from on the market. The main disadvantage of these products is that they require significant hardware components and are expensive. Computer vision can be considered a more non-intrusive domain for fall detection despite these sensor-based approaches. The researchers have explored many different approaches and introduced a few image/video datasets.

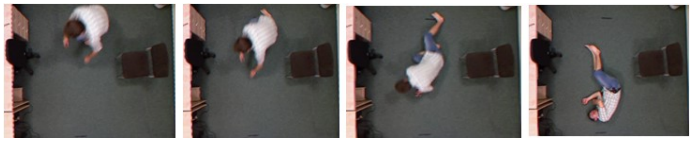

Fig. 1. fall sequence frames

The solution to fall detection may be considered in different ways. First, we have classical signal-based methods. Classical methods look at indicators from the video frames, e.g., the person's angle from the vertical, the rate of change of height of the person in the video, and predict whether a fall occurs based on some pre-decided conditions. This has certain obvious limitations, but it simplifies the problem too much. Actual falls are much more complicated and cannot be reduced to simply checking whether some features are below or above a particular threshold. Next, we have classical machine learning (ML) based methods. The ML-based methods extract features similar to those found in classical signal-based approaches. However, instead of having a pre-decided condition, these features are used to train a machine learning classifier that predicts whether a fall has occurred or not.

Geertsema et al. [1] use two types of datasets for fall detection along with SVM are used as a classifier. The accuracy and sensitivity of the experimental dataset Lei2 are 92% and 90%, respectively. The specificity and sensitivity of real-time dataset SEIN are 69% and 60-75%. Tsai et al. [2] provided the novelty in the proposed method by introducing the pruning technique; the NTU RGB-D dataset is used for training and testing. The accuracy of the NTU RGB-D dataset was reported as 99.4% and precision as 99.9%. Seredin et al. [3] proposed a technique for monitoring and identification falls by promoting privacy and unobtrusiveness. They proposed an intrusive fall detection system without wearable devices. One class classifier + CUSUM model is used to detect whether falls occur or not. The 91.7% accuracy was reported by them on the dataset used. Wisesa et al. [4] used different sensors, i.e., magnetometer, gyroscope, and accelerometer embedded in wearable devices for data collection. There are two types of phenomena, including ADLs and falls. The best validation accuracy was 100%, but the method is intrusive and has a disadvantage over non-intrusive mechanisms. Lin et al. [5] used two types of data named FAs and ADLs from a commercial smartwatch containing an accelerometer. ESA's method was used for preprocessing of data. After this, SVM and KNN classifiers were used to classify data and got specificity=98.92%, sensitivity= 96.09%, and a mean value is 97.45%. Kwolek et al. [6] used image-based data set for the analysis of fall detection. They used different classifiers and reported accuracy using KNN and SVM 95.83% and 91.67%, respectively. Yacchirema et al. [7] proposed a 3-D axial accelerometer embedded with a wearable device. They have employed four machine learning classifiers for the analysis. The average accuracy, precision, sensitivity, and specificity are reported as 98.72%, 98.72%, 94.60%, and 99.48%, respectively. Saleh et al. [8] showed the working of a fall detection system using a 3-D triaxial accelerometer. Three classifiers (SVM, KNN, and ANN) were used to detect falls efficiently. Results showed that SVM has the highest accuracy of 99.9%. Wang et al. [9] focused on the detection of falls using deep and shallow neural networks, SisFall datasets used in this study are publicly available. Four types of classifiers (SVM, DT, KNN, and extreme gradient boosting method (XGB)) were used. The best results are obtained that detect the fall occurrence with an accuracy of 99.9%.

Bourke et al. [10] used a total of 240 signals for the dataset. The threshold-based algorithm was designed using the upper fall threshold (UFT) and lower fall threshold (LFT). The results showed the specificity of the (UFT) = 100% and the sensitivity of the (UFT) = 83.3%. Mehmood et al. [11] used a wearable SHIMMER sensor for data collection. Mahalanobis Distance (MD) was used to find correlations and compare these features. If there is a similarity between ADL and sliding windows, then there is a high possibility of fall; otherwise, it is not considered a fall. Chandra et al. [12] used a gyroscope for the calculation of acceleration and a gyroscope for the detection of falls in terms of angular velocity. The gyroscope and accelerometer are both used to measure the difference in values between jumping and falling positions. So, this method detects falls and activates the alarm when older adults fall with an accuracy of 95.53% by comparing the threshold value with the acceleration at the time of fall. Lee et al. [13] used a built-in accelerometer in the smartphone, which is assumed to be kept in the front pocket to extract SAM, ZMean, and Xmean features. If the value of SAM, Zeman, and Xmean is greater than the threshold, then a fall occurs, and the message is sent to the medical center for assistance with 99.38% accuracy. Lee et al. [14] fall detection algorithm was designed by keeping in view thresholding techniques. Four types of threshold methods and three types of decision tree methods are used to analyze the dataset. The decision tree classifier is used to classify falls and ADLs with an accuracy of 97%.

There are many vision-based datasets available; e.g., in SDU-Fall, [15] 10 subjects were used for dataset collection. The fall to the floor is considered in this dataset and was captured using one Microsoft Kinect camera. EDF-OCCU [16] used 5 to 10 subjects for dataset collection and fall types were in different directions. UR-FALL [17] used two Microsoft Kinect cameras with five subjects for dataset collection. LARSEN [17] collected the dataset using eight infrared cameras; this dataset was captured with the occluded place's TST v2 [18] used 3 Microsoft Kinect cameras with 264 sequences for the collection of the dataset. PKU-MMD [19] used 3 Microsoft cameras with 66 subjects and contained six sequences and ten interaction actions for dataset collection. In the NTU RGB-D [20] dataset, 3 Microsoft cameras with multiple actors were used; this dataset is captured with 155 different camera viewpoints.

In this research work, we have tried to explore the potential of the popular YOLO and its variants for fall detection. The following are the key contributions to this research work:

- We have produced the annotations for the UR Fall dataset for the YOLO Darknet framework.
- We have implemented and empirically compared the YOLOv1, YOLOv2, YOLOv3, YOLOv4, and Tiny-YOLOv4 for fall detection.
- We have used the optimized model and deployed it on the edge device, i.e., OAK-D. Once all models had been trained, we used OpenVino to convert YOLO weights for edge inference.

II. Methodology

A series of steps must be carried out to achieve real-time fall detection. The proposed methodology is shown in Fig. 2, in which the dataset is acquired, and each image is annotated explicitly. Then, the annotated data is split into training and testing before being passed to deep learning models, i.e., the YOLO family, for custom model training. The weights obtained after training are used for model performance evaluation on testing data. The custom weights are then converted into the OpenVino IR format to perform real-time detection on the OAK-D and Raspberry Pi as the host computers. The detailed methodology is explained in the subsequent sub-sections.

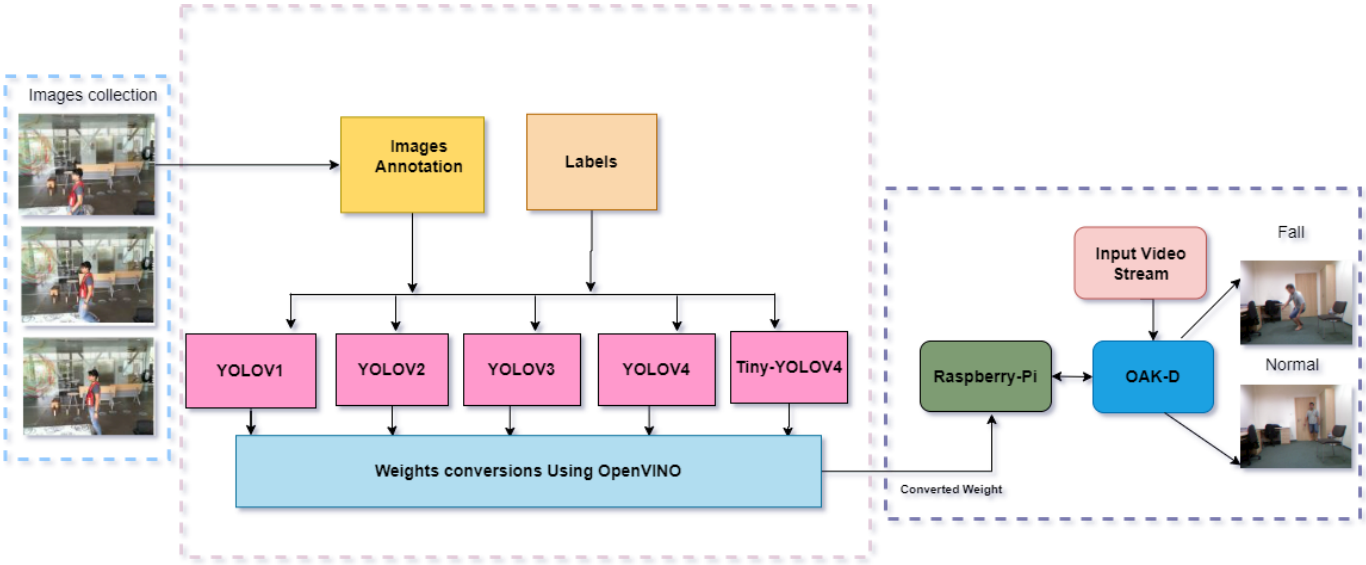

Fig. 2. Proposed Methodology

A. Baseline

Many approaches for real-time fall detection are discussed. Include threshold and sensor-based techniques in the literature section. However, vision-based techniques are better as compared to the others. Techniques due to non-intrusiveness. Deep learning models seem more capable of detecting falls in real-time efficiently and accurately. The same has been chosen after examining the available approaches. As a baseline to explore the fall detection solution, you only look once (YOLO); a classification/detection model is selected as a baseline to explore the fall detection solution.

B. Dataset

The UR fall detection dataset, which is publicly available online [21], is used in this research work. This dataset includes 70 sequences (30 falls + 40 activities of daily living). Falling events are captured using two Microsoft Kinect cameras and an accelerometer sensor. Table I shows the dataset details.

TABLE I
UR FALL DATASET

Sr.NO	Class Type	UR fall extracted frames Dataset
1	Fall	1691
2	normal	1731
	Total	**3201**

C. Data-Driven Model

Experiments are performed using 70% of the datasets for training and the remaining 30% for testing. Similarly, ten rounds of cross-validation are carried out for real-time fall detection.

D. Techniques and Architectures

In our case, fall detection, an open-source neural network framework written in CUDA and C languages, is used, making it extremely fast, which is important for real-time detection. We have used YOLOv1, YOLOv2, YOLOv3, YOLOv4, and Tiny-YOLOv4 for real-time fall detection.

E. YOLO Architecture and Family

Fall detection is explored differently in the YOLO framework. Using the entire image as a single instance predicts the bounding box coordinates and class probabilities for these boxes. The main advantage of YOLO is that it is easy to implement. It can also process 45 frames per second, which is exceptionally fast. YOLO understands the concept of generic object representation; that is why one of the best object detection algorithms is comparable to the R-CNN algorithm in terms of performance.

YOLO divides the input image into grid cells. Each cell is responsible for detecting an object and predicting its bounding box coordinates, classifying and localizing multiple objects in a single go. Unlike other algorithms that require a scan of the input image multiple times, YOLO performs this. This approach introduced a turning point in the field of real-time object detection. For fall detection, YOLO will be trained using images of fall and normal along with their associated Y labels, i.e., for our case of fall detection, there are two separate object classes and a 3 X 3 X 16 grid cell.

The new image will be divided into the same number of grids as during the training period. The model predicts a 3 X 3 X 16 output for each grid. This prediction's 16 values will be the same format as the training labels. The first eight numbers correspond to anchor box 1, with the first value being the probability of an object in that grid. The bounding box coordinates for that object will be values 2-5, and the

final three numbers will tell us which class it belongs to. The following eight anchor box values will follow a similar pattern; then, the predicted boxes will be used for the Non-Max suppression to obtain a single prediction per object.

III. Experimentation and Results

A. Dataset Annotation

After acquiring the dataset, the next step was to annotate it manually. Therefore, we used label image to do so. It is a free graphical picture annotation application that generates YOLO darknet labels. The annotations should be in YOLO format (object-class¿ x¿ y¿ width¿ height¿) for training the dataset.

B. Experimentation Setup

A significant difference between YOLO and region-based object detection algorithms such as Faster R-CNN is that YOLO predicts bounding boxes and class probabilities for these boxes with only a single neural network. In our situation, we employ YOLO as a fall detector to determine whether or not a fall occurred. One benefit of these bounding boxes is that they can be used for localization. YOLO is a detection system that can both localize and classify an object. YOLO does not occur for pure classification purposes; instead, it occurs for detection as YOLO uses both localization and classification.

The training of the YOLO and its variants is carried out on a machine having a Titan XP GPU. For the fall detection, the dataset folder has been named obj, consisting of images corresponding to its label .txt file. A total of 1691 fall images and 1731 normal images were split into training and testing, with ratios of 70% and 30%, respectively. After all this, we have 2395 images for training and 1027 images for testing. Similarly, ten rounds of cross-validation were carried out for real-time fall detection. The files required for training are obj. names, which provide the names of the classes, and obj. data, that contains the number of classes and the paths to the train, test, and backup folders. A backup folder is created to save the weights after every 100th iteration. The primary file required for training is the configuration file, which is to be changed according to the model requirements. In our case, each model has been trained for 20,000 maximum iterations, with the batch size of 64 having sub-divisions of 32 and a learning rate of 0.001 being enclosed in the .cfg configuration file. Moreover, the filters were set to 18 according to the formula filter size = (class+5) *3, where class = 2 in the cfg file. Finally, existing pre-trained weights for the model we wish to train for higher performance can be downloaded from Google. YOLO v1, v2, v3, v4, and Tiny-YOLO v4 pre-trained weights were downloaded for training and used as transfer learning for fall detection.

C. Evaluation Measures

Following are the evaluation measures used in this research work:

- True positives (TP): “Fall” detected as “Fall”
- False positives (FP): “Not Fall” detected as “Fall”
- True negatives (TN): “Not Fall” detected as “Not Fall”
- False negatives (FN): “Fall” detected as “Not Fall”

$$Precision = \frac{TP}{TP+FP} \tag{1}$$

$$Recall = \frac{TP}{TP+FN} \tag{2}$$

$$F1 = \frac{2*Precision*Recall}{Precision+Recall} = \frac{2*TP}{2*TP+FP+FN} \tag{3}$$

D. Results

Table II summarizes the fall detection results, mAP, precision, recall, F1-score, and average inference time per frame for all the trained deep learning models. Tiny-YOLOv4 showed the highest mAP of 96% . The inference time of Tiny-YOLOv4 is the lowest of all at 6.71 ms. The overall evaluation parameter of other deep learning models has been shown in Table III, YOLO divides the whole image into grid cells of equal sizes. Each cell is responsible for detecting the object if it lies in the center of the cell. Furthermore, Tiny-YOLOv4 detects fall detection for our dataset accurately. Fig 3 shows the prediction results for the fall and normal frames. In Fig 4, we presented

Fig. 3. Fall and Normal Prediction results

the fall and normal frames for which our proposed approach made false-positive and false-negative predictions. These are the few instances where our approach was unable to predict and lacked accuracy for fall and normal frames.

E. Edge Implementation

To implement the proposed approach on an edge platform, we have selected the OpenCV AI Kit, i.e., OAK-D. The OAK-D device was chosen as a Spatial AI tool because it can run complicated neural networks while delivering depth through its left and suitable stereo cameras and detection with the 4K RGB main camera. Darknet is not optimized to run on the Myriad X VPU hardware, which is in Luxonis’s OAK-D. To run our custom model on OAK-D for real-time detection, we first need to convert our Darknet YOLO weights to OpenVino format. To do so, we do not have any direct conversion method for darknet weights. We first need to convert to TensorFlow .pb weights and then to OpenVino. We obtained the .blob file after the conversion, which we used on the OAK-D kit. We need a host computer (Raspberry Pi in our case) with a USB port to plug in OAK-D to make it computationally fast. Installing DepthAI, a computer vision library offered by Luxonis, is the next step. DepthAI is used to get our model fully operational. After installing the depthAI requirements, we were able to run a custom model on OAK-D. After training the YOLO family and its variants, we have performed inference on the edge

device, i.e., the Raspberry Pi. For Raspberry Pi inference, we have used trained weights. The Raspberry Pi 3 was chosen as an edge because it can run neural networks with OAK-D support.

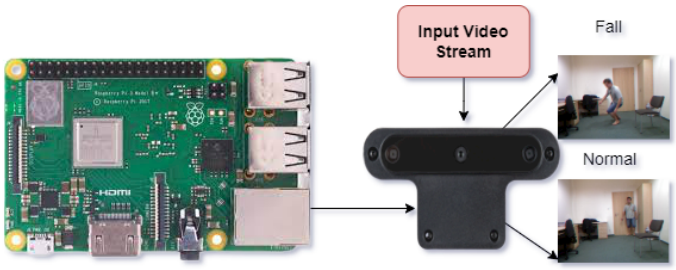

Fig. 4. Edge Inference

Fig. 5. False Postive and Negative Frames

TABLE II
FALL DETECTION RESULTS USING DIFFERENT VARIANTS OF YOLO

Model	Precision (%)	Recall (%)	F1-Score (%)	mAP	infer Time ms
YOLOv1	93.21	95.31	94	93	323
YOLOv2	94.22	95.83	94.2	94	90.7
YOLOv3	94.61	94	95	94.2	66.31
YOLOv4	94.8	95.21	95.3	95	42.51
Tiny-YOLOv4	95	96	95	95.2	6.71

TABLE III
REAL-TIME FALL DETECTION ON OAK-D AND RASPBERRY PI

Model	Detected Fall (%)	Accuracy (%)	FPS
YOLOv1	93.21	95.31	12
YOLOv2	94.22	95.31	13
YOLOv3	94.61	94	15
YOLOv4	94.8	95.21	15
Tiny-YOLOv4	95	96	26

F. Comparison

We have compared our achieved results with other techniques; table TableIV shows that our proposed approach using the Tiny-YOLOv4 trained model has performed better in fall detection and minimum inference time. The work presented in [22] used UR Fall dataset that led to an overall 94.99% accuracy using SVM, but our approach has achieved the best mAP of 95.2, which is more accurate as compared to this approach, whereas Lahiri et al. [23] presented the work using UR fall dataset and achieved accuracy 92.5 with higher inference time. As compared to this method, our approach achieved better results with less inference time. Kwolek et al. [24] improved fall detection by the use of a depth sensor and an accelerometer and achieved results with an accuracy of 92.5, which is less compared to our variants of YOLO. Our approach, Tiny-YOLO, achieved real-time results in terms of FPS, which is 26 Furthermore, the same is reported in TableIV.

TABLE IV
COMPARISON WITH OTHER EXISTING WORKS

Contribution	Dataset-Type	Methods	Evaluation(%)
Kwolek et al [22]	UR Fall	SVM	Accuracy=94.99 Precision=89.57 spectivity=91.25
Lahiri et al [23]	UR Fall	ML	Accuracy=92.5
Kwolek et al. [24]	UR Fall	KNN, SVM	KNN Accuracy=95.83 SVM Accuracy=91.67
Our Appoach	**UR Fall**	**Tiny YOLOV4**	**mAP=95.2 Precion=95 Recall=96 F1-Score=95**

IV. CONCLUSION

In this paper, a computer vision-based fall detection system has been proposed using YOLO and its variants. The research concluded that YOLO variants are effective for detecting human falls on edge devices in real-time. We have improved the accuracy of real-time human fall detection by using the YOlO algorithm and its variants for the UR fall dataset. As a result of this study, Tiny-YOLOV4 was identified as the most appropriate model for real-time fall detection. This same principle can be further extended to human activity recognition. In the future, we can extend this study by exploring alternative deep learning models for human fall detection and conducting an experimental study on human fall detection using different techniques and datasets. We are also exploring the use of pose estimation techniques to develop a real-time fall detection system. For pose estimation as a backbone, we intend to use different pose extractors, alphapose, blazepose, and openpose, to classify whether fall occurs or not. For fall detection, we also use vision transformers that employ attention mechanisms.

ACKNOWLEDGMENT

The authors acknowledge the funding from National Centre for Robotics and Automation (NCRA) for this research work.

REFERENCES

[1] E. E. Geertsema, G. H. Visser, M. A. Viergever, and S. N. Kalitzin, "Automated remote fall detection using impact features from video and audio," *Journal of biomechanics*, vol. 88, pp. 25–32, 2019.

[2] T.-H. Tsai and C.-W. Hsu, "Implementation of fall detection system based on 3d skeleton for deep learning technique," *IEEE Access*, vol. 7, pp. 153 049–153 059, 2019.

[3] O. Seredin, A. Kopylov, S.-C. Huang, and D. Rodionov, "A skeleton features-based fall detection using microsoft kinect v2 with one class-classifier outlier removal." *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, 2019.

[4] I. W. W. Wisesa and G. Mahardika, "Fall detection algorithm based on accelerometer and gyroscope sensor data using recurrent neural networks," in *IOP Conference Series: Earth and Environmental Science*, vol. 258, no. 1. IOP Publishing, 2019, p. 012035.

[5] L. Chen, R. Li, H. Zhang, L. Tian, and N. Chen, "Intelligent fall detection method based on accelerometer data from a wrist-worn smart watch," *Measurement*, vol. 140, pp. 215–226, 2019.

[6] B. Kwolek and M. Kepski, "Improving fall detection by the use of depth sensor and accelerometer," *Neurocomputing*, vol. 168, pp. 637–645, 2015.

[7] D. Yacchirema, J. S. de Puga, C. Palau, and M. Esteve, "Fall detection system for elderly people using iot and big data," *Procedia computer science*, vol. 130, pp. 603–610, 2018.

[8] M. Saleh and R. L. B. Jeannès, "Elderly fall detection using wearable sensors: A low cost highly accurate algorithm," *IEEE Sensors Journal*, vol. 19, no. 8, pp. 3156–3164, 2019.

[9] G. Wang, Q. Li, L. Wang, Y. Zhang, and Z. Liu, "Elderly fall detection with an accelerometer using lightweight neural networks," *Electronics*, vol. 8, no. 11, p. 1354, 2019.

[10] A. Bourke, J. O'brien, and G. Lyons, "Evaluation of a threshold-based tri-axial accelerometer fall detection algorithm," *Gait & posture*, vol. 26, no. 2, pp. 194–199, 2007.

[11] A. Mehmood, A. Nadeem, M. Ashraf, T. Alghamdi, and M. S. Siddiqui, "A novel fall detection algorithm for elderly using shimmer wearable sensors," *Health and Technology*, vol. 9, no. 4, pp. 631–646, 2019.

[12] I. Chandra, N. Sivakumar, C. B. Gokulnath, and P. Parthasarathy, "Iot based fall detection and ambient assisted system for the elderly," *Cluster Computing*, vol. 22, no. 1, pp. 2517–2525, 2019.

[13] J.-S. Lee and H.-H. Tseng, "Development of an enhanced threshold-based fall detection system using smartphones with built-in accelerometers," *IEEE Sensors Journal*, vol. 19, no. 18, pp. 8293–8302, 2019.

[14] C. M. Lee, J. Park, S. Park, and C. H. Kim, "Fall-detection algorithm using plantar pressure and acceleration data," *International Journal of Precision Engineering and Manufacturing*, vol. 21, no. 4, pp. 725–737, 2020.

[15] X. Ma, H. Wang, B. Xue, M. Zhou, B. Ji, and Y. Li, "Depth-based human fall detection via shape features and improved extreme learning machine," *IEEE journal of biomedical and health informatics*, vol. 18, no. 6, pp. 1915–1922, 2014.

[16] Z. Zhang, C. Conly, and V. Athitsos, "Evaluating depth-based computer vision methods for fall detection under occlusions," in *International Symposium on Visual Computing*. Springer, 2014, pp. 196–207.

[17] B. Kwolek and M. Kepski, "Human fall detection on embedded platform using depth maps and wireless accelerometer," *Computer methods and programs in biomedicine*, vol. 117, no. 3, pp. 489–501, 2014.

[18] S. Gasparrini, E. Cippitelli, E. Gambi, S. Spinsante, J. Wåhslén, I. Orhan, and T. Lindh, "Proposal and experimental evaluation of fall detection solution based on wearable and depth data fusion," in *International conference on ICT innovations*. Springer, 2015, pp. 99–108.

[19] C. Liu, Y. Hu, Y. Li, S. Song, and J. Liu, "Pku-mmd: A large scale benchmark for continuous multi-modal human action understanding," *arXiv preprint arXiv:1703.07475*, 2017.

[20] J. Liu, A. Shahroudy, M. Perez, G. Wang, L.-Y. Duan, and A. C. Kot, "Ntu rgb+ d 120: A large-scale benchmark for 3d human activity understanding," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 10, pp. 2684–2701, 2019.

[21] B. Kwolek and M. Kepski, "Human fall detection on embedded platform using depth maps and wireless accelerometer," *Computer methods and programs in biomedicine*, vol. 117, no. 3, pp. 489–501, 2014.

[22] ——, "Human fall detection on embedded platform using depth maps and wireless accelerometer," *Computer methods and programs in biomedicine*, vol. 117, no. 3, pp. 489–501, 2014.

[23] D. Lahiri, C. Dhiman, and D. K. Vishwakarma, "Abnormal human action recognition using average energy images," in *2017 Conference on Information and Communication Technology (CICT)*. IEEE, 2017, pp. 1–5.

[24] B. Kwolek and M. Kepski, "Improving fall detection by the use of depth sensor and accelerometer," *Neurocomputing*, vol. 168, pp. 637–645, 2015.