The Detection for Falling

Matthew Gan

March 24, 2021

Abstract

This project is currently focusing on detecting falling. The main idea of this project starts with obtaining footages of people falling along with non-falling activities. To analyze the footage and extract features, a tool named "OpenPose" developed by CMU is applied to the footages to obtain the positions of the various parts of the body in each frame. These data will then be fed into a CNN model with labels that indicate if the person is falling in a frame. Unfortunately, the result shows a low accuracy, with extremely high False Positive Rate. The next step would be analyzing the results and try to find the reasons behind the low accuracy.

1. Topic:

The Topic of this project is detecting the action of **falling** inside a video

2. Motivation

The eventual goal and hope of this project is to allow a smart home monitoring system to detect the falling of a person who may have underlying issue. For example, if a person with underlying medical issue suddenly has a stroke and falls onto the ground, the system would be able to immediately alert surrounding people and send messages to medical staff.

3. Related Works:

Given the motivation described above, various studies have already been conducted to try to solve this problem. In 2010, Lai and Huang[1] utilize various sensors, such as neck sensor, waist sensor, and thigh sensor all together to perform a body posture analysis, with the adaptive adjustment model to eventually determine if the person's posture is during a falling motion. Even as early as 2002, a group of researchers in Spain[2] already developed a distributed intelligent architecture for falling detection, with a similar approach to the previously mentioned study that utilizes body sensors and feeds the data into an algorithm. However, the approach we are taking here focuses on computer vision, coupled with feature analysis and extraction.

4. Proposed Model:

The model I used to train here include three RNN models and one CNN model. The RNN models are GRU , LSTM and biLSTM, with the latest one yielding the best accuracy. Here are the summary of the model architecture:

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_2 (Bidirection (None, 1, 64)             24576
_________________________________________________________________
bidirectional_3 (Bidirection (None, 1, 32)             10368
_________________________________________________________________
dense_2 (Dense)              (None, 1, 1)              33
=================================================================
Total params: 34,977
Trainable params: 34,977
Non-trainable params: 0
_________________________________________________________________
```

Figure 1: Model Summary of biLSTM

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 1, 32)             12288
_________________________________________________________________
lstm_3 (LSTM)                (None, 16)                3136
_________________________________________________________________
dense (Dense)                (None, 1)                 17
=================================================================
Total params: 15,441
Trainable params: 15,441
Non-trainable params: 0
_________________________________________________________________
```

Figure 2: Model Summary of LSTM

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 3, 21, 64)         640
_________________________________________________________________
batch_normalization_5 (Batch (None, 3, 21, 64)         12
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 3, 21, 64)         36928
_________________________________________________________________
batch_normalization_6 (Batch (None, 3, 21, 64)         12
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 10, 64)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 640)               0
_________________________________________________________________
dense_7 (Dense)              (None, 512)               328192
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 513
=================================================================
Total params: 366,297
Trainable params: 366,285
Non-trainable params: 12
_________________________________________________________________
```

Figure 3: Model Summary of CNN

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_7 (Conv1D)            (None, 1, 32)             2048
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 1, 32)             0
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 1, 32)             1056
_________________________________________________________________
gru_1 (GRU)                  (None, 32)                6336
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 9,473
Trainable params: 9,473
Non-trainable params: 0
```

Figure 4: Model Summary of GRU

5. Dataset

The dataset used for this first attempt include a dataset named “UR Fall Detection Dataset” that comes from the University of Rzeszow and another Dataset called “Multiple cameras fall dataset”. The UR Fall dataset contains 70 videos, with 30 falls and 40 activities of daily living and each video lasts about 5 minutes. This dataset contains video footages of these activities with two cameras, with one horizontal with the ground and one on the top. However, at this point we will only be using camera 0 that has a horizontal angle view. Link for this dataset: http://fenix.univ.rzeszow.pl/~mkepski/ds/uf.html .

The Multiple Cameras fall dataset contains 24 scenario, with the first 22 being an actual person falling and the last 2 being normal activities that are somewhat similar to the action of falling. Each scenario contains 8 cameras from 8 different angles. Link: http://www.iro.umontreal.ca/~labimage/Dataset/

With these two combined, we have roughly a total of 262 videos.

a. Openpose

After obtaining all the videos, they will be fed into a open framework called Openpose. OpenPose is a real-time multi-person system to jointly detect human body, hand, facial, and foot key points on single images. In this project, we will use this tool to extract the keypoints of a person’s body part, which include a total of 75 keypoints representing 25 parts of a human body, from nose, neck, right shoulder, right elbow, to left hip, left knee, left ankle and so one. However, in this project, we would not use the keypoints for right + left eye and right + left ear since these would not yield any information for the action of falling, with the reason being that people’s facial expression normally does not change as much during falling. This would reduce the keypoints from 75 to 63.

Since OpenPose will create 63 keypoints per frame with 30 frames per second of a video, we would have a large chunk of data. These data will then be filtered to remove invalid keypoints, such as keypoints that do not detect any person body. With this filtration, we have a total of 56268 keypoints, with 3265 of them being a person falling and 53003 of them being not falling. This will create a significantly imbalanced dataset. We will address this issue in the next section

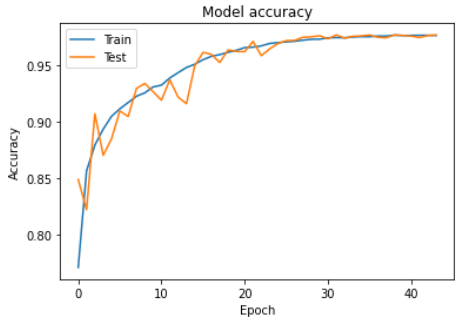
6. Model Training and Performance

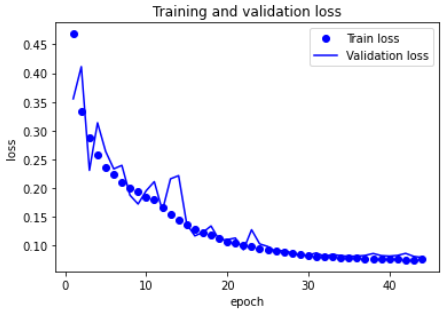
To address the issue of the imbalanced dataset, we first use a technique called up-sampling to scale up the minority class to match the majority class (since we have a binary classification here). With up-sampling, it brings a total data of 53003. Subsequently, we will shuffle these data and normalized them by the video’s width and height to bring the scale of each data value between 0 and 1. To avoid overfitting, we will also have an early stopping with a patience of 5 on accuracy and a dynamic learning rate that shrinks based on the number of epochs. The data set is then split into 90% of training data and 10% or testing data

a. GRU

GRU has a total of five layers, with two 1D convolutional layers and 1 gru layers. Since the convolutional layer is 1D with our binary classification dataset, the kernel size here is simply 1, with the learning rate being 0.001 at the beginning and it shrinks as the model gradually approach the

maximum point. The final training accuracy of this is 0.9766 and the validation accuracy is 0.9766 as well, with the test accuracy being 0.975.

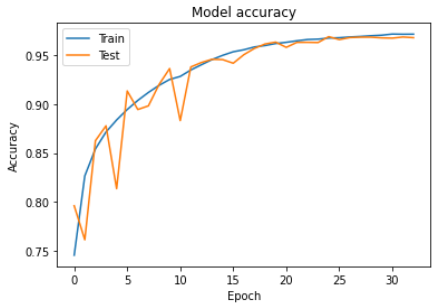


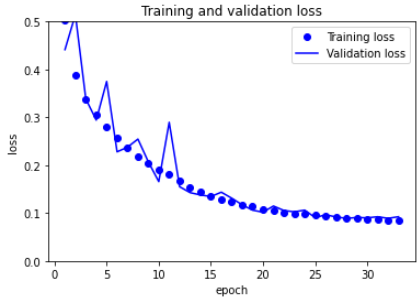


b. LSTM

This RNN network generally is more suitable for large amount of dataset and surprisingly it performs better than the GRU dataset. In this model we have two layers of LSTM and one Dense layer for final classification. The learning rate and the early stop is the same as GRU, with the only exception that the early stop now is in regard of validation loss in stead of training accuracy. The reason for this change is that we are seeing a slight overfit in the previous training for GRU model and early stop focusing on validation loss gives us a better insight on when to stop training. The validation split is set to 0.2 in this model.

With early stop, the training stops at 33 epochs, with a training accuracy of 0.9720 and a validation accuracy of 0.9684. The test set shows an accuracy of 0.9689 as well. Here are the model accuracy and validation loss





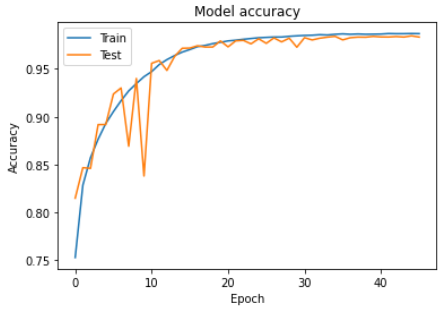
Fortunately we are not seeing any significant overfit here with both training and testing accuracy being similar.

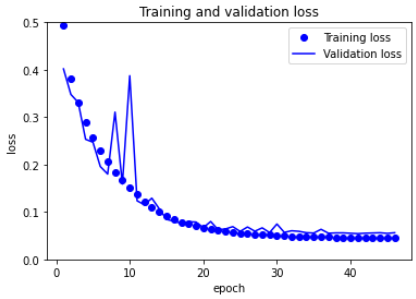
c. biLSTM

This model is called the bidirectional LSTM. The major difference between this model and normal LSTM is that biLSTM will run the data in two directions, one from past to the future and one from future to past while the normal LSTM only preserves information of the past. Therefore, in our scenario it might give a better performance since it will see both direction from standing->falling and falling->standing. All the hyperparameters are the same as LSTM.

The performance is slightly better than LSTM, with the final training accuracy of 0.987 and validation accuracy of 0.9833. The test accuracy is 0.9837, which again is similar to training accuracy. However,

the caveat of this model is efficiency. This model takes roughly 45 minutes versus the 30 minutes for training normal LSTM.





d. CNN

This model has two convolutional layers, two batch_normalization layers, one flatten layer and two dense layer. The convolutional layers have a kernel size of 3, given that we have relatively small dimension and only 63 data points per set. The learning rate and early stop would be same as LSTM and BILSTM.

This model actually yields the best performance, with the highest efficiency and highest accuracy. The training accuracy is 0.9966 and validation accuracy is 0.9953. The test accuracy is 0.9943. And it only takes roughly 8 minutes to finish training.

7. Performance on Youtube Videos

To apply this model to an actual video, a similar approach as the training approach would apply here. The youtube video would be first fed into the OpenPose engine to generate body landmarks, which will be fed into the model and it will produce a probability of people falling of this frame.

Here we choose the CNN model since it yields the highest accuracy on both training and testing set. After obtaining the probability of falling in each frame, we would compute segments of clips by finding consecutive frames with probability higher than 90%. We would then obtain the starting time and the ending time of the segment by using the frame number to calculate the time, based on Openpose generating 30 frames per second. For example, a frame named test_01_00033.json means this is at frame 33 and the time for this is at 33/30 = 1.1 second. In addition, any segment that lasts less than 0.2s would be filtered out. Eventually we have around 853 clips for a total of 10 videos

To check the False-Positive Rate and False-Negative Rate, we need to look into each segment. Unfortunately, with the number of segments being so large, I have to randomly choose 100 of them and check if they are valid. Before listing the actual data, we have to first define false-positive and false-negative.

False positive should be the prediction shows the segment is about a person falling and the person in the video does not fall. False negative should be the prediction shows the person in the segment is not falling but he/she actually does.

Here is a summary of the data in a 100 segment:

False positive: 37 False Negative: 8

Therefore, we have an estimated false-positive rate of 37% and false-negative rate of 8%

8. Improve Accuracy and Efficiency

Given high training/testing accuracy, my training process is extremely low efficient, because I have to use openpose to get the body landmarks first and use the landmark data for training. Both of these process are time consuming. Therefore, the next step would be to 1) not use all the frame in the video and 2) find alternative and faster approach to replace openpose framework. Another idea would be using the “weak supervision” to autonomously label the data instead of me manually doing so.

References:

H. Chao, C. Lai, J. Park and Y. Huang, "Adaptive Body Posture Analysis for Elderly-Falling Detection with Multisensors" in *IEEE Intelligent Systems*, vol. 25, no. 02, pp. 20-30, 2010.

M. Prado, J. Reina-Tosina and L. Roa, "Distributed intelligent architecture for falling detection and physical activity analysis in the elderly," *Proceedings of the Second Joint 24th Annual Conference and the Annual Fall Meeting of the Biomedical Engineering Society] [Engineering in Medicine and Biology*, Houston, TX, USA, 2002, pp. 1910-1911 vol.3, doi: 10.1109/IEMBS.2002.1053088.