Natural Network Project Report

April 26, 2020

Yijun Zhang
629009140

Abstract

For smart home design, fall detection is very important. In this project, I study the vision-based solution using Convolutional Neural Network and Long Short-Term Memory Network to decide if a frame contains a person falling. The figure shows how the probability of falling changes according to the time of the video. The result shows that using the body landmark as input, the Convolutional Neural Network model can achieve accuracy of 0.9612 and Long Short-Term Memory Network could achieve accuracy of 0.9333. The final ensembled model has accuracy 0.9611.

1 Dataset

There are 3 different data sets namely – *'UR', 'MC', 'YOUTUBE'*. There are total 484 videos, including 252 videos with fall and 232 videos without fall.

1.1 UR Fall Detection Dataset[1] (UR)

Figure 1: Example of video in UR Fall Detection Dataset

This dataset contains 70 (30 falls + 40 activities of daily living) sequences. Fall events are recorded with 2 Microsoft Kinect cameras(camera 0 from side angle and camera 1 from directly above angle) and corresponding accelerometric data. ADL events are recorded with only one device (camera 0). It also includes label files: urfall-cam0-falls.csv and urfall-cam0-adls.csv. In these 2 csv files, each row contains one sample of data corresponding to one image frame of a video.

I use videos recorded by camera 0. There are 3 reasons:

- It is from the side angle, which is easier for later body landmark generation.
- Using videos recorded by camera 0 also because it is more similar to the camera set in smart home.
- ADL events are recorded with only one device (camera 0).

1.2 Multiple Camera Dataset[2] (MC)

Figure 2: Example of video in Multiple Camera Dataset

This dataset contains 24 scenarios recorded with 8 IP video cameras. The first 22 first scenarios contain a fall and confounding events; the last 2 ones contain only confounding events. There is a total of 192 videos.

Each video may contain some staff and a person to perform falling action. For the accuracy of extracting body landmarks, I trim each video to only contain one person.

This dataset does not contain any label file. Therefore, I trim each video into 2 parts: one only contains frames of a person from falling to laying down, the other only contains frames of a person doing other things before falling. Later, I mark all frames of the first part to be fall(1) and marked all frames of the second part to be not-fall(0).

1.3 YOUTUBE Dataset

I download videos contains fall from YouTube. There are 23 videos including coach teaching old people how to fall safely, elderly people fall in the hospital, elderly people fall at home, people fall outdoor. For later labeling, I trim each video into 2 parts: one only contains frames of a person from falling to laying down, the other only contains frames of a person doing other things before falling. Later, I mark all frames of the first part to be fall(1) and marked all frames of the second part to be not-fall(0).

1.4 Variety of Videos

A person (such as elderly person, patient) may fall in various ways. There are many actions similar to fall actions. To correctly detect different kinds of fall and to reduce false positive, I consider the variety of videos in my dataset.

Videos in my datasets that contain fall action includes different kinds of fall. For daily activity videos that does not contains fall, there are different kinds of confusion actions.

Table 1: Types of fall videos

Type of fall	**Number of video**
Includes elderly person	10
Fall forward	187

Fall backward	25
Fall sideward	40
Outdoor	8
Indoor	244
Kick something then fall	10
Dizzy then fall	2

Table 2: Confusion Action of videos

Confusion Action	**Number of video**
Find or pick something up on the ground	31
Lay down in sofa	10
Squat	21
Lay down in bed	24
Sit in chair/sofa	35
Tie shoelaces	9
Do push up	8

2 Data Processing

2.1 Body Landmark Extraction

I use the open source tool[3][4][5][6] from CMU to produce body landmark for frames of each video. The website of this tool: https://github.com/CMU-Perceptual-Computing-Lab/openpose.

All body landmarks for each frame are stored in a json file. The generated body landmarks for each frame contains 25 key points: {0, "Nose"}, {1, "Neck"}, {2, "RShoulder"}, {3, "RElbow"}, {4, "RWrist"}, {5, "LShoulder"}, {6, "LElbow"}, {7, "LWrist"}, {8, "MidHip"}, {9, "RHip"},{10, "RKnee"},{11, "RAnkle"},{12, "LHip"},{13, "LKnee"}, {14, "LAnkle"}, {15, "REye"}, {16, "LEye"}, {17, "REar"}, {18, "LEar"}, {19, "LBigToe"}, {20, "LSmallToe"}, {21, "LHeel"}, {22, "RBigToe"}, {23, "RSmallToe"}, {24, "RHeel"}, {25, "Background"}.

For fall detection, I think body landmarks of the face are unnecessary: {15, "REye"}(right eye), {16, "LEye"}(left eye), {17, "REar"}(right ear), {18, "LEar"}(left ear). After reading body landmarks from JSON files, I remove these 4 key points for each frame.

In case of unstable connection of Google Colab or crash of local machine. I store body landmarks into CSV files so that later I can read these data from CSV files, which will be faster than re-read all JSON files.

2.2 Labeling

UR dataset has its label files for each frame. Labels are stored in csv files. Attribute in csv files includes:

- sequence name - camera name is omitted, because all of the samples are from the front camera ('fall-01-cam0-d' is 'fall-01', 'adl-01-cam0-d' is 'adl-01' and so on).
- frame number - corresponding to number in sequence.
- label - describes human posture in the depth frame; '-1' means person is not lying, '1' means person is lying on the ground; '0' is temporary pose, when person "is falling", we don't use '0' frames in classification.

I transform the label using the rule:

- For the falling videos, original label '-1' means 0 (not falling), and original label '0' and '1' means 1 (falling).
- For the activities of daily living videos, all frames are labeled 0 (not falling).

Videos in MC dataset and YOUTUBE dataset are manually trimmed to 2 parts: one only contains frames of a person from falling to laying down, the other only contains frames of a person doing other things before falling. I mark all frames of the first part to be fall(1) and marked all frames of the second part to be not-fall(0).

2.3 Normalization and Up-Sampling

I normalize data by dividing the width and height of the videos.

```
# normalize data according to the size of videos in different dataset
def normalize(list, width, height):
  for record in list:
    for i in range(0, 63, 3):
        record[i] = float(record[i])/ width
        record[i + 1] = float(record[i + 1]) / height

normalize(ur_data, 640, 240)
normalize(mc_fall_data, 720, 480)
normalize(mc_notfall_data, 720, 480)
normalize(youtube_fall_data, 640, 360)
normalize(youtube_notfall_data, 640, 360)
```

Figure 3: Annotated code of data normalization.

There are 20661 frames are not fall(0) and 13017 frames of fall(1). To make the data balance, I used up-sampling. After up-sampling, both fall frames and not fall frames are 20661.

Table 3: Frames before and after Up-sampling

	Fall frames(1)	**Not fall frames (0)**
Before up-sampling	12895	19787
After up-sampling	19787	19787

```python
# up-sampling to make the data balance
# Separate majority and minority classes
df = pd.DataFrame.from_records(all_data)
header = ['label']
for i in range(63):
  header.append(i)

df.columns = header
df_majority = df[df.label==0]
df_minority = df[df.label==1]
print("before re-sampling, not fall vs fall: ")
print(df['label'].value_counts())

from sklearn.utils import resample
print("begin to re sample...")
# Upsample minority class
df_minority_upsampled = resample(df_minority,
                                 replace=True,      # sample with replacement
                                 n_samples=19787)    # to match majority class

# Combine majority class with upsampled minority class
df_upsampled = pd.concat([df_majority, df_minority_upsampled])

print("after re-sampling...")
# Display new class counts
df_upsampled.label.value_counts()
```

```
before re-sampling, not fall vs fall:
0    19787
1    12895
Name: label, dtype: int64
begin to re sample...
after re-sampling...
1    19787
0    19787
Name: label, dtype: int64
```

Figure 4: Annotated code of up-sampling.

2.4 Train-Test set Split

I randomly shuffle the data to ensure that each data point creates an "independent" change on the model, without being biased by the same points before them.

Then, I split the dataset into training set and test set.

```
# split data into train and test set
np.random.shuffle(all_data)
split_point = len(all_data) // 9
print("split point:", split_point)
test_data = all_data[:split_point]
train_data = all_data[split_point: ]

print("number of test data:", len(test_data))
print("number of train data:",len(train_data))

print(train_data[0])
```

Figure 5: Annotated code of train-test set split.

3 Model Training

3.1 LSTM model Training

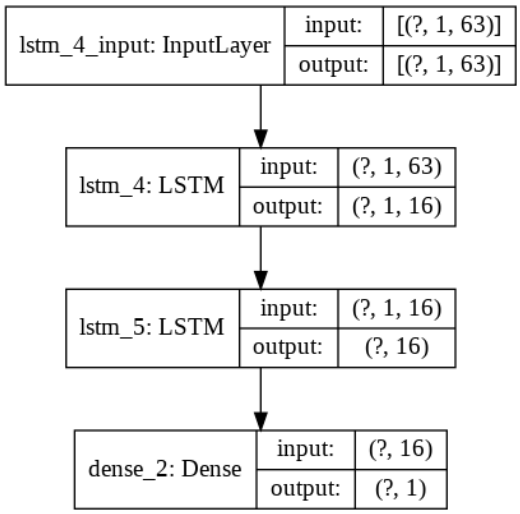

Figure 6: Shape of tensor of each layer of LSTM model

Table 4: Shape of tensor of the first and last layer of LSTM model

Input shape of tensor	x_train: (35177, 1, 63)
	x_test: (4397, 1, 63)
Output shape of tensor	y_train shape is (35177,16)
	y_test shape is (4397,1)

The LSTM model is built on a linear stack of layers with the sequential model. There are two hidden layers in total.

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 1, 16)             5120
_________________________________________________________________
lstm_5 (LSTM)                (None, 16)                2112
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 17
=================================================================
Total params: 7,249
Trainable params: 7,249
Non-trainable params: 0
_________________________________________________________________
```

Figure 7: Summary of LSTM model.

For learning rate, I define a scheduler. When epoch is less than 10, the learning rate will be 0.01. When epoch is more then 10, it will decrease as epoch increase.

I apply early stopping with a patience 5 epochs, which means if the loss is not improving for 5 epochs, the training will stop. This policy can avoid unnecessary full training.

I try different hyperparameters, and combination of the best performance is Batch size 16, epochs 52 and without dropout.

Table 5: Performance comparison of different batch size

Batch size	**16**	**32**	**64**
Loss	0.2491	0.2668	0.2962
Accuracy	0.8883	0.8789	0.8624

Best batch size is 16.

Table 6: Performance comparison of different drop out

Drop out	**none**	**0.1**	**0.5**
Loss	0.2491	0.2884	0.2544
Accuracy	0.8883	0.8789	0.8801

The performance does not improve, so I removed dropout.

According to the above discussion, hyperparameters of LSTM model are as follows:

Table 7: Hyperparameters of LSTM model

Batch Size	16
Epochs	52
Dropout	(unnecessary)

A snippet of code of constructing the model/network is shown as follows:

```
def scheduler(epoch):
  if epoch < 10:
    return 0.001
  else:
    return 0.001 * np.exp(0.1 * (10 - epoch))

x_train = []
y_train = []

for record in train_data:
  x_train.append(record[1:])
  label = int(record[0])
  y_train.append(label)

def train_lstm_model(x_train, y_train):
  x_train = array(x_train)
  x_train = x_train.reshape((len(x_train), 1, len(x_train[0])))
  print("x_train.shape", x_train.shape)
  print(x_train[0])

  y_train = array(y_train)
  print("y_train.shape", y_train.shape)

  # imrpove log: use batch size 16 and add one more lstm layer

  lstm_model = Sequential()
  lstm_model.add(LSTM(16,
                input_shape=(1, 63),
                return_sequences=True))
  lstm_model.add(LSTM(16, ))
  lstm_model.add(layers.Dense(1, activation='sigmoid'))
  lstm_model.compile(optimizer='rmsprop',
                loss='binary_crossentropy',
                metrics=['acc',
                         metrics.AUC(),
                         metrics.FalseNegatives(),
                         metrics.Recall(),
                         metrics.Precision(),
                         metrics.FalseNegatives(),
                         metrics.TrueNegatives(),
                         metrics.FalsePositives(),
                         metrics.TruePositives()])
  lstm_history = lstm_model.fit(x_train, y_train,
                      epochs=100,
                      batch_size=16,
                      validation_split=0.2,
                      callbacks=[callbacks.EarlyStopping(monitor='val_loss', patience=5),
                      callbacks.LearningRateScheduler(scheduler)])
  print("finish training lstm model")
  return lstm_model, lstm_history

lstm_model, lstm_history = train_lstm_model(x_train, y_train)
```

Figure 8: Annotated code of LSTM model training

Training and Testing performance of LSTM model are as follows:

```
Epoch 1/100
1798/1798 [==============================] - 21s 12ms/step - loss: 0.4138 - acc: 0.8093 - auc_1: 0.8223
Epoch 2/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.2858 - acc: 0.8848 - auc_1: 0.9150
Epoch 3/100
1798/1798 [==============================] - 21s 12ms/step - loss: 0.2544 - acc: 0.8995 - auc_1: 0.9347
Epoch 4/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.2363 - acc: 0.9078 - auc_1: 0.9441
Epoch 5/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.2245 - acc: 0.9126 - auc_1: 0.9499
Epoch 6/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.2151 - acc: 0.9149 - auc_1: 0.9541
Epoch 7/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.2075 - acc: 0.9187 - auc_1: 0.9572
Epoch 8/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.2013 - acc: 0.9205 - auc_1: 0.9596
Epoch 9/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1959 - acc: 0.9214 - auc_1: 0.9616
Epoch 10/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1904 - acc: 0.9236 - auc_1: 0.9633
Epoch 11/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1866 - acc: 0.9258 - auc_1: 0.9646
Epoch 12/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1818 - acc: 0.9286 - auc_1: 0.9659
Epoch 13/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1773 - acc: 0.9289 - auc_1: 0.9670
Epoch 14/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1734 - acc: 0.9307 - auc_1: 0.9681
Epoch 15/100
1798/1798 [==============================] - 21s 11ms/step - loss: 0.1706 - acc: 0.9318 - auc_1: 0.9690
Epoch 16/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1680 - acc: 0.9332 - auc_1: 0.9698
Epoch 17/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1657 - acc: 0.9346 - auc_1: 0.9706
Epoch 18/100
1798/1798 [==============================] - 22s 12ms/step - loss: 0.1637 - acc: 0.9346 - auc_1: 0.9713
Epoch 19/100
1798/1798 [==============================] - 21s 12ms/step - loss: 0.1614 - acc: 0.9369 - auc_1: 0.9719
Epoch 20/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1601 - acc: 0.9360 - auc_1: 0.9725
Epoch 21/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1587 - acc: 0.9375 - auc_1: 0.9731
Epoch 22/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1574 - acc: 0.9374 - auc_1: 0.9736
Epoch 23/100
1798/1798 [==============================] - 21s 11ms/step - loss: 0.1556 - acc: 0.9392 - auc_1: 0.9741
Epoch 24/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1551 - acc: 0.9392 - auc_1: 0.9745
Epoch 25/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1542 - acc: 0.9387 - auc_1: 0.9749
Epoch 26/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1534 - acc: 0.9390 - auc_1: 0.9753
Epoch 27/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1524 - acc: 0.9398 - auc_1: 0.9756
Epoch 28/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1517 - acc: 0.9404 - auc_1: 0.9760
Epoch 29/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1512 - acc: 0.9407 - auc_1: 0.9763
Epoch 30/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1506 - acc: 0.9410 - auc_1: 0.9766
Epoch 31/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1503 - acc: 0.9405 - auc_1: 0.9769
Epoch 32/100
1798/1798 [==============================] - 21s 12ms/step - loss: 0.1499 - acc: 0.9407 - auc_1: 0.9771
Epoch 33/100
1798/1798 [==============================] - 21s 12ms/step - loss: 0.1493 - acc: 0.9404 - auc_1: 0.9773
Epoch 34/100
1798/1798 [==============================] - 21s 12ms/step - loss: 0.1489 - acc: 0.9418 - auc_1: 0.9776
Epoch 35/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1484 - acc: 0.9414 - auc_1: 0.9778
Epoch 36/100
1798/1798 [==============================] - 21s 11ms/step - loss: 0.1483 - acc: 0.9409 - auc_1: 0.9780
Epoch 37/100
1798/1798 [==============================] - 21s 11ms/step - loss: 0.1479 - acc: 0.9424 - auc_1: 0.9782
Epoch 38/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1479 - acc: 0.9412 - auc_1: 0.9784
Epoch 39/100
1798/1798 [==============================] - 20s 11ms/step - loss: 0.1475 - acc: 0.9423 - auc_1: 0.9785
Epoch 40/100
1798/1798 [==============================] - 21s 11ms/step - loss: 0.1473 - acc: 0.9423 - auc_1: 0.9787
```

```
Epoch 41/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1696 - acc: 0.9312 - auc_5: 0.9734
Epoch 42/100
1759/1759 [==============================] - 20s 12ms/step - loss: 0.1694 - acc: 0.9315 - auc_5: 0.9736
Epoch 43/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1694 - acc: 0.9307 - auc_5: 0.9738
Epoch 44/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1692 - acc: 0.9305 - auc_5: 0.9739
Epoch 45/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1692 - acc: 0.9307 - auc_5: 0.9741
Epoch 46/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1690 - acc: 0.9308 - auc_5: 0.9743
Epoch 47/100
1759/1759 [==============================] - 20s 12ms/step - loss: 0.1689 - acc: 0.9308 - auc_5: 0.9744
Epoch 48/100
1759/1759 [==============================] - 21s 12ms/step - loss: 0.1687 - acc: 0.9315 - auc_5: 0.9746
Epoch 49/100
1759/1759 [==============================] - 20s 12ms/step - loss: 0.1686 - acc: 0.9310 - auc_5: 0.9747
Epoch 50/100
1759/1759 [==============================] - 20s 12ms/step - loss: 0.1685 - acc: 0.9310 - auc_5: 0.9748
Epoch 51/100
1759/1759 [==============================] - 21s 12ms/step - loss: 0.1684 - acc: 0.9315 - auc_5: 0.9750
Epoch 52/100
1759/1759 [==============================] - 21s 12ms/step - loss: 0.1684 - acc: 0.9316 - auc_5: 0.9751
Epoch 53/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1683 - acc: 0.9311 - auc_5: 0.9752
Epoch 54/100
1759/1759 [==============================] - 20s 12ms/step - loss: 0.1684 - acc: 0.9311 - auc_5: 0.9753
Epoch 55/100
1759/1759 [==============================] - 20s 12ms/step - loss: 0.1682 - acc: 0.9314 - auc_5: 0.9755
Epoch 56/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1682 - acc: 0.9310 - auc_5: 0.9756
Epoch 57/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1681 - acc: 0.9311 - auc_5: 0.9757
Epoch 58/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1681 - acc: 0.9312 - auc_5: 0.9758
Epoch 59/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1681 - acc: 0.9313 - auc_5: 0.9759
Epoch 60/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1680 - acc: 0.9308 - auc_5: 0.9760
Epoch 61/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1680 - acc: 0.9312 - auc_5: 0.9761
Epoch 62/100
1759/1759 [==============================] - 20s 12ms/step - loss: 0.1680 - acc: 0.9313 - auc_5: 0.9762
Epoch 63/100
1759/1759 [==============================] - 20s 12ms/step - loss: 0.1679 - acc: 0.9313 - auc_5: 0.9762
Epoch 64/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1678 - acc: 0.9316 - auc_5: 0.9763
Epoch 65/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1680 - acc: 0.9313 - auc_5: 0.9764
Epoch 66/100
1759/1759 [==============================] - 20s 12ms/step - loss: 0.1679 - acc: 0.9313 - auc_5: 0.9765
Epoch 67/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1678 - acc: 0.9311 - auc_5: 0.9766
Epoch 68/100
1759/1759 [==============================] - 21s 12ms/step - loss: 0.1679 - acc: 0.9312 - auc_5: 0.9766
Epoch 69/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1678 - acc: 0.9314 - auc_5: 0.9767
Epoch 70/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1678 - acc: 0.9314 - auc_5: 0.9768
Epoch 71/100
1759/1759 [==============================] - 20s 12ms/step - loss: 0.1678 - acc: 0.9311 - auc_5: 0.9768
Epoch 72/100
1759/1759 [==============================] - 20s 12ms/step - loss: 0.1678 - acc: 0.9312 - auc_5: 0.9769
Epoch 73/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1678 - acc: 0.9311 - auc_5: 0.9769
Epoch 74/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1678 - acc: 0.9312 - auc_5: 0.9770
Epoch 75/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1678 - acc: 0.9312 - auc_5: 0.9771
Epoch 76/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1678 - acc: 0.9313 - auc_5: 0.9772
Epoch 77/100
1759/1759 [==============================] - 20s 11ms/step - loss: 0.1678 - acc: 0.9312 - auc_5: 0.9772
finish training lstm model
```

Figure 9: Training log of LSTM model training

```
1100/1100 [==============================] - 7s 7ms/step - loss: 0.1700 - acc: 0.9305 - auc_5: 0.9773 - false_negatives_9: 76206.8984 - recall_5: 0.9440 - precision_5: 0.9070
[0.16995015740394592, 0.9304943680763245, 0.9772706031799316, 76206.8984375, 0.944014310836792, 0.9069869518280029, 76206.8984375, 1233348.75, 131772.5, 1284915.5]
```

Figure 10: Testing result of LSTM model

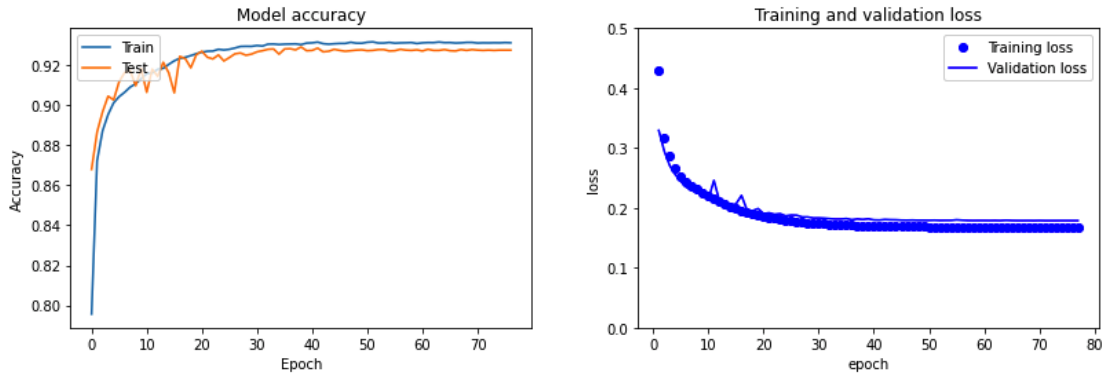

Figure 11: Performance of LSTM model training

Table 8: Test result of LSTM model

Testing Loss	0.1700
Testing Accuracy	0.9305
AUC	0.9773
Sensitivity /Recall $(\boldsymbol{Sensitivity} = \frac{\boldsymbol{TP}}{\boldsymbol{TP+FN}})$	0.9440
Specificity $(\boldsymbol{Specificity} = \frac{\boldsymbol{TN}}{\boldsymbol{TN+FP}})$	0.9070

After training the model and use test dataset to get the test score, I trained the final LSTM model using all the data.

```
# train the final lstm model with all data (include test data)

x_train_final = []
y_train_final = []

for record in all_data:
  x_train_final.append(record[1:])
  label = int(record[0])
  y_train_final.append(label)

final_lstm_model, final_lstm_history = train_lstm_model(x_train_final, y_train_final)

final_lstm_model.save("/content/drive/My Drive/636project/final_lstm_model.h5")

```

Figure 12: Annotated code of LSTM model training using all data

3.2 CNN model Training

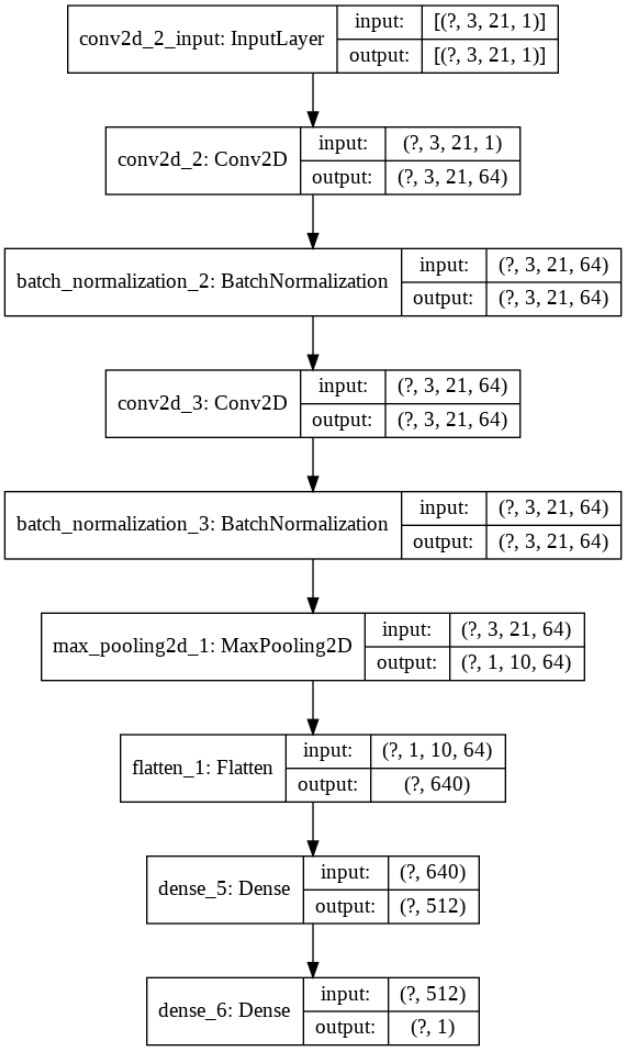

Figure 13: Shape of tensor of each layer of CNN model

Table 9: Shape of tensor of the first and last layer of CNN model

Input shape of tensor	x_train: (35177, 3, 21, 63)
	x_test: (4397, 3, 21, 1)
Output shape of tensor	y_train shape is (35177,512)
	y_test shape is (4397,1)

The CNN model is built on a linear stack of layers with the sequential model. There are 2 2D-convolution layers, 1 max pooling layers and 2 dense layers.

I added batch normalization between 2 convolution layers to adaptively normalize data as the mean and variance change over time during training.

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 3, 21, 64)         640
_________________________________________________________________
batch_normalization_2 (Batch (None, 3, 21, 64)         12
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 21, 64)         36928
_________________________________________________________________
batch_normalization_3 (Batch (None, 3, 21, 64)         12
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 1, 10, 64)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 640)               0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               328192
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 513
=================================================================
Total params: 366,297
Trainable params: 366,285
Non-trainable params: 12
_________________________________________________________________
```

Figure 14: Summary of CNN model.

According to the above discussion, Hyperparameters of CNN model are as follows:

Table 10: Hyperparameters of CNN model

Batch Size	16
Epochs	18
Dropout	(unnecessary)

For the learning rate, I use the same scheduler as LSTM model. When the epoch is less than 10, the learning rate will be 0.01. When the epoch is more than 10, it will decrease as epoch increases.

I apply early stopping with a patience 5 epochs, which means if the loss is not improving for 5 epochs, the training will stop. This policy can avoid unnecessary full training.
I try different hyperparameters, and the combination of the best performance is Batch size 16, epochs 22 and without dropout.

Table 11: Performance comparison of different batch size

Batch size	**16**	**32**	**64**
Loss	0.0667	0.0713	0.0812
Accuracy	0.9692	0.9621	0.9343

Best batch size is 16.

Table 12: Performance comparison of different drop out

Drop out	**none**	**0.1**	**0.5**
Loss	0.0667	0.0877	0.0670
Accuracy	0.9692	0.9601	0.9652

The performance does not improve, so I remove dropout.

Regarding the optimization of the network parameters, I found Adam is better than RMSprop here.

Table 13: Performance comparison of different optimizer

Optimizer	**Adam**	**RMSprop**
Loss	0.0667	0.0877
Accuracy	0.9692	0.9601

A snippet of code of constructing the model/network is shown as follows:

```
19 def train_cnn_model(x_train, y_train):
20   x_train = array(x_train)
21   x_train = x_train.reshape((len(x_train), 3, int(len(x_train[0])/3), 1))
22
23   y_train = array(y_train)
24
25   #create model
26   cnn_model = Sequential()
27   cnn_model.add(Conv2D(64,
28                   kernel_size=3,
29                   activation='relu',
30                   input_shape=(3,21,1),
31                   padding='same'))
32   cnn_model.add(layers.BatchNormalization(1))
33   cnn_model.add(Conv2D(64,
34                   kernel_size=3,
35                   activation='relu',
36                   padding='same'))
37   cnn_model.add(layers.BatchNormalization(1))
38   cnn_model.add(MaxPooling2D(2,2))
39   cnn_model.add(Flatten())
40   cnn_model.add(Dense(512, activation = 'relu'))
41   cnn_model.add(Dense(1, activation='sigmoid'))
42
43   # compile and fit
44   cnn_model.compile(optimizer='Adam',
45                 loss='binary_crossentropy',
46                 metrics=['acc',
47                         metrics.AUC(),
48                         metrics.FalseNegatives(),
49                         metrics.Recall(),
50                         metrics.Precision(),
51                         metrics.FalseNegatives(),
52                         metrics.TrueNegatives(),
53                         metrics.FalsePositives(),
54                         metrics.TruePositives()])
55   cnn_history = cnn_model.fit(x_train, y_train,
56                       epochs=100,
57                       batch_size=16,
58                       validation_split=0.2,
59                       callbacks=[callbacks.EarlyStopping(monitor='val_loss', patience=5),
60                       callbacks.LearningRateScheduler(scheduler)])
61
62   print("finish training cnn model")
63   return cnn_model, cnn_history
64
65
66 cnn_model, cnn_history = tain_cnn_model(x_train, y_train)
```

Figure 15: Annotated code of CNN model training

Training and Testing performance of CNN models are as follows:

```
Epoch 1/100
1759/1759 [==============================] - 17s 10ms/step - loss: 0.2900 - acc: 0.8794 - auc_6: 0.9043
Epoch 2/100
1759/1759 [==============================] - 16s 9ms/step - loss: 0.1798 - acc: 0.9239 - auc_6: 0.9616
Epoch 3/100
1759/1759 [==============================] - 16s 9ms/step - loss: 0.1479 - acc: 0.9386 - auc_6: 0.9720
Epoch 4/100
1759/1759 [==============================] - 16s 9ms/step - loss: 0.1286 - acc: 0.9447 - auc_6: 0.9773
Epoch 5/100
1759/1759 [==============================] - 16s 9ms/step - loss: 0.1144 - acc: 0.9498 - auc_6: 0.9805
Epoch 6/100
1759/1759 [==============================] - 16s 9ms/step - loss: 0.1061 - acc: 0.9538 - auc_6: 0.9828
Epoch 7/100
1759/1759 [==============================] - 17s 9ms/step - loss: 0.1002 - acc: 0.9559 - auc_6: 0.9846
Epoch 8/100
1759/1759 [==============================] - 16s 9ms/step - loss: 0.0914 - acc: 0.9597 - auc_6: 0.9860
Epoch 9/100
1759/1759 [==============================] - 15s 9ms/step - loss: 0.0885 - acc: 0.9602 - auc_6: 0.9871
Epoch 10/100
1759/1759 [==============================] - 16s 9ms/step - loss: 0.0872 - acc: 0.9605 - auc_6: 0.9880
Epoch 11/100
1759/1759 [==============================] - 16s 9ms/step - loss: 0.0826 - acc: 0.9627 - auc_6: 0.9886
Epoch 12/100
1759/1759 [==============================] - 16s 9ms/step - loss: 0.0778 - acc: 0.9641 - auc_6: 0.9893
Epoch 13/100
1759/1759 [==============================] - 16s 9ms/step - loss: 0.0715 - acc: 0.9665 - auc_6: 0.9898
Epoch 14/100
1759/1759 [==============================] - 16s 9ms/step - loss: 0.0717 - acc: 0.9664 - auc_6: 0.9903
Epoch 15/100
1759/1759 [==============================] - 16s 9ms/step - loss: 0.0682 - acc: 0.9675 - auc_6: 0.9907
Epoch 16/100
1759/1759 [==============================] - 16s 9ms/step - loss: 0.0664 - acc: 0.9682 - auc_6: 0.9911
finish training cnn model
```

Figure 16: Training log of CNN model training

```
1100/1100 [==============================] - 6s 6ms/step - loss: 0.0800 - acc: 0.9640 - auc_6: 0.9914 - false_negatives_11: 7782.6284 - recall_6: 0.9731 - precision_6: 0.9324 -
[0.08004513382911682, 0.9640390276908875, 0.9914093613624573, 7782.62841796875, 0.9731386303901672, 0.9323545098304749, 7782.62841796875, 270209.75, 20458.236328125, 281997.25]
```

Figure 17: Testing result of CNN model

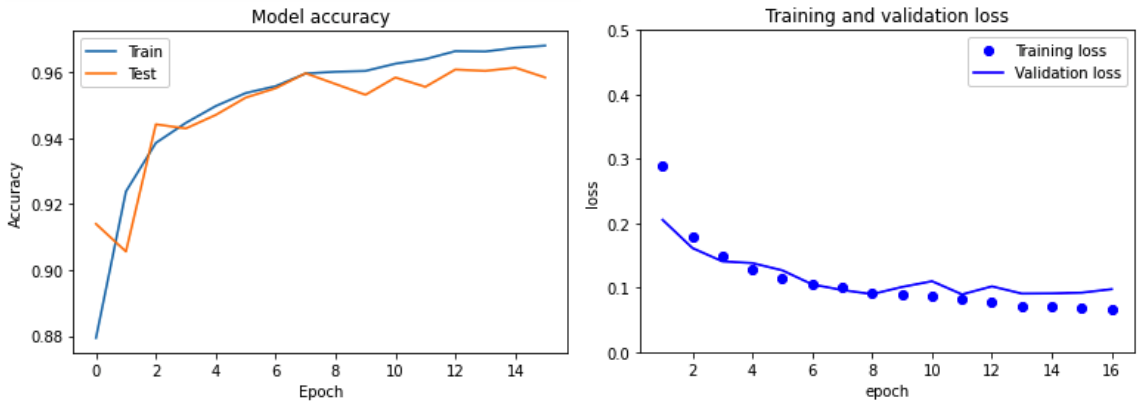

Figure 18: Performance of CNN model training

Table 14: Test result of CNN model

Testing Loss	0.0800
Testing Accuracy	0.9640
AUC	0.9914

Sensitivity / Recall $(Sensitivity = \frac{TP}{TP+FN})$	0.9731
Specificity $(Specificity = \frac{TN}{TN+FP})$	0.9324

After training the model and use test dataset to get the test score, I trained the final CNN model using all the data.

```
# train the final cnn model with all data (include test data)
x_train_final = []
y_train_final = []

for record in all_data:
  x_train_final.append(record[1:])
  label = int(record[0])
  y_train_final.append(label)

final_cnn_model, final_cnn_history = train_cnn_model(x_train_final, y_train_final)

final_cnn_model.save("/content/drive/My Drive/636project/final_cnn_model.h5")
```

Figure 19: Annotated code of CNN model training using all data

3.3 Ensemble Model Training

The LSTM model and CNN model are good models and they prediction from different aspects. I ensemble these two models with different ratio($prediction\ of\ ensembled\ model\ =\ prediction\ of\ CNN\ model\ *\ fraction1\ +\ prediction\ of\ LSTM\ model\ *\ fraction2,\ fraction1\ +\ fraction2\ =\ 1$) and compare the performance(*Accuracy* = $\frac{(TruePositive+TrueNegative)}{ALLframes}$) of ensembled model and CNN model and LSTM model.

Table 15: Ratio of CNN model to LSTM model, *Accuracy* comparison

CNN vs. LSTM	Ensembled model	CNN model	LSTM model
0.1 : 0.9	0.9348877057707606	0.9615996573037146	0.931093568325072
0.2 : 0.8	0.9393549966342329		
0.3 : 0.7	0.9439140811455847		
0.4 : 0.6	0.9499724619056361		
0.5 : 0.5	0.9607123187075455		
0.6 : 0.4	0.9615078636558351		
0.7 : 0.3	0.9618444403647267		
0.8 : 0.2	**0.9619056361299798**		
0.9 : 0.1	0.9616914509515941		

It is shown that the ensembled model has improved the accuracy of prediction. With *ratio 0.8 : 0.2*(CNN vs. LSTM) the ensembled model have better performance(0.9619) compare to

CNN model(0.9615) and LSTM model(0.9310). Also, for predictions of frames for a whole video, the ensembled model has better performance.
These are two test case for two of the test videos:

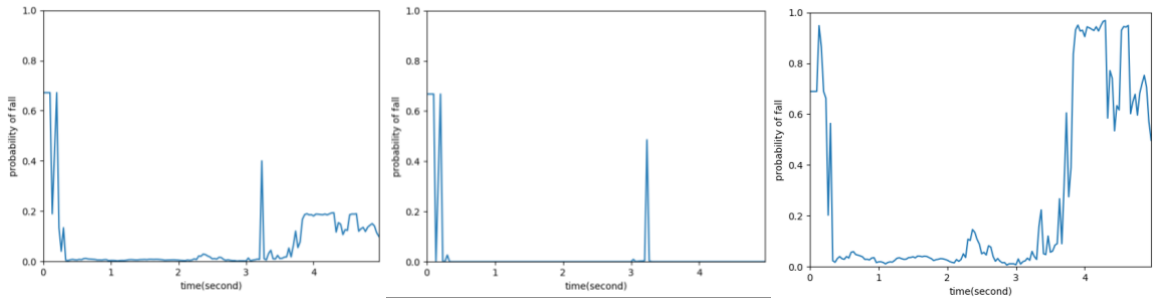

Figure 20: Prediction for video UR-adl-01-cam0 (no fall) of different models

from left to right: ensembled model, CNN model, LSTM model

The above video does not contain fall action, we can see that in the third second, the ensembled model has a prediction of less than 0.5, while CNN model has a prediction of 0.5. In the fourth and fifth seconds, the ensembled model performs better than LSTM model.

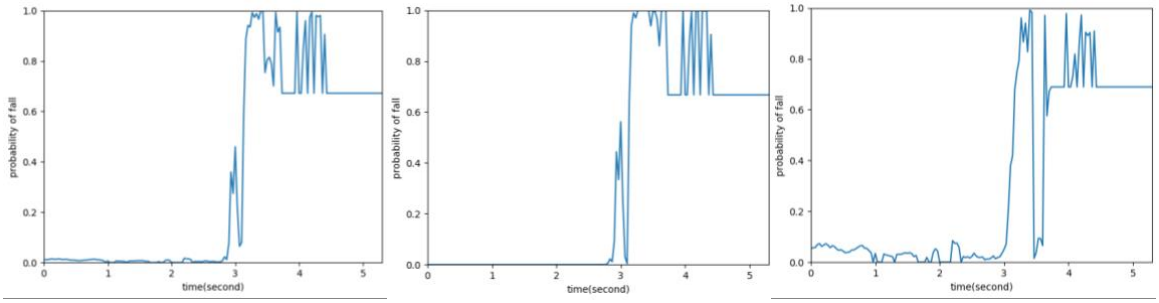

Figure 21: Prediction for video UR-fall-01-cam0 (contains fall) of different models

from left to right: ensembled model, CNN model, LSTM model

In this video, in the first 2.5 seconds, the person does not fall, then he falls and lays down until the end. The prediction of the ensembled model is much more accurate than LSTM model.

Therefore, I decide to make the final model for prediction as the ensembled model.

This is the code of model ensembling:

```
# use model and write the json file for a video
print("loading model and computing probability for each frame...")
cnn_model = keras.models.load_model('/content/drive/My Drive/636project/final_cnn_model.h5')
lstm_model = keras.models.load_model('/content/drive/My Drive/636project/final_lstm_model.h5')

# for cnn model, reshape and predict
test_cnn = array(test_cnn)
test_cnn = test_cnn.reshape((len(test_cnn), 3, int(len(test_cnn[0])/3), 1))
probability_cnn = cnn_model.predict_proba(test_cnn)

# for lstm model, reshape and predict
test_lstm = array(test_lstm)
test_lstm = test_lstm.reshape((len(test_lstm), 1, len(test_lstm[0])))
probability_lstm = lstm_model.predict_proba(test_lstm)

# emsemble the prediction of cnn and lstm model
frame_num = 0

# fp: false positive
# fn: false negative
fp = []
fn = []
for i in range(len(probability_cnn)):
    timestamp = frame_num / 30
    probability_fall = float(probability_cnn[i][0]) * 0.9 + float(probability_lstm[i][0]) * 0.1
  # emsembled model
    if probability_fall < 0.5:
      prediction = 0.0
    else:
      prediction = 1.0

    if prediction == float(y_test[i]):
      correct += 1
    else:
      incorrect += 1
      if prediction == 1.0:
        fp.append(i)
      else:
        fn.append(i)

  # cnn model
    if probability_cnn[i][0] < 0.5:
      prediction_cnn = 0.0
    else:
      prediction_cnn = 1.0

    if prediction_cnn == float(y_test[i]):
      cnn_correct += 1
    else:
      cnn_incorrect += 1

  # lstm model
    if probability_lstm[i][0] < 0.5:
      prediction_lstm = 0.0
    else:
      prediction_lstm = 1.0

    if prediction_lstm == float(y_test[i]):
      lstm_correct += 1
    else:
      lstm_incorrect += 1

print('finish prediction')

print('emsembled model')
print(correct)
print(incorrect)
print(correct / (correct + incorrect))
```

```
# get name of fail frames
false_positive_frames = []
false_negative_frames = []

for fail in fp:
  false_positive_frames.append(frame_name[fail])


for fail in fn:
  false_negative_frames.append(frame_name[fail])

print(len(false_positive_frames))
print(len(false_negative_frames))
```

Figure 22: Annotated code of model ensembling

4 Problems and Fail Cases

For all data I used, I use the final model(ensembled model) to get all fail cases. For a total of 32682 video frames, there are 1309 fail cases. In these fails frames, there are 1185 false-positive frames(actual fall but predicts not fall) 124 false-negative frames(actual not fall but predicts fall). For all these fail cases, I got a CSV file of their video number and frame number. I could analyze them by checking each frame.

```
# get name of fail frames
false_positive_frames = []
false_negative_frames = []

for fail in fp:
  false_positive_frames.append(frame_name[fail])


for fail in fn:
  false_negative_frames.append(frame_name[fail])

print(len(false_positive_frames))
print(len(false_negative_frames))
```

```
1185
124
```

```
# save names of fail frames
df_false_negative = pd.DataFrame.from_records(false_negative_frames)
df_false_negative.to_csv (r'/content/drive/My Drive/636project/false_negative_frames.csv', index = False, header=False)
df_false_positive = pd.DataFrame.from_records(false_positive_frames)
df_false_positive.to_csv (r'/content/drive/My Drive/636project/false_positive_frames.csv', index = False, header=False)
```

Figure 23: Annotated code of getting fail cases

By checking the image of fail video frames. I summarize main reasons of fail prediction.

Table 16: Reasons of fail prediction

Fail reason	Number of fail frames	Percentage
Uncertainty of falling action	610	46.6%
Confusing actions	431	32.9%
Uncertain reason	268	20.5%

First, the uncertainty of falling action. These fail cases happen before the fall and are consider as fall. The main reason for this failure is because the trimming and labeling of videos.

UR dataset has a label for each frame, but the Multiple Camera Dataset does not have. Then I trimmed each video into 2 parts: the first part is before fall(the person may walk, sit or stand)

and the second part is falling and laying on the floor. I labeled all frames of the first part to be 0(not fall) and all frames of the second part to be 1(fall). The trimming is judge by a human, and the fall action is a consecutive action. Therefore, there is no exact time to trim the video.

For example, the frame image below shows a person is leaning and about to fall. I trimmed the video at this moment. However, these two frames will have every similar body landmark data.

Figure 24: Video 59 from Multiple Camera Dataset

Left frame: marked as fall;

Right frame: marked as not fall

Second, not enough data for confusing actions.

The actor performs confusing actions. For example, laying on the sofa bending to seek objects under the sofa or grabbing an object on the floor. There are some videos contains confusing actions for training, but not enough. Then the network does not have enough samples to learn.

Videos from YOUTUBE dataset improved the performance by adding videos of real video of old people falling in the hospital or people falling outdoor. However, it is still not enough videos. Fall is a dangerous and not fun action, there are not many videos could be found from the internet and it also hurts to trying make fall video by myself.

If there are enough video data of different kinds of fall and confusion actions, my training structure will get a more precise model.

Third, other uncertain reasons.

I did not know the exact reason of other fail predictions. It could be the limitations of the cameras and the body landmark extraction algorithm.

The UR dataset only has two camera positions (parallel to the floor and ceiling-mounted, respectively). The video from ceiling mounted cannot generate body landmark, so I only use video recorded by camera parallel to the floor. Though the Multiple Camera Dataset contains 8 cameras from different positions, there are only 22 videos contains fall. Videos from YOUTUBE dataset has different angles but there are only 46 videos of this dataset.

Therefore, there is not enough video from different positions for training. The body landmark extraction also has its limitations and cannot be 100% accurate.

5 Smart Home Solution

I found that for every 30 frames of each video, there are less than 15 consecutive fail frames. Therefore, for a real application, the problem of fail prediction can be solved by for every 30 frames(for video, 30 frames would be 1 second), the prediction of fall or not fall could be decided by the mode of predictions of these 30 frames.

6 Annotated Code

Github repository:
https://github.com/YJZFlora/Fall_Detection_Deep_Learning_Model

Model Training:
(The code of training CNN model and LSTM model)
model_training_May08.ipynb
model_training_may08.py

Model Executing code:
Final model(ensembled model): execute_model_ensembled.py
LSTM model: execute_model_lstm.py
CNN model: execute_model_cnn.py

Other code:
Model ensembling and test accuracy of ensembled model:
test_acc_emsembled_model.py
test_acc_emsembled_model.ipynb

Plot the architecture of models: model_plot.py

Final model:
final_cnn_model.h5
final_lstm_model.h5

7 Instruction of Using Models

1. Clone this repository:
https://github.com/YJZFlora/Fall_Detection_Deep_Learning_Model

2. Turn to the directory:
```cd …/YJZFlora/Fall_Detection_Deep_Learning_Model```

3. Install necessary libraries:
* python 3
* Keras
```

* Tensorflow
* pandas
* Numpy

4. Generate body landmark json files, copy the directory of it.(or use the json files provided in sample directory

I used the open source tool by CMU-Perceptual-Computing-Lab(https://github.com/CMU-Perceptual-Computing-Lab/openpose) to generate body landmark.

Sample body landmark files have been included in samples directory.

eg, Fall_Detection_Deep_Learning_Model/samples/bodylandmark/ test_case1

5. Run one of the following scripts:

Execute new LSTM model:
```python3 execute_model_lstm.py <width of the video> <height of the video> <directory of body landmarks for a video> ```

Execute CNN model:
```python3 execute_model_cnn.py <width of the video> <height of the video> <directory of body landmarks for a video>```

Execute emsembled model:
``` python3 execute_model_ensembled.py <width of the video> <height of the video> <directory of body landmarks for a video>```

eg:
```python3 execute_model_ensembled.py 720 720 ./samples/bodylandmark/test_case1```

6. Result files

Ensembled model:
./results/timeLabel.json
./results/timeLabel.png

LSTM model:
./results/timeLabel_lstm.json
./results/timeLabel_lstm.png

CNN model:
./results/timeLabel_cnn.json
./results/timeLabel_cnn.png

And in each pair in the file, such as [2.721365,0.23445825932504438], the first number is the time, and the second number is the value of the label your binary-classification model predicts. So the above example shows that at 2.721365 second in the video, the label predicted by your binary-classification model changes to 0.23445825932504438.)

7. Video demo
https://youtu.be/r2CNC9QNPMg

8. Test samples
1)
Video: https://youtu.be/U8xwla8tQzA

Body landmark position:
https://github.com/YJZFlora/Fall_Detection_Deep_Learning_Model/tree/master/samples/bodylandmark/test_case1
Results:
https://github.com/YJZFlora/Fall_Detection_Deep_Learning_Model/tree/master/samples/test_case1

2)
Video: https://youtu.be/5KCNlL_NP18

Body landmark position:
https://github.com/YJZFlora/Fall_Detection_Deep_Learning_Model/tree/master/samples/bodylandmark/test_case2
Results:
https://github.com/YJZFlora/Fall_Detection_Deep_Learning_Model/tree/master/samples/test_case2

3)
Video: https://youtu.be/553VQJozd_c

Body landmark position:
https://github.com/YJZFlora/Fall_Detection_Deep_Learning_Model/tree/master/samples/bodylandmark/test_case3
Results:
https://github.com/YJZFlora/Fall_Detection_Deep_Learning_Model/tree/master/samples/test_case3

4)
Video: https://youtu.be/55pul8OFeMs

Body landmark position:
https://github.com/YJZFlora/Fall_Detection_Deep_Learning_Model/tree/master/samples/bodylandmark/test_case4
Results:
https://github.com/YJZFlora/Fall_Detection_Deep_Learning_Model/tree/master/samples/test_case4

5)
Video: https://youtu.be/3WK1YGByQDc
Body landmark position:

https://github.com/YJZFlora/Fall_Detection_Deep_Learning_Model/tree/master/samples/bodylandmark/test_case5
Results:
https://github.com/YJZFlora/Fall_Detection_Deep_Learning_Model/tree/master/samples/test_case5

Reference

[1] Bogdan Kwolek, Michal Kepski, Human fall detection on embedded platform using depth maps and wireless accelerometer, Computer Methods and Programs in Biomedicine, Volume 117, Issue 3, December 2014, Pages 489-501, ISSN 0169-2607.

[2] E. Auvinet, C. Rougier, J.Meunier, A. St-Arnaud, J. Rousseau, Multiple cameras fall dataset, Technical report 1350, DIRO - Université de Montréal, July 2010.

[3] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, Y. A. Sheikh, OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2019.

[4] Tomas Simon, Hanbyul Joo, Iain Matthews, Yaser Sheikh, Hand Keypoint Detection in Single Images using Multiview Bootstrapping, 2017.

[5] Zhe Cao, Tomas Simon, Shih-En Wei, Yaser Sheikh, Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields, 2017

[6] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, Yaser Sheikh, Convolutional pose machines, 2016.