A Security-Centric Deep Learning Enabled Camera Solution for Real-Time Human Fall Detection

Hamid Reza Tohidypour, Mahsa T. Pourazad, and Panos Nasiopoulos
Electrical & Computer Engineering, Univeristy of British Columbia
Vancouver, BC, Canada
{htohidyp,pourazad,panosn}@ece.ubc.ca

***Abstract*— Automatic human real-time fall detection is a challenging task in remote healthcare, demanding a non-intrusive, secure and affordable solution. In this paper, we present a real-time hardware system that uses a deep learning model for fall detection embedded in a color camera. To reduce the startup delay and achieve real-time performance for the inference phase, we optimized our model using TensorRT. In addition, we addressed the board memory limitation using virtual memory and linear memory allocation and garbage collection. Moreover, GStreamer was used to perform most of the video processing using Jetson's GPU. Our live evaluation shows that our system achieved the accuracy of 84.44% and real-time performance.**

Keywords— fall detection, hardware, deep learning, smart home

I. INTRODUCTION

Remote health and wellness monitoring has gained a serious attention in the past decade [1]. Research in the related fields is focusing on developing solutions that aim at monitoring and improving the wellness of the occupants. These solutions are expected to be easily incorporated into existing as well as future smart homes and reduce healthcare costs. Fall detection is one of the lifesaving services that smart homes can offer to all occupants, but it can be essential especially for elderly people and patients recently discharged from the hospitals.

The existing fall detection systems can be categorized into three categories: wearable devices, ambient sensors, and video cameras [2,3]. Although, the wearable systems are the cheapest option the fact that they have to be worn all the time makes them inconvenient, invasive, and intrusive for users. On the other hand, the ambient based sensor solutions suffer from poor signal to noise ratio and having a limited coverage. The video camera based solutions are the only available options that sense rich information and cover large area for health monitoring [4]. Privacy concerns that may be associated with this approach can be addressed by providing real-time solutions while not storing any data. Most of the existing video based fall detection approaches are based on traditional machine learning approaches and hand-craft features that require making several assumptions [5]. The most common assumption is that the monitored person goes quickly from the standing position to a lying position. These approaches mainly use inactivity detection, 3D head tracking, and body shape change or bounding box transformation analysis [6]. Only a few of the existing works rely on the assumption that a fallen person remains on the ground after the fall incident with little or no motion [7,8].

The recent approaches are designed based on deep learning to provide robustness to the system and reduce assumptions. For instance, the method proposed in [9] uses a combination of a convolutional neural network (CNN) and a long short-term memory (LSTM) network to detect fall incidents in real-time. However, this approach uses a large number of learning parameters that make it computationally expensive in practice. In addition, it was shown to be not robust due to fact that it was biased towards the background and the floor pattern used during capturing and thus present in the training dataset. In our previous work [10], we proposed a long-term recurrent convolutional network (LRCN) for fall detection. The network consists of a CNN that was specifically designed to extract features related to fall detection from the frames and an LSTM that decoded the temporal information sequentially to detect the incident of fall. This method achieved the accuracy of 85.70% outperforming the existing works. However, the main concern in all the video based fall detection methods is security and intrusiveness, with industry looking for solutions where the entire process takes place on cameras with limited built-in machine learning capabilities. Such solutions minimize the chances of information being hacked as it is transferred from the cameras to a more powerful central processing unit.

In this paper, we propose a light-weight deep learning network, which offers the desired accuracy for fall detection and yet can be embedded in a surveillance camera. Our prototype test solution uses a Jetson Nano and Raspberry Pi Camera V2. We optimize the start-up delay and the inference time to achieve real-time performance by serializing our model, optimizing the inference code, performing most of the video processing computations using GPU, and managing memory limitations of the Jetson Nano for our application. Our live evaluations show that our system achieved the accuracy of 84.44% and real-time performance.

II. PROPOSED METHOD

Our gool is to design an automatic real-time human fall detection home device that uses a camera with limited machine learning processing power to meet the requirements of a non-intrusive video based secure system. To address the privacy and security concerns, the hardware system should be able to work without the help of a cloud or a locally hosted computer interface. Our hardware consists of a camera running an on-boarded deep learning model which upon detection of a fall

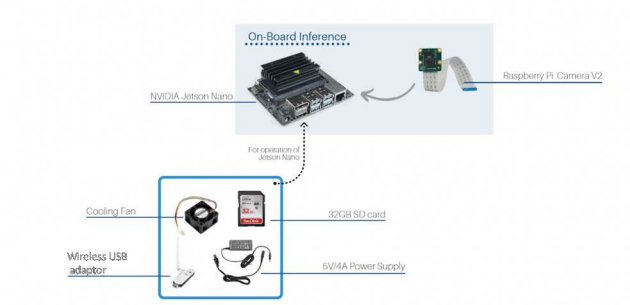

Fig. 1. Graphical representation of our prototype for fall detection.

sends a notification to a caregiver (see Fig. 1).The deep learning model for fall detection is based on our previous work presented in [10]. Our initial model was implemented using Keras deep learning framework and trained on a desktop setup, while it was optimized for a laptop for the inference phase. Thus, the model had to be optimized for the inference phase, which is supposed to be done on our hardware system with both limited memory and machine learning capabilities. In order to find the best computing board, given our deep learning model, our application, and cost, we conducted a thorough investigation of existing boards with machine learning capabilities available in the market. The hardware solution should be portable, non-intrusive, and affordable. Our investigation showed that the NVIDIA Jetson Nano, which is a small powerful computer that runs on Linux and can be used for deploying deep learning models, meets our needs and restrictions [11]. Jetson Nano is 70mm x 45 mm in size, making it an ideal portable board for our application. Table I shows the key features of the NVIDIA Jetson Nano. Flexibility in camera selection was one of the major other reasons for choosing the Jetson Nano. For the camera, we chose the Raspberry Pi Camera V2 is an 8MP CSI camera that is compatible with the Jetson Nano [12]. In order to create optimal working conditions for Jetson's GPU, we used a cooling fan to ensure the temperature of the system is less than 50^0 C. Fig. 1 depicts a graphical representation of our prototype, showing the NVIDIA Jetson Nano Board and the Raspberry Pi V2 Camera that we used.

A. *Setup for the NVIDIA Jetson Nano board*

Before being able to run the deep learning model on the Jetson Nano, the processor must be configured so it functions properly for our task. When transitioning the model from running on a laptop with a webcam to running on the Jetson Nano with a Raspberry Pi Camera V2, several configuration steps must be completed so that the Jetson Nano has a functioning environment for the deep learning model to run on.

The setup for the Jetson Nano can be broken down into 3 distinct sections: installation, configuration, and finalizing the processing and full setup. The installation step entails downloading all the necessary packages and dependencies needed to allow the deep learning model to run.

The first step is to install the Jetpack software, which is the NVIDIA Software Developer Kit for Jetson Nano [13]. This software includes all software development tools, the operating system for the processor, and all the libraries that the software depends on. This should be downloaded onto a microSD memory card and flashed into the processor.

TABLE I. KEY FEATURES OF THE SELECTED NVIDIA JETSON NANO

Key features		
NVIDIA Maxwell GPU 128-core	Quad-core CPU ARM A57@1.43 GHz	Memory: 4GB 64-bit LPDDR4 25.6 GB/s | 16GB Storage
Camera interface: 1x MIPI CSI-2 DPHY lanes	Connectivity -> Gigabit Ethernet, M.2 Key E| WiFi Adaptor Not included	Other GPIO -> I2C, I2S, SPI, UART

The second step involves the installation of TensorFlow and Keras. TensorFlow is an open source machine learning software library that the deep learning model uses. Keras is the high-level application programming interface (API) used for the training and development of deep learning models. We installed the Tensorflow and Keras that are compatible with the versions used to train our deep learning model on the desktop setup. In addition, we installed all the other dependencies and libraries needed to run the code.

The configuration step consists of configuring the Jetson Nano to read the camera's video stream and output it to its screen display. First, a webcam that was used with the previous desktop setup was used to begin the configuration process, since it was already compatible with the original code developed for the inference using a laptop in our previous work. We used that camera configuration as a preliminary step to set up the Raspberry Pi Camera V2, the camera we later used for our hardware setup. Small parameter changes were made in the code to achieve compatibility with the new camera.

To finalize the overall setup and have the model running, a 3D printed stand was made to mount the processor and camera, facilitating calibration during operation and testing.

B. *Addressing the memory limitations of NVIDIA Jetson Nano*

After completing the above steps, the Jetson Nano processor was able to support our deep learning model. Jetson Nano's onboard memory supports only 4GBs. However, running our deep learning model, which consists of about 56 million parameters, requires significant amounts of processing power and RAM, much greater than those supported by Jetson Nano. Therefore, memory optimization is a necessary step in order for Jetson Nano to support our model.

To overcome the lack of physical RAM available, a virtual RAM or commonly known as swap file was configured on the Jetson Nano. The Virtual RAM only starts shifting processes to the hard disk when the available physical RAM is critically low.

Despite installation of virtual memory, the existing deep learning model failed to run on the Jetson Nano. While stepping through the code during run-time, we discovered that the Keras based algorithm could not account for virtual memory available on the system, as it was trying to allocate as much memory as

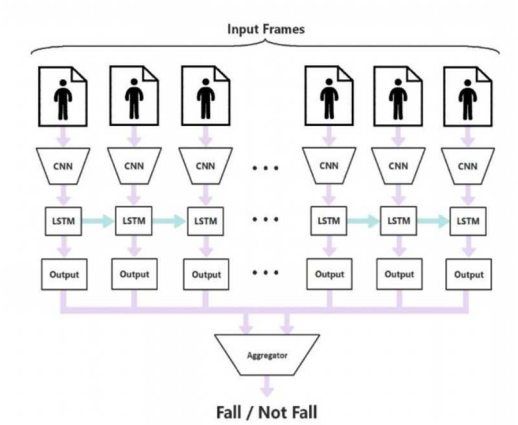

Fig. 2. Our deep learning network structure.

possible at the start. For this reason, the TensorFlow backend configuration was changed to slowly and linearly allocate memory during operation rather than allocating the total amount memory requested at the beginning by Keras.

This linear and slow memory allocation allowed the algorithm to take up all of the physical RAM & partially the virtual RAM. This memory allocation algorithm was able to run without crashing despite the long loading time. After ensuring that the model runs on Jetson Nano, we moved onto improving system performance through a series of design decisions/solutions.

C. TensorRT based Deep Learning Model Optimization

The memory allocation solution introduced in the previous subsection enabled the Jetson Nano to run the inference phase of our deep learning model without crashing. However, the system took about 17 minutes to load the model and start predicting fall scores. Even while predicting falls, there was a significant delay, partially due to the highly unoptimized nature of our deep learning model.

Recall that our deep learning network designed for automatic video based human fall detection consists of a convolutional neural network (CNN) and LSTM [10]. Our CNN extracts features from each video frame. The LSTM aggregates the features of the window frame which decodes sequential information and classifies into fall or not fall. Fig. 2 shows our network structure. We implemented our original network using Keras API [10]. Optimizing this deep learning architecture is a challenging task as the models (i.e., CNN and LSTM) were developed with fewer constraints (e.g., desktop environment).

We used TensorRT to optimize our model, as it is a machine learning framework proposed for NVIDA's Jetson boards allowing the deep learning models to be optimized for inference step on Jetson boards [14]. TenorRT optimizes models by maximizing the throughput using INT8 quantization precision (8-bit signed integer) tensors to optimize models. This optimization preserves the accuracy while performing layer and tensor fusion, using tensor memory, and introducing scalable design to process multiple batches of input (in the case of our model frames). As 99.91% of the learning parameters of our network belongs to CNN part, we focused on optimizing only the CNN part. To this end, the CNN part of the Keras based model was frozen and converted into a pure TensorFlow network graph definition. After a pure TensorFlow graph was obtained, it was fed through a TensorRT optimizer, which makes the model highly optimized for a specific GPU.

Utilizing TensorRT significantly reduced the required hardware cost without sacrificing quality and speed. At this point, the start-up time of the model running on Jetson was reduced to 1 minute and 54 seconds, as the CNN model was now serialized and didn't require to be built at every single run.

After resolving the start-up delay issue, we had to deal with the delay in inference time. One step towards addressing this challenge, was to maximize the clock speed, which regulates the CPU and GPU of Jetson. Although this change effectively reduced the latency, we realized that all video preprocessing was done at a software level, which caused significant strain on the system resources. This processing strain caused the video capturing and processing to be delayed by 3 seconds, a delay that violates our real-time requirement. To address this issue, we utilized a GPU accelerated GStreamer pipeline, to shift processing towards hardware image processors, and then feeding these preprocessed images into our model. Since the Raspberry Pi Camera V2 CSI camera supports multiple resolutions, we examined the use of different resolutions and frame rates, aiming at determining which combination may lead to real-time and yet accurate solution. After thorough testing, the 1080p @30Fps combination was found to be the perfect balance between speed and accuracy. By implementing all the above steps, Jetson was able to run our optimized deep learning model in real-time, processing 30 frames in less than 1 second.

D. Deleting Frames & Garbage Collection

The limited memory of the hardware requires better utilization when it comes to memory usage. This means, our system had to reuse memory when possible to achieve optimal conditions. A good coding practice to reuse memory is to free the memory space of variables that are no longer needed, and reuse them again instead of them occupied with unneeded previously used data. First, we delete previously used frames. This practice is to both protect user privacy and achieve non-intrusiveness, and to free up the memory space from the past frames. The other optimization was garbage collection on

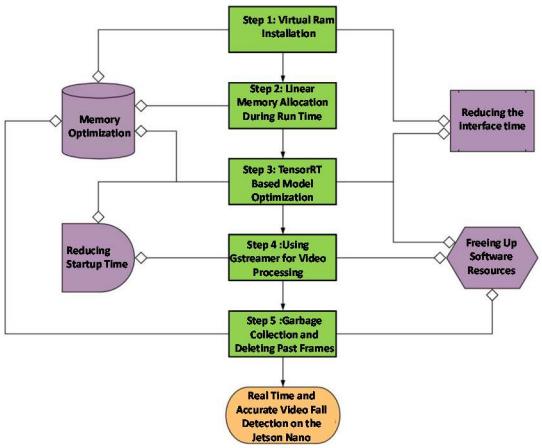

Fig. 3. Flow chart of steps taken to optimize our hardware setup.

variables when they are no longer needed.
This measure enabled us with a better memory utilization, and it helped us to run the model on the chosen hardware more efficiently and reduced the chances of frames stored in the buffer being accidentally stored or intentionally hacked to almost zero. Fig. 3 summarizes the steps that were taken to optimize our hardware and software setup to achieve an accurate, real-time video based fall detection system.

III. Performance Evaluation

In order to evaluate the performance of our method, we designed a series of live tests, consisting of a variety camera setups, i.e., viewing angle, distance and height. Table II shows the list and outcomes of live testing validation. Each test case was performed several times by different subjects. Results of fall and no fall classifications were categorized into 4 outcomes:

1. True Positive
2. True Negative
3. False Positive
4. False Negative

The live test scenarios include the following scenarios: front fall, walking, sitting, backwards falling, sideways falling, and falling to knees to floor. As it can be seen from Table II, our hardware system successfully classified 38 out of the total 45 tests, achieving the accuracy of 84.44%. Our performance evaluations showed that our system achieved this accuracy in real-time, processing 30 frames in less than 1 second.

IV. Conclusion

In this paper, we proposed a hardware implementation for real-time video based human fall detection for smart homes. In our implementation, Jetson Nano was the choice of a board with limited machine learning capabilities. Our initial deep learning model for fall detection, which was designed to run on a powerful processing unit, was modified accordingly to run on Jetson. In order to achieve real-time performance, we optimized our model using TensorRT. In addition, we addressed the memory limitations of Jetson using virtual memory, linear memory allocation, and garbage collection. Moreover, GStreamer was used to perform most of the video processing using Jetson's GPU. Our live test showed that our system achieved the accuracy of 84.44% in real-time.

Acknowledgment

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC – PG 11R12450), and TELUS (PG 11R10321). The authors would also like to acknowledge the work performed by Abdul Moiz, Alessandro Narciso, Kirsten Kwan, Winnie Gong, and Mohamed Hamdan.

References

[1] V. Gruessner, "The History of Remote Monitoring, Telemedicine Technology," [Online]. Available: https://mhealthintelligence.com/news/the-history-of-remote-monitoring-telemedicine-technology, 2015.

[2] L. Chen and I. Khalil, "Activity recognition: Approaches, practices and trends," in Activity Recognition in Pervasive Intelligent Environments, Atlantis Press, pp. 1-31, 2011.

[3] A. Singh, S. U. Rehman, S. Yongchareon, and P. H. J. Chong, "Sensor Technologies for Fall Detection Systems: A Review," IEEE Sensors Journal, vol. 20, no. 13. 2020.

[4] E. E. Stone and M. Skubic, "Fall detection in homes of older adults using the Microsoft Kinect," IEEE journal of biomedical and health informatics, vol. 19, no. 1, pp. 290-301, 2015.

[5] M. Zeng, L. T. Nguyen, B. Yu, O. J. Mengshoel, J. Zhu, P. Wu, and J. Zhang, "Convolutional neural networks for human activity recognition using mobile sensors," in Proc. 6th International Conference on Mobile Computing, Applications and Services (MobiCASE), 2014.

[6] X. Yu, "Approaches and principles of fall detection for elderly and patient," In Proc. of 10th International Conference on e-health Networking (HealthCom), Applications and Services, 2008.

[7] H. Nait-Charif and S.J. McKenna, "Activity summarisation and fall detection in a supportive home environment," In Proc. of the 17th International Conference on Pattern Recognition (ICPR), 2004.

[8] B. Jansen and R. Deklerck, "Context aware inactivity recognition for visual fall detection," In Proc. of Pervasive Health Conference and Workshops, 2006.

TABLE II. Results of the Fall Live Testing.

Camera Setup			Mat Angle to Camera	Number of Tests	Outcomes	Conclusion
Distance (m)	Height (m)	Angle (deg)				
2.5	1.5	30	Vertical	5	1, 1, 1, 1,1	Pass
			Horizontal	5	1, 1, 1, 1,1	Pass
			Diagonal	5	1, 1, 1, 1,1	Pass
2.8	1.5	30	Vertical	5	1, 1, 1, 1,1	Pass
			Horizontal	5	1, 1, 1, 1,1	Pass
			Diagonal	5	1, 1, 1, 1,1	Pass
3.1	1.5	30	Vertical	5	1, 4, 4,1,4	Fail
			Horizontal	5	1, 4, 1,4,4	Fail
			Diagonal	5	1, 4, 1,1,1	Pass

[9] Mahsa T. Pourazad and et al., “A Non-Intrusive Deep Learning Based Fall Detection Scheme Using Video Cameras,” International Conference on Information Networking (ICOIN), pp. 443-446, 2020.

[10] H. R. Tohidypour, A. Shojaei-Hashemi, P. Nasiopoulos, and M. T. Pourazad, “A Deep Learning Based Human Fall Detection Solution,” PErvasive Technologies Related to Assistive Environments (PETRA), July 2022 (Accepted).

[11] “Jetson Nano Developer Kit”, [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit. [Accessed: 15-May-2022]

[12] “Camera Module,” Camera Module - Raspberry Pi Documentation. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/camera/. [Accessed: 15-May-2022]

[13] “JetPack,” NVIDIA Developer, 11-Oct-2019. [Online]. Available:https://developer.nvidia.com/embedded/jetpack. [Accessed: 15-May-2022]

[14] “NVIDIA TensorRT Documentation,” NVIDIA Developer Documentation. [Online]. Available: https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html. [Accessed: 15-May-2022].