A Convolutional Neural Network Approach Utilizing MobileNetV2 with Motion History Images for Fall Rate Detection, Identification, and Analysis

Jonah Falk, Samuel Pete, Normandy Rivver, Niko Robbins, and Jacob Schmoll
Department of Computer Science
Missouri State University
Springfield, MO, USA
jaf12@live.missouristate.edu, sbp0702@live.missouristate.edu, bryson11@live.missouristate.edu,
njr926@live.missouristate.edu, schmoll99@live.missouristate.edu

***Abstract*— The increasing aging population will require substantial care in the coming years. This care can be divided into three options: aging in place, nursing homes, or living with one's family. At minimum, one will require increased supervision at some point as one's health deteriorates. Most households in the US are headed by two wage earners, meaning both individuals go to work full-time. As the demands of the current economic conditions, which requires two wage earners, and the needs of older adults collide, new methods of extending aging in place or with the family will be needed. One way of extending aging in place or aging with family is the development of fall detection to notify caregivers of a potential fall situation. In this current paper, we examine the ability of MobileNetV2 to detect fall situations using Motion History Images (MHI) across both fall situations and activities of daily living (ADL). One of the main advantages in MobileNetV2, as compared to MobileNetV1, is that it requires 30% less parameters, uses 2x fewer operations, and is 30-40% more accurate.**

Keywords*— *Convolutional Neural Network, Computer Vision, Fall Detection, OpenCV, MobileNet, Motion History Images.

I. Introduction

Currently the fastest growing segment of the United States population is those over 65. This increase has resulted in the aging population exploding by 15% in just 15 years when compared to previous numbers [1]. One of the main goals during older adulthood is to provide the individual with the autonomy to age in place for as long as possible.

Several devices have been developed to extend aging in place for older adults such as LifeAlert®, MedMinder©, and fall detection wearables to name a few. However, these devices, such as fall detection wearables, are often limited in effectiveness or not effective at all if the older adult forgets to put it on. With these limitations in mind, better methods of fall detection are required to ensure older adults can age in place safely or can be cared for in a nursing home environment safely.

Since falls are often one of the most common reasons for serious health issues in older adults, ensuring quick notification when such an event occurs will no doubt save lives [11]. Due to the fact that current fall detection systems are not sufficient, and the US population is increasingly getting older, better fall detection methods are a must to ensure that levels of care meet the requirements for individual autonomy. In an effort to develop a better system of fall detection, we propose a convolutional neural network using MobileNetV2 trained on MHI.

MHI offer a unique approach to fall detection because pixel density is connected to movement and shape information through time. With this in mind, our approach at time n takes 32 frames, and at time n+1 takes 16 frames plus an appended 16 frames from the previous frame, corresponding to the last 16 frames in frame n+1-1. The sampling strategy is designed to provide a condensed first 16 frames and a spaced 16 last frames in an attempt to better approximate movements in humans.

The rest of this paper is organized as follows: Literature review regarding MHI fall detection methods in section 2. In section 3, the proposed method and technical details are explained in greater depth. Performance evaluation of experimental results are presented in section 4 and we conclude this paper in section 5.

II. Procedure for Paper Submission

Several researchers have implemented MHI into a project implementing fall detection. These implementations have varied in a few key ways regarding fall detection. First, implementations have varied in the selection of neural networks used to detect falls. For example, Xi Cai, Xinyue Liu, Suyuan Li, and Guang Han used a VGG16 network in combination with color-coded MHI to show better results than a VGG16 network in combination with MHI [4].

Jantima Thummala and Suree Pumrin used no neural network or machine learning approach and instead focused on detecting falls through MHI and shape deformation [5].

Likewise, Miao Yu, Syed Mohsen Naqvi, and Jonathon Chambers opted for a non-neural network approach by focusing instead on MHI and codebook background subtraction combined along with head tracking as a motion parameter to detect falls [3].

Truls Haraldsson used a modified MobileNet along with background subtraction [2]. Each project obtained satisfactory results [2], [3], [4], [5], [6].

Second, while all implemented MHI as an input to detect falls, what was used in combination with MHI varied. Four out of five papers relied on background subtraction as a way to focus in on the object of interest. One paper relied on MHI, background subtraction, and shape deformation [5]. Shape deformation aims to measure the change in height between particular movements as a way to distinguish between normal and abnormal movements along with the targets current speed.

The goal is to highlight abnormal shapes as being more indicative of a fall situation rather than normal movement shapes.

Another project, [4], as a way to increase the visual quality, allowed for more spatiotemporal effect and image information. Their focus when using MHI is the spatiotemporal information, of which an algorithm can provide, and as such many different options exist that can apply the technique to fall detection.

III. Proposed Solution

As discussed in the beginning of the paper, our team approach utilizes a MobileNetV2 trained on MHI without background subtraction. MobileNetV2 is far superior to MobileNetV1 as indicated below:

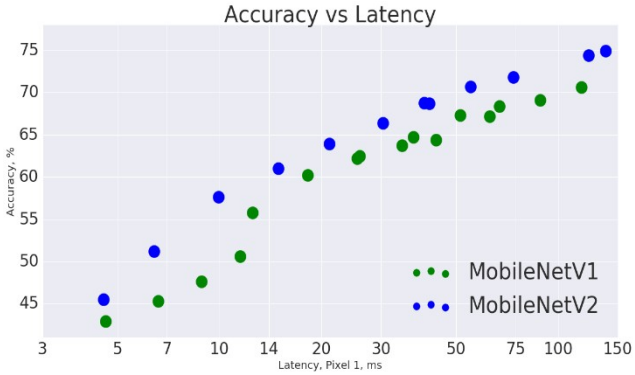

Figure 1. MobileNet Accuracy and Latency Comparison

TABLE I. MOBILENET PERFORMANCE UTILIZING SSDLITE

Model	Params	Multiply-Adds	mAP	Mobile CPU
MobileNetV1 + SSDLite	5.1M	1.3B	22.2%	270ms
MobileNetV2 + SSDLite	4.3M	0.8B	22.1%	200ms

TABLE II. MOBILENET PERFORMANCE UTILIZING DEEPLABV3

Model	Params	Multiply-Adds	mIOU
MobileNetV1 + DeepLabV3	11.15M	14.25B	75.29%
MobileNetV2 + DeepLabV3	2.11M	2.75B	75.32%

MobileNetV2 is a lightweight neural network and has improvement over MobileNetV1 by using depthwise separable convolutional, of which can be used as building blocks [7]. Our MobileNetV2 consisted of 19 layers plus an input layer. The MobileNetV2 model is available in both the Keras and PyTorch libraries and has the option to be pretrained.

Initially the team attempted to use the Keras implementation, but due to installation issues which resulted in the IDE Spyder becoming uninstalled, we opted for the PyTorch model provided by Nithiroj Tripatarasit. The model was trained using MHI from a fall detection dataset. An MHI is a way to represent spatiotemporal information by allocating heavier pixel densities to movement which has occurred earliest, and also movement, which is deemed more intense (i.e., stronger motion movement) [10].

One option we had considered was using Lucas-Kanade optical flow algorithm, but due to increased time complexity associated with the algorithm, we decided to rely on MHI. Another tradeoff between Lucas-Kanade optical flow algorithm and MHI algorithm is that MHI generally provides less information, which may impact accuracy [4].

Our components for this proposed solution rely heavily on [6]. The implementation of the MHI algorithm followed the pseudocode as seen in the following:

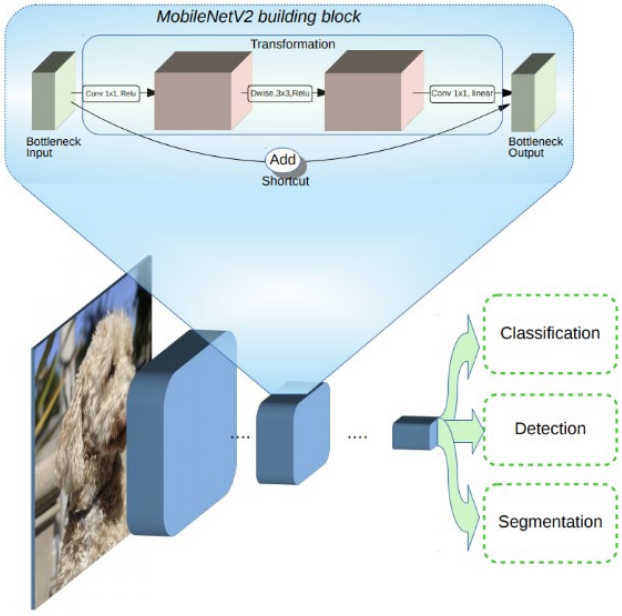

Figure 2. MobileNetV2's Usage of Depthwise Separable Convolution

Figure 3. Output of Motion History Images

One difference between our implementation and Nithiroj Tripatarasit's is the usage of an ONNX model whereas Nithiroj aims to implement it via an AWS cloud using a

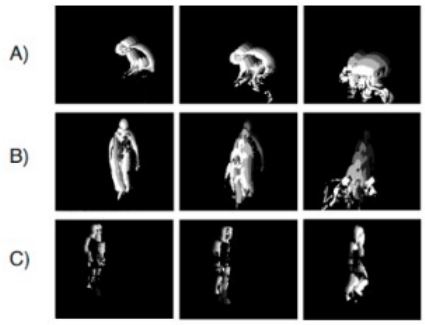

Figure 8: Shows generated MHIs. In A) a person is falling from a chair, B) a person is falling from standing position, C) a person walking around.

Input : A MHI mhi, the current image I_c, the previous image I_p, a duration δ, and a threshold ξ.
Output: An updated MHI mhi^*.
$I_d \leftarrow \text{DiffRGB}(I_c, I_p)$;
$I_g \leftarrow \text{ConvertToGrayScale}(I_d)$;
$I_t \leftarrow \text{BinaryThreshold}(I_g, \xi)$;
$mhi^* \leftarrow \text{Decay}(mhi, \delta)$;
$mhi^* \leftarrow \text{Add}(mhi^*, I_t)$;

PyTorch model. Since we use OpenCV to run the program, the original PyTorch model had to be converted into ONNX format as OpenCV does not support PyTorch models currently.

When exporting to an ONNX model, one must first load the original PyTorch model, then create a random PyTorch tensor using the exact specifications in the original model. For example, the original model requires a batch size of 32, color scale of 1 (grayscale), and dims (height and width) of 128x128 for a final shape of (32,1,128,128).

While exporting the model is relatively easy, getting the model loading into OpenCV was not. We will not go into details in this paper regarding exact implementation details, but the process itself took five days as a number of dependency issues exist, such as some dependencies being only available by PIP and not Conda, alongside vague implementation instructions. Another issue was the lack of exact version numbers required by both ONNX and OpenCV regarding dependencies and the OpenCV version number needed.

There are several components in this project, with the most obvious being the MobileNetV2 model. The initial frame input has a shape of (1, 256, 224, 3) which needs to be transformed into a shape the model expects which is (32,3,256,224). Several transformations are used to create the necessary input shape for the model, but one will notice a disconnect between what we exported as the shape for the ONNX model and what grabbing one frame shape is in OpenCV. Rather unconventionally, PyTorch and OpenCV place the RGB, height, and weight in different spots leading to a required transformation of the data into the necessary shape for inference.

This leads us to our second component, which is the transformer. This component focuses on taking an initial OpenCV frame and converting the frame to be used in the MHI algorithm. Of note here is that we cannot simply convert the frame to grayscale as the OpenCV ONNX model requires a 3-channel RGB frame. For the system to convert to grayscale, we assign each 3-channel RGB values to a temporary frame from the original frame into grayscale, thereby ending up with a grayscale frame, which is 3 channels.

Our MHI component selects the first frame and resizes it based on the dim requirements. When resizing the image, we apply the algorithm INTER_AREA, which focuses on resampling the pixel using area relation. In terms of speed, an INTER_NEAREST for interpolation is supposedly faster, but neither results nor latency seemed to be impacted from using either one.

The second time through the program we again apply the same method to a new frame assigned its own value. Using absdiff, we take the difference between the two frame arrays, which gives us a difference output array between the two. The difference array is then compared in a Boolean against our hyperparameter threshold. The comparison between the diff and threshold is such that a diff greater than or equal to the threshold gives a value of 1 assigned to the variable binary, and a difference less than our threshold assigns a 0 to our binary. The result is if the object is moving and above the threshold value, detect it, and assign the updated difference in pixel intensity and, if not, the object is currently not moving.

Finally, the MHI is calculated using the binary with the maximum selected between the previous frame and the previous frame minus our decay rate. In essence, this last section is resulting in the decay of pixel intensity across time.

Our last component finalizes the transformation so the input can be accepted by the model. As previously noted, the PyTorch shape corresponds to (1, 256, 224, 3), while the model expects (32,3,256,224). First, we resize our MHI using INTER_AREA. Next, we expand the dimensions at axis 0, which gives us an MHI frame shape of (1, 256, 224, 3). Then, we utilizes PyTorch to permute the image into the correct shape of (1, 3, 256, 224). The last transformation process is taking 32 frames and appending them to a NumPy array. The component accomplishes this by appending each frame in the 32 to a batch and uses the NumPy concatenate at axis zero, which finally gives us a shape of (32,3,256,224)—the required model input.

The last component alters the model output into a more human-readable form by first applying softmax, as the values without it are in logit form. After softmax is applied, we now have 32 probabilities corresponding to fall or not-fall in a given frame. The component loops through each probability and compares the probability of fall to the probability of not-fall. If at any point fall is greater or equal to not-fall, the system provides a notification that a fall is detected.

One question our team had was whether or not movement which corresponded to the beginning in the current frame but ending in the next frame would cause issues of accuracy. As such, we approached a continuous bleed-over effect from the current frame into the next frame. This was accomplished by taking all 32 frames during the first iteration. Then on all subsequent iterations, we append the last 16 frames from the current frame onto the beginning of the next frame.

To ensure an equal distribution between the beginning, middle, and end in regard to the 32 frames we split, around 10 frames in each section, the system processes 32 frames, but only takes every second frame. Together, 16 frames in the current frame and 16 frames from the previous, we end up with 32 frames connecting one frame to the next frame through time. Our approach provides an examination of frames which condenses the current frame to 0-15 in terms of time by selecting the last 16 frames from the previous frame and for 16-31 spaced by taking every second frame of the current frame. This sampling corresponds to what may be more typical in human movement, meaning at various times, movements are condensed, such as simply sitting, standing, or lying down, and at other times, movements are spaced, such as when walking, running, or falling.

IV. Performance Evaluation

As noted earlier, our project utilized MobileNetV2 built using PyTorch which was trained using a fall detection dataset [8]. The dataset consisted of various locations, such as Office, Lecture Room, Coffee Room, and Home. The model training utilized a 70%, 20%, 10% train, validate, test method. In the testing phase, a total of 1458 samples were viewed by the model with an accuracy of 95% (1383/1458).

After the model was loaded into Python using OpenCV, we took the dataset found at [9] with a total of 60 fall videos and 40 ADL videos. All samples were used on both models. First,

all 60 fall videos then all 40 ADL were tested on the condensed-spaced model. Afterward, all 60 fall videos and then all 40 ADL were processed by the regular model. Results were tallied and are presented below for both the regular model and the condensed-spaced model:

TABLE III. CONDENSED-SPACE MODEL ACCURACY FOR FALLS RESULTS

Condensed-Spaced Model	
Accuracy for Falls	
Fall Cam 0	15/30 = 50%
Fall Cam 1	6/30 = 20%
Overall	21/60 = 35%

TABLE IV. CONDENSED-SPACE MODEL ACCURACY FOR ADL RESULTS

Condensed-Spaced Model
Accuracy for ADL
19/40 = 47.5%

TABLE V. OVERALL CONDENSED-SPACE MODEL PERFORMANCE

Condensed-Spaced Model	
Performance Evaluation Overall	
Precision	40/61 = 65%
Recall	40/89 = 45%
F1-Score	0.59/1.1 = 53%

As shown above, the condensed-spaced model had an overall accuracy of 35% regarding fall detection. The difference between a higher and lower accuracy rating is likely due to a few key reasons. First, the threshold value can be set within a range, 0.1-0.5, which provides different results. For example, a threshold value of 0.4 may correctly identify a fall in one of the fall videos whereas 0.1 may not, or vice versa. In fact, several papers which used the MHI process usually mention this threshold range issue. Second, as previously stated, the model was trained on a very limited number of videos and as such likely lacks generalizability. This is an easy fix of simply requiring more model training, but due to time constraints and the time it takes to train the model on a laptop, we had to reduce training time overall. In terms of detecting ADL, or rather, not detecting falls when ADL are occurring, the model performed at 47.5% overall. A few reasons for this score are the ADL videos progress from regular brightness, to dim, to complete darkness with the person under monitoring wearing a light on their body. The model performed much better in environments were lighting was not adequate. This is likely due to the model being unable to properly adjust to the lighting changes. As such, the true score of false positive ADL detection is probably higher. These two elements, fall detection in fall videos and fall detection in ADL, bring up the core issue with MHI. This core issue revolves around the idea of making the model too strict where it misses falls it otherwise would have detected with a less strict threshold, or making the model too lenient so that normal movements become incorrectly identified as a fall situation. Our team spent several days changing hyperparameters around, but this issue of strictness versus lenience is not something which can be overcome simply through hyperparameter selection alone. More training is required, but it should be of no surprise that even with a well-trained model, one is at the mercy of strictness versus leniency in threshold selection. This threshold selection problem is why most teams opted to use MHI in combination with something else as MHI alone, even when sent into a neural network, will eventually hit the threshold selection problem. The performance measures of our model are a bit better than our accuracy, but are not even close to be useful for actual deployment into a real-world environment. For the regular model, our results show:

TABLE VI. REGULAR MODEL ACCURACY FOR FALLS RESULTS

Regular Model	
Accuracy for Falls	
Fall Cam 0	16/30 = 53%
Fall Cam 1	9/30 = 30%
Overall	25/60 = 42%

TABLE VII. REGULAR MODEL ACCURACY FOR ADL RESULTS

Regular Model
Accuracy for ADL
18/40 = 45%

TABLE VIII. OVERALL REGULAR MODEL PERFORMANCE

Regular Model	
Performance Evaluation Overall	
Precision	43/65 = 66%
Recall	43/78 = 55%
F1-Score	0.73/1.2 = 61%

As shown above, the regular model outperforms the condensed-spaced model in terms of accuracy by providing an overall accuracy rating of 42%. The suggestion here is there is no issue with a movement starting in one batch, but rather ending in another batch. This makes sense in some respects, as the model weights are adjusted each time an input is sent in. So even though a movement may be disconnected, meaning it exists in two different batches, the model still processes the situation as a continuous feed of updated weights and not as a processing of each unique batch frame, of which produces a set of its own unique weights.

Accuracy regarding ADL show the regular model performing a bit worse than the condensed-spaced model at 45%, but neither model performs well. The performance measures other than accuracy lead to similar conclusions, which was said about the condensed-spaced model in that the model is nowhere near good enough for deployment in real-

life settings.

V. Procedure for Paper Submission

Overall, both our models missed the mark in terms of what we set for accuracy expectations. With that said, this lack of accuracy is most likely attributed to the fact of reduced training received by the models. In the future, it would make most sense to train the models on more fall situations in more environments, and for more ADL as well, of which mimic fall situations. Another important aspect is combining MHI with the models and some other detection process as described in the papers presented here. Since MHI is likely to have a threshold selection issue, one could add another technique, of which could reduce the impact of the threshold selection problem and would be beneficial overall.

The condensed-spaced sampling method used seemed to provide little benefit, though the sampling method may be better suited to optical flow images, which allow information about the speed change for moment n to n+1. Future research would be aimed at looking into the two areas discussed above.

References

[1] Mahidol University Institute for Population and Social Research. (2013). “Population Ageing,” [Online]. Available: http://www.ipsr.mahidol.ac.th/ipsrbeta/th/ResearchClusters.aspx?ArticleId=46. [Accessed: Mar. 13, 2019].

[2] T. Haraldsson, “Real-time Vision-based Fall Detection,” pp. 1–38, 2018.

[3] M. Yu, S. M. Naqvi, and J. Chambers, “Fall detection in the elderly by head tracking,” 2009 IEEE/SP 15th Workshop on Statistical Signal Processing, 2009.

[4] X. Cai, X. Liu, S. Li, and G. Han, “Fall Detection Based on Colorization Coded MHI Combining with Convolutional Neural Network,” 2019 IEEE 19th International Conference on Communication Technology (ICCT), 2019.

[5] J. Thummala and S. Pumrin, “Fall Detection using Motion History Image and Shape Deformation,” 2020 8th International Electrical Engineering Congress (iEECON), 2020.

[6] N. Tripatarasit, “Fall Detection with PyTorch,” Medium, 01-May-2020. [Online]. Available: https://medium.com/diving-in-deep/fall-detection-with-pytorch-b4f19be71e80. [Accessed: 27-Jul-2020].

[7] “MobileNetV2: The Next Generation of On-Device Computer Vision Networks,” Google AI Blog, 03-Apr-2018. [Online]. Available: https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html. [Accessed: 27-Jul-2020].

[8] “Laboratoire Electronique, Informatique et Image UMR CNRS 6306,” Le2i. [Online]. Available: http://le2i.cnrs.fr/Fall-detection-Dataset?lang=fr. [Accessed: 27-Jul-2020].

[9] UR Fall Detection Dataset. [Online]. Available: http://fenix.univ.rzeszow.pl/~mkepski/ds/uf.html. [Accessed: 27-Jul-2020].

[10] Bobick A F , Davis J W . The recognition of human movement using temporal templates[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2001, 23(3):0-267.

[11] Yang, Lei, Ren, et al. Sensors, Vol. 15, Pages 23004-23019: New Fast Fall Detection Method Based on Spatio-Temporal Context Tracking of Head by Using Depth Images[J]. 2015.