Deep Neural Networks for real-time remote fall detection

Andrea Apicella and Lauro Snidaro

Department of Mathematics, Computer Science and Physics
University of Udine
Udine, Italy
lauro.snidaro@uniud.it
apicella.andrea@spes.uniud.it

Abstract. We present a pose estimation, Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) based fall detection method. Our RNN takes time series of 2D body poses as inputs. Each pose is made of 34 numerical values which represent the 2D coordinates of 17 body keypoints and is obtained using a combination of PoseNet[8] and a CNN on RGB videos. Each series is classified as containing a fall or not. The proposed method can be configured to be suitable for real-time usage even on low-end machines. Furthermore, the proposed architecture can run completely inside the web browser, making it possible to use it without a dedicated software or hardware. As no data is exchanged with external servers, the proposed system preserves user privacy. Our implementation focuses on the detection of a single individual's fall in a controlled environment, such as an elderly person who lives alone or does not have continuous assistance.

Keywords: Remote Fall detection · Real-time · Deep Neural Networks · PoseNet · Elderly Fall Detection

1 *Introduction*

According to the reports of the World Health Organization [1], in 2018, falls represented the second cause of unintentional injury deaths worldwide, only exceeded by road traffic injuries. Consequently, this problem has generated a wide range of researches aimed at developing effective ways to detect such accidents. In this work, we approach the fall detection task developing a vision-based system, which employs data acquired with a single RGB camera and processed with Deep Neural Networks. We observed that many recent pieces of research which rely on complex techniques and demanding hardware setups obtained very high accuracy. At the same time, there is a lack of successful fall detection applications in the market. As a result, we aimed at developing a system which can be used by virtually anyone to detect falls of a single individual in a controlled

[1] World Health Organization. Fall. `https://www.who.int/news-room/fact-sheets/detail/falls`. Accessed November 12th, 2020

environment, such as a home. A broad adoption could lead to the expansion of publicly available fall datasets, which are few and mostly contain simulated falls. The key novelty of our method consists in performing the RGB video analysis and the classification predictions entirely inside the web-browser. Additionally, our architecture can be tuned to allow for real-time detection.

2 *Related Work in Computer Vision*

Recently, Computer Vision techniques based on deep learning managed to obtain great fall classification results, by employing different variations of CNNs, Recurrent Neural Networks and many combination of those. Fall detection researchers have pursued high classification accuracy scores for a long time. The introduction of new and more complex deep learning architectures is an instance of how the research focuses on the most common metrics, often neglecting real-time performance, suitability for low power devices and the possibility to make these technologies broadly available to consumers. One such example is the work from Cameiro et al. [2], who used a multi-stream approach to combine the classification results of three hand-crafted features into a single result predicted by an additional SVM classifier. The lack of large enough, real or simulated, falls datasets represented an obstacle that has been recently addressed in two main ways. The first way is trying to create new datasets from scratch, to have more falls videos which picture a single subject falling in a controlled environment. An example is the work by Pourazad et al. [9], who used four Microsoft Kinects, each capturing a different view, similarly to what Multicam's [1] authors did. Further instance is the work from Feng et Al. [3], who focused on falls which occur in scenes crowded with people, and created a new dataset of 220 videos. The other, opposite, way is the one followed by Lu et al. [7]. It consists in pre-training complex CNNs on large human actions datasets (Sports-1 M in their case) and then using transfer learning to fine tune the last layers on the smaller fall datasets or, as Lu et al. actually did, feeding the CNN output as input for a further LSTM network that was separately trained on much smaller fall datasets. The approach of recent pieces of research also varies with regard to the type of features used during training. Some experiment with "hand-crafted" features, such as skeletal data, as in [5] and in this work. Other researchers prefer using features automatically extracted from data by CNNs, like in Pourazad et al.'s and Lu et al.'s works, to obtain better robustness in real-life scenarios. Although over the past few years the main focus was to achieve high accuracy scores, some attempts to streamline the architectures have been made recently. Han et Al. [4] replaced the traditional VGG16 network architecture with a simplified and slimmed down model. Compared to the original one, this new model occupied as much as 63 times less memory and required half the time to train, while achieving nearly identical accuracy scores. In our opinion, this kind of optimization is a right step to make the best fall detection systems usable on low-power devices and is similar to the assumptions that made us set our work on complete compatibility with the web environment.

3 *Dataset*

Despite the large number of pieces of research published in recent years around the problem of fall detection, there is a poor choice of datasets containing fall videos. Factors, such as the accidental nature of a fall event and the privacy of the subject involved, led to the creation of datasets which include simulated falls performed by actors in a controlled environment. After several researches, we were able to find two datasets that could fit our needs: UR Fall Detection Dataset [6] and Multiple Cameras Fall Dataset [1].

3.1 UR Fall Detection Dataset

UR Fall Detection Dataset (URFall) includes 70 video sequences, 30 of which containing a fall and the other picturing activities of daily life (Fig.1). All actions were recorded with two Microsoft Kinect cameras, one shooting horizontally and one vertically. The 30 fall sequences, unlike the ones containing ADL, were also measured with accelerometers. However, we used video data only in our work. Since it is unlikely to find a ceil-mounted camera in a non simulated setting, we decided to use only the video sequences captured with the front-facing Kinect. Furthermore, we decided to use only the first 30 sequences that contain a fall, in order to prevent the problems consequent to a highly imbalanced dataset. Of those sequences, we grouped in the "Fall" category the few frames which portray a subject falling or lying down on the ground after a fall and classified the remaining ones as "No Fall". Once performed this undersampling, we obtained a similar number of frames for the two categories and a total of 2995 frames.

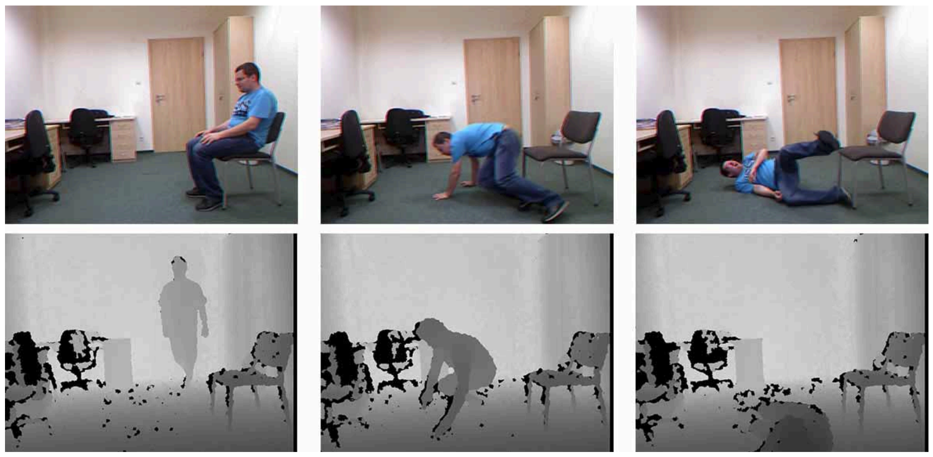

Fig. 1. UR Fall Detection Dataset. Top row: RGB raw images. Bottom row: Kinect's Depth data.

3.2 Multiple Cameras Fall Dataset

Multiple Cameras Fall dataset (Multicam) contains 24 scenarios, each one recorded with 8 IP cameras placed around a studio (Fig.2). The original video files contain a total of 261339 frames, but only half of those have been labeled by the authors. Additionally, due to the time difference required to start the video recording, the cameras have been synchronized by stopping them at the same time. Consequently, we deleted several delay frames. After this initial processing we obtained 134216 frames labeled "No Fall" and 8504 frames labeled "Fall". Finally, when adding the supplementary CNN described in Section 4.2, we chose to get rid of the last two scenarios, as they contain no falls but only confounding events. Indeed, we intended to decrease the unbalance between the number of frames that show a subject falling and the ones which do not.

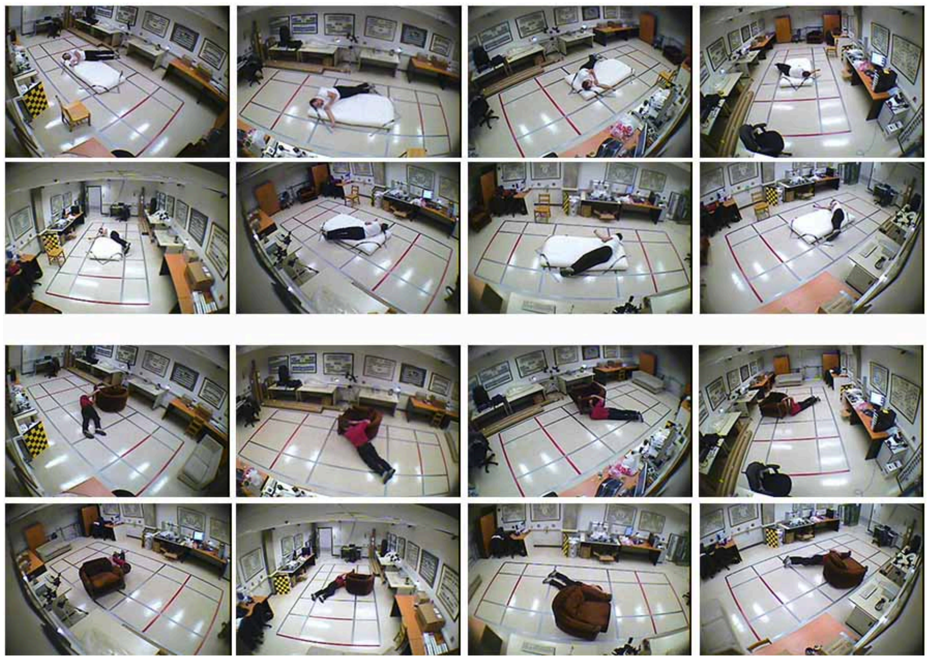

Fig. 2. Multiple Cameras Fall Dataset.

3.3 Combined Dataset

Once performed the described preprocessing on the two datasets, we trained the LSTM with poses which were extracted employing just PoseNet on Multicam, URFall, Balanced Multicam or a combination of these, as described in Section 4.3. After collecting these baseline results, we added a supplementary CNN as explained in Section 4.2. In order to train the CNN on as much data as possible without using the same data for training and testing, we merged URFall and Multicam into one unique dataset and reserved half of it for training the CNN and the other half for poses extraction with PoseNet and the CNN. To decide which half to reserve for training and which for testing/extracting poses, we initially trained our CNN on both halves, one at a time. We will call the first half split A and the second half split B: we initially trained the CNN on split A (80% of which we used for training and 20% for validation). We repeated the same process on split B and compared the results. The CNN obtained higher validation scores on split B, so we discarded the model trained on split A. Split B is then the only set used for training while split A is the only set used for testing/poses extraction.

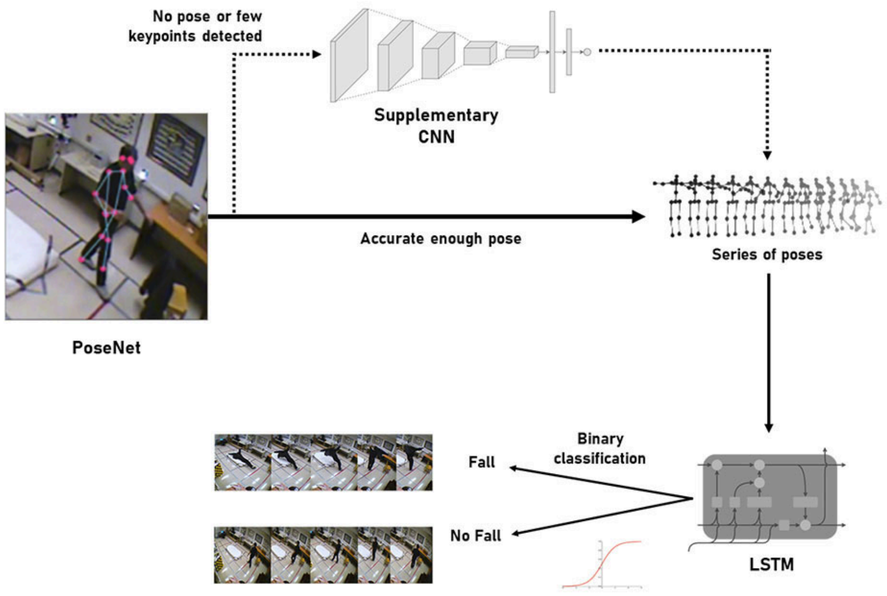

Fig. 3. Architecture overview: PoseNet is initially used to process the current video frame. If the pose is detected with sufficient confidence, it is saved in a time series which will contain 20 poses. If PoseNet fails to detect the pose or detects less than one-third of the 17 keypoints, the supplementary CNN is used as described in Fig.4. Finally, the LSTM is fed with the time series of poses to perform binary classification.

4 *Proposed Method*

We propose a method for detecting falls based on time series analysis of 2D keypoint locations data of a single individual in a controlled environment. In particular, we employ a pose detection model and a supplementary Convolutional Neural Network in order to obtain the 2D position of 17 body key points of the subject shown in each video frame. The keypoint locations data is then split into time-series of 20 poses each and passed as input for a Recurrent Neural Network which classifies such series in two categories: “Fall” and “No Fall” (Fig. 3).

4.1 Pose Detection Model

The first step of our architecture relies on PoseNet, a pose detection model based on the Personlab research conducted by Papandreou et al. [8]. PoseNet can be used to estimate either a single pose or multiple poses in a video. Each pose contains the 2D coordinates of 17 body keypoints (34 numerical total values) and a confidence score for each person detected. Additionally, each keypoint has its own confidence score. In this work we used PoseNet’s algorithm which can estimate multiple poses. Although we typically have only one individual in each video, the multiple poses estimation is more accurate when the subject is partially occluded and it is as fast as the single pose estimation method. Finally, we accept only poses and keypoints with a confidence score greater than 30% and set all other keypoints position values to zero. By running on TensorFlow.js, the detection is performed entirely in the web browser, preserving user privacy as no data needs to be processed by a server. Additionally, no dedicated hardware or complicated system setup is required. PoseNet allows the tuning of its underlying CNN parameters in order to trade less processing speed for more accuracy. PoseNet has been trained using COCO dataset, which is a general-purpose object detection, segmentation, and captioning dataset. For this reason, when testing PoseNet with falls videos, it performed poorly on those frames containing a subject who is lying down or in a similar position. In order to address this problem, we introduced an additional CNN in our architecture, which helps extract information from those frames on which PoseNet struggles the most.

4.2 Supplementary CNN

With the aim of extracting poses from those frames which PoseNet failed to process properly, we implemented a CNN based on a MobileNetV2 model pre-trained on ImageNet dataset and we used it as described in Section 4.3. Among the pre-trained models we tried, MobileNetV2 proved to be the fastest, especially inside the web browser. Indeed, its prediction take an average 10.7 ms latency per image on a Nvidia GTX 1060 GPU, as shown in Table 3 and described in Section 5.2. Additionally, we employed transfer learning to retain the feature extraction capabilities acquired by the model during its training on ImageNet dataset. As a result, we froze the pre-trained weights and swapped the original output layer with two Dense layers with 512 neurons each, plus a final Dense layer with a single neuron and Sigmoid activation to perform binary classification. We trained the model with an 80-20 train-validation split on the second half of our combined dataset described in Section 3.3, and tested it on the first half. As reported in

Table 1, the model we trained peaked at 90% average accuracy on the validation split and obtained a 79% accuracy on the test split. We also tried some other CNN architectures, such as InceptionResNetV2[10] and EfficientNet[11], that currently achieve better top-1 accuracy scores on the ImageNet dataset, but we were limited by overfitting during the training of both. However, when we ran these tests, neither of the two models was compatible with the conversion to TensorFlow.js, as they use some layers which are yet to be implemented in TensorFlow.js. Finally, we tested a VGG16 pre-trained model, freezing the weights of the first three blocks and training the last two. This model reached higher accuracy than the MobileNet V2 (Table 1), but with far worse inference times (Table 3).

Table 1. Training results with different CNN models on validation data: the second half of Multicam and URFall combined. The first half was not used for training the CNN models. It has been reserved for testing and for poses extraction as described in Section 3.3.

Models	InceptionResNetV2	EfficientNetB7	VGG16	MobileNetV2
Accuracy	0.89	0.80	0.96	0.90

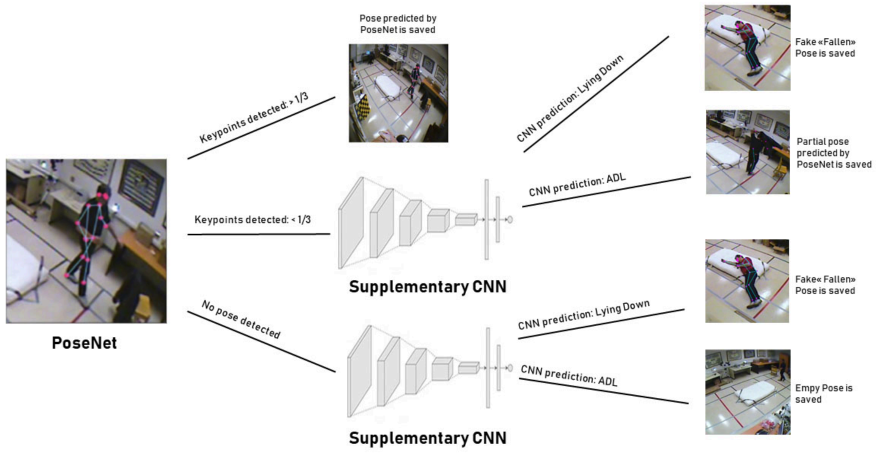
Pose predicted by PoseNet is saved
Keypoints detected: > 1/3
Keypoints detected: < 1/3
No pose detected
PoseNet
Supplementary CNN
Supplementary CNN
CNN prediction: Lying Down
CNN prediction: ADL
CNN prediction: Lying Down
CNN prediction: ADL
Fake «Fallen» Pose is saved
Partial pose predicted by PoseNet is saved
Fake« Fallen» Pose is saved
Empy Pose is saved

Fig. 4. Proposed architecture for extracting one pose from each video frame.
When PoseNet fails to detect the subject or if the pose confidence score is not sufficient, we use the supplementary CNN and save a different pose based on its prediction result.

4.3 Extracting poses

To extract a pose from each frame of the first half of our combined dataset, we developed the following strategy (Fig.4): firstly, we used PoseNet to extract the first detected pose. On one hand, if this pose is empty, meaning that there is no subject in the frame or that PoseNet failed to find it, we run the CNN. If our CNN suggests that a lying subject is present in that frame, we save a dummy pose. This is a particular pose we extracted from a random video frame that contains a lying subject of whom PoseNet was able to detect all the 17 keypoints. As a result, we ensure that, when PoseNet completely fails to detect the lying person, we can always attribute some information to that particular frame, even though it is not the real position of its keypoints. On the other hand, if even the CNN does not detect any subject in the frame, we save an empty pose (an array of 34 zeroes). Another possibility is that PoseNet detects a pose, but the confidence score of more than two-thirds of the keypoints is less than the required 30%. In this case, we use again our CNN: if it predicts the presence of a lying subject, we save the dummy pose, otherwise we save the pose originally detected by PoseNet. Although that particular pose has a little information, for sure it contains more than an empty one.

4.4 Classifying series of poses

The last step of our architecture consists of splitting all the poses we collected with PoseNet and the CNN into series of twenty and using a custom Recurrent Neural Network with LSTM cells to classify such series. We chose to group our poses in time-series of twenty elements because we observed that, in the datasets at our disposal, a fall always spans less than twenty frames. Testing our architecture with time-series of forty poses we obtained worst classification scores. This result is due to the way we label a single series of poses. To condense twenty labels (one for each frame) into a single label (for a series of twenty poses) we determine which label occurs the most in the sequence and choose it. For example, given a sequence of 20 frames, if 13 frames have a "No Fall" label the sequence will be labeled as "No Fall". Once obtained, these labels represent our RNN model's target. Because of this process, splitting poses in time-series of more than twenty, for example forty, introduces the risk of having a series which contains a fall, but gets labeled as "No Fall" as the majority of those forty frames had such label. The Recurrent neural network we used consists of two layers of 34 LSTM cells each. Each LSTM layer uses ReLu activation, while the final Dense layer uses Sigmoid activation to return a binary classification. We include a 20% dropout between each layer and use the Adam loss optimizer.

5 *Experiment*

We tested different variations of our architecture. Initially, we excluded the supplementary CNN to obtain baseline results using only the Recurrent Neural

Table 2. Experimental results obtained training the LSTM model on time-series of poses extracted from different variations of the datasets at our disposal. The last two rows show the classification result after the addition of the MobileNetV2 and the VGG16 CNNs to reinforce the pose extraction phase, while the results showed in the previous rows have been obtained by training the LSTM on series of poses extracted by PoseNet only. Class weighing has been employed when training on Multicam and URFall + Multicam datasets.

	Accuracy	Precision			Recall			F-Score		
Dataset		Avg	No Fall	Fall	Avg	No Fall	Fall	Avg	No Fall	Fall
URFall	0.708	0.700	0.733	0.667	0.693	0.786	0.600	0.695	0.759	0.632
Multicam	0.511	0.518	0.972	0.065	0.602	0.502	0.701	0.390	0.662	0.119
URFall + Multicam	0.753	0.547	0.960	0.134	0.650	0.768	0.533	0.534	0.853	0.214
Balanced Multicam	**0.780**	**0.776**	0.802	0.750	**0.775**	0.811	0.739	**0.776**	0.807	0.745
URFall + Balanced Multicam	0.694	0.694	0.714	0.674	0.694	0.684	0.705	0.694	0.699	0.689
URFall + Multicam (half) with MobileNetV2	**0.818**	**0.809**	0.838	0.781	**0.801**	0.872	0.730	**0.805**	0.855	0.754
URFall + Multicam (half) with VGG16	**0.849**	**0.840**	0.878	0.803	**0.840**	0.878	0.803	**0.840**	0.878	0.803

Network. To achieve every reported result, we used an 80-20 training validation split, using time-series of twenty poses as input for our RNN. We employed Adam as optimizer with learning rate set to 0.0001 and included early stopping with an upper bound of 1000 epochs and patience of 20.

5.1 Experimental Results

We started by training the Recurrent Neural Network on our down-sampled URFall dataset only. As shown in Table 2, we achieved a 73.3% precision score on the "No Fall" category and a 66.7% precision score on the "Fall" category. When we repeated the same training on the original Multicam dataset only, we obtained extremely imbalanced results. In an effort to compensate for the much higher number of frames belonging to the "No Fall" class, we introduced class weights for our two labels with the compute_class_weight method provided by the sklearn.utils library [2].Nevertheless, we obtained poor results on the "Fall" class, with a 0.65% precision score and a 70.1% recall, meaning that our model learned to guess mostly the same class, but being the "Fall" events so few, it still detected some of them. Even combining the two datasets, we obtained only minimal improvements in precision results, with the "Fall" category reaching a 1.34% precision. The "Fall" recall result got worse, with a 53.3% rate, while the

[2] Scikit Learn: compute_class_weights. `https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html`. Accessed November 17th, 2020.

"No Fall" recall improved and reached 76.8% from the 50.2% obtained with imbalanced Multicam only. Being Multicam so much bigger than our down-sampled URFall dataset, the addition of URFall's few frames proved to be practically irrelevant. After these initial results showed that Multicam's high imbalance was negatively affecting our model performance, we decided to perform an undersample of such dataset, in an attempt to decrease the gap between our two categories. We manually analyzed every Multicam's video and chose a new first and last frame for each one, to make sure that the number of frames of the fall event itself was similar to the sum of pre-fall and post-fall frames. After this undersampling, Multicam counted 15840 frames, 7640 of which were labeled as "No fall" and 8200 as "Fall". Training our RNN on the newly balanced Multicam dataset only, we obtained much more balanced and generally better results, with a 80.2% and a 75% precision score for the "No Fall" and "Fall" class respectively. After combining URFall with Balanced Multicam and not achieving better results, we finally tested our complete architecture with the addition of the MobileNetV2 model in the poses extraction phase. This model enabled us to extract features from most of the frames that PoseNet would partially or completely fail to process. Given this new chance, we rolled back to using the original Multicam dataset, of which we discarded only the last two chutes because they do not contain fall events. We combined the original Multicam with URFall and obtained our best overall scores, including a 81.8% average accuracy rate. The baseline results we obtained show that performing undersamples on imbalanced datasets, which are frequent when considering fall ones, can lead to more homogeneous results.

5.2 Time Cost Analysis

Our aim is to make possible detecting a fall in real time, virtually on any machine and with minimal additional setup requirements. These requirements are a fundamental prerequisite to allow the use of a fall detection system in consumer applications, such as in elderly care. Indeed, we find it important to make fall detection technologies broadly available to elderly people in the first place, as they are not only the subject who are most prone to such type of accidents, but also those who risk suffering the most serious consequences. As shown in Table 3, PoseNet is perfectly capable to estimate poses in real time. Especially when using the MobileNetV1 backbone, it runs at well above 24 fps even on a machine with integrated graphics. We can say the same for our custom LSTM network. We conducted further tests running PoseNet on a more capable machine. The results show that even when using the more demanding ResNet50 backbone, and maxing out its settings to achieve the best possible pose estimation accuracy, PoseNet is able to run in real-time inside a web browser. When adding the supplementary CNN to reinforce the poses extraction we are able to obtain better classification scores. Furthermore, the MobileNetV2 model we chose and trained, which is the evolution of the MobileNetV1 architecture used at PoseNet's core, performed predictions with an average 19ms per frame latency on the Macbook. When tested on a Desktop PC with discrete graphics, it runs at nearly double the speed, with an average 10.7ms latency per frame as shown in Table 3. As reported in Section 4.2, we tested other CNN architectures, among which only the VGG16 is currently compatible with the conversion to TensorFlow.js. Even if this models performs better in terms of accuracy (Table 1), its inference times are much higher. On the Macbook, the VGG16 takes almost 38 times more than the MobileNetV2 to process a single image (Table 3). On the Deksktop PC the results are significantly better, but the average 12 frames per second inference speed is still not feasible for real-time processing. Additionally, we use the supplementary CNN only on those frames which PoseNet fails to estimate correctly, hence choosing the MobileNetV2 over a better performing model, such as the VGG16, resulted in a 3% drop in LSTM classification accuracy. In conclusion, models which have the same, if not better, representational capacity as the VGG16 and run as fast as MobileNetV2 already exist for other platforms. The recently announced EfficientNet Lite [3] is the perfect example as it matches InceptionV4 Top1 80.4% accuracy on the ImageNet dataset with a 30ms per frame latency on a smartphone CPU. With this, or similar CNNs, coming to TensorFlow.js it will be possible to run our fall detection architecture in real time, achieving even better classification results, even on low-powered machines.

Table 3. Average inference times for a single 200x200 image reported in milliseconds and frames per second on a base model 2019 Macbook pro with Intel Core i5 8th gen CPU, integrated GPU and a Desktop PC equipped with Intel Core i7 8700k CPU, Nvidia GTX 1060 3GB GPU. PoseNet(MobileNetV1) backbone settings are: outputStride set to 16, multiplier set to 0.5. PoseNet with ResNet50 backbone settings are: outputStride set to 32, quantBytes set to 1. All tests were conducted using Microsoft Edge browser version 86.0.622.69.

	PoseNet (MobileNetV1)		PoseNet (ResNet50)		MobileNetV2		VGG16		LSTM	
	ms	fps	ms	fps	ms	fps	ms	fps	ms	fps
Macbook	19.6	51.0	83.3	12.0	20.6	47.6	380.6	2.6	38.5	26.3
Desktop PC	10.4	96.2	27.3	36.6	10.7	93.5	79.5	12.6	17.2	58.1

5.3 Comparing with state of the art

Training our model on URFall only led to obtaining 70.8% accuracy, while Jeong et. al [5], who used poses to train an LSTM as we did, obtained 61.5% on the same dataset. By combining URFall with SDUFall (a fall dataset that is currently

[3] EfficientNet Lite. `https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet/lite`. Accessed November 17th, 2020.

unavailable online) and implementing other features extraction methods, such as the human center line coordinate (HCLC), Jeong et al. achieved their best score of 99.2% accuracy. However, they do not report a time cost analysis. Comparing with Feng et al. [3] attention guided LSTM model, we can see that they obtain higher results on both URFall and Multicam datasets with a 91.4% recall on the former and 91.6% sensitivity on the latter (accuracy score have not been reported), but they also do not report a time cost analysis. Pourazad et al. [9] trained their LRCN model on their newly proposed dataset, obtaining an 87% accuracy score. Their model can process videos in a short time window, but they do not report a more precise processing speed metric. Xu et al. [12] obtain an even higher 91.7% accuracy rate by training an Inception-ResNet-v2 [10] model on a combination of URFall, Multicam, NTU RGB+D Action data set and fall videos found online. However, they reported that their model takes around 45 seconds to make a prediction for a single image without hardware acceleration, making it not feasible for a real-time use. Han et Al. [4] obtained accuracy scores as high as 98.25%, but they used a self collected dataset made of only 3628 pictures, 50% of those used for training and the other 50% for testing. They report that their newly introduced MobileVGG model took 2.96 seconds to make inferences on the whole test set (1314 images 224x224 resolution) and 0.75 seconds when testing with 100x100 image resolution. Finally, Na et Al. [7] achieved an impressive 100% accuracy score on Multicam, and proved that their implementation is effective even if tested on activity classification benchmarks like UCF11 and HMDB-51. They also presented a time-cost analysis, showing that their 3D CNN extracted the UCF11 dataset's features at 60.2 fps. However, they used a high-end Nvidia Tesla K40 GPU and it is unclear which is the time cost of their LSTM.

Table 4. Comparison with other state of art pieces of research. For each study only the best result and the corresponding inference times, when reported, are shown.

	Pourazad et al.	Xu et al.	Han et Al.	Jeong et. al	Na et Al.	Ours
Dataset	Self collected	URFall + Multicam + NTU + Youtube videos	Self collected (3628 images)	URFall + SDU Fall	Multicam	URFall + Multicam (Half)
Accuracy	87%	91.7%	98.3%	99.2%	**100%**	81.8%
Latency	-	45 s (CNN only)	2.3 ms (CNN only)	-	16.7 ms (CNN only)	**37ms (full architecture)**
GPU	Nvidia Tesla K80	-	Nvidia Quadro P4000	Nvidia GTX 1080	Nvidia Tesla K40	Nvidia GTX 1060 3GB
Browser	No	No	No	No	No	**Yes**

6 Conclusions

In this work we proposed a fall detection system based on pose data obtained from RGB videos that can run completely inside the web browser, preserving users' privacy as no data is sent to a server to be processed.

The proposed architecture is modular because it is possible to exploit a supplementary MobileNetV2 CNN during the poses extraction phase to achieve better fall classification results. At the same time, our system can run in real-time even on low powered machines, while preserving users' privacy as the processing is entirely done locally inside the web browser.

Our architecture represents an attempt to highlight the potential of tools like PoseNet and TensorFlow.js and to shift fall detection researchers' focus towards the development of software that can be actually used by interested users. Furthermore, if user privacy is respected, we could earn the trust necessary to achieve important results, such as obtaining user permission to collect real falls videos. The proposed system shows great potential for specific users, including elderly people, who could finally benefit from an effective and low or zero cost solution. Indeed, running inside the browser, our system could be distributed as a web app and used with an IP camera or a cheap webcam, allowing any individual to detect falls in a home environment.

6.1 Future Developments

As discussed in Section 5.2, the first and most immediate improvement to our system would come from including a CNN architecture that is both highly performing and efficient enough to run in real time on most computers. While waiting that models like EfficientNet [11] are ported to tensorFlow.js, we are planning to develop a custom CNN, specifically designed for our purposes, for fast inference time in the browser. The pose extraction process itself could be improved by addressing the problem of the not accurately detected keypoints. We could interpolate the values of a previous and following keypoint to determine reasonable coordinates for the keypoints that PoseNet fails to detect. This would mean having more meaningful data fed to the LSTM during its training process, probably resulting in higher classification results. Finally, considering the fast inference times of the MobileNetV2 model we trained, we could ditch PoseNet completely and use only the CNN to extract self learned features that could be fed directly to the LSTM to perform classification. This could lead to better robustness in real use cases thus making our model capable of detecting falls even in situations with more than one individual or with challenging environmental conditions.

References

1. Auvinet, E., Rougier, C., Meunier, J., St-Arnaud, A., Rousseau, J.: Multiple cameras fall dataset. DIRO-Université de Montréal, Tech. Rep **1350** (2010)

2. Cameiro, S.A., da Silva, G.P., Leite, G.V., Moreno, R., Guimarães, S.J.F., Pedrini, H.: Multi-stream deep convolutional network using high-level features applied to fall detection in video sequences. In: 2019 International Conference on Systems, Signals and Image Processing (IWSSIP). pp. 293–298. IEEE (2019)
3. Feng, Q., Gao, C., Wang, L., Zhao, Y., Song, T., Li, Q.: Spatio-temporal fall event detection in complex scenes using attention guided lstm. Pattern Recognition Letters **130**, 242–249 (2020)
4. Han, Q., Zhao, H., Min, W., Cui, H., Zhou, X., Zuo, K., Liu, R.: A two-stream approach to fall detection with mobilevgg. IEEE Access **8**, 17556–17566 (2020)
5. Jeong, S., Kang, S., Chun, I.: Human-skeleton based fall-detection method using lstm for manufacturing industries. In: 2019 34th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC). pp. 1–4. IEEE (2019)
6. Kwolek, B., Kepski, M.: Human fall detection on embedded platform using depth maps and wireless accelerometer. Computer methods and programs in biomedicine **117**(3), 489–501 (2014)
7. Lu, N., Wu, Y., Feng, L., Song, J.: Deep learning for fall detection: Three-dimensional cnn combined with lstm on video kinematic data. IEEE journal of biomedical and health informatics **23**(1), 314–323 (2018)
8. Papandreou, G., Zhu, T., Chen, L.C., Gidaris, S., Tompson, J., Murphy, K.: Personlab: Person pose estimation and instance segmentation with a bottom-up, part-based, geometric embedding model. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 269–286 (2018)
9. Pourazad, M.T., Shojaei-Hashemi, A., Nasiopoulos, P., Azimi, M., Mak, M., Grace, J., Jung, D., Bains, T.: A non-intrusive deep learning based fall detection scheme using video cameras. In: 2020 International Conference on Information Networking (ICOIN). pp. 443–446. IEEE (2020)
10. Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.A.: Inception-v4, inception-resnet and the impact of residual connections on learning. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence. pp. 4278–4284 (2017)
11. Tan, M., Le, Q.: Efficientnet: Rethinking model scaling for convolutional neural networks. In: International Conference on Machine Learning. pp. 6105–6114 (2019)
12. Xu, Q., Huang, G., Yu, M., Guo, Y.: Fall prediction based on key points of human bones. Physica A: Statistical Mechanics and its Applications **540**, 123205 (2020)