Natural Network Project Report 3

April, 16, 2020

Yijun Zhang 629009140

(Improvements are marked red)

1. Topic

Falling Detection.

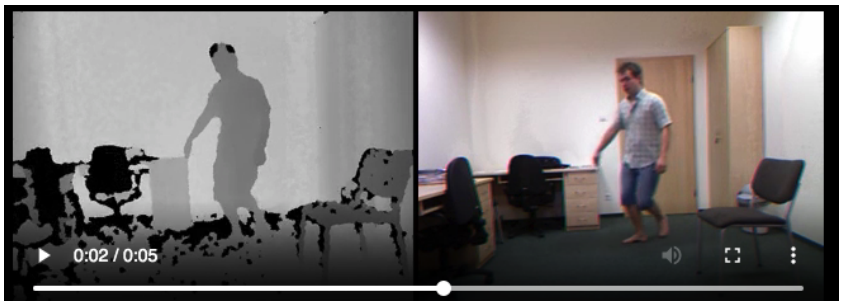
The figure shows how the probability of falling changes according to time of the video.

2. Datasets

2.1. Content of Datasets

1) UR Fall Detection Dataset: http://fenix.univ.rzeszow.pl/~mkepski/ds/uf.html

This dataset includes 30 videos of falling and 40 videos of daily activity(don't have falling).
Falling videos name like: "fall-01-cam0"
Daily activity videos like: "adl-01-cam0"



It also includes label file urfall-cam0-falls.csv and urfall-cam0-adls.csv .

Extracted features from depth maps are stored in CSV format. Each row contains one sample of data corresponding to one depth image. The columns from left to right are organized as follows:

- 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 adl videos, all frames are labeled 0 (not falling).

2) Multiple cameras fall dataset http://www.iro.umontreal.ca/~labimage/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 are 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 trimmed each video to only contain one person.

This dataset doesn't contain any label file. Therefore, I trimmed 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 marked all frames of the first part to be fall(1) and marked all frames of the second part to be not-fall(0).

2.2. Data cleaning and processing

(1) Produce body landmark

I used the open source tool 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.

(2) Read body landmark from json file.

The generated body landmarks for each frame contains 25 key points:

```
// Result for BODY_25 (25 body parts consisting of COCO + foot)
// const std::map<unsigned int, std::string> POSE_BODY_25_BODY_PARTS {
//     {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 detecting fall, these key points are unnecessary:
// {15, "REye"}, (right eye)
// {16, "LEye"}, (left eye)
// {17, "REar"}, (right ear)
// {18, "LEar"}, (left ear)

Therefore, when 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.

(3) Read and mark labels

For UR dataset, I read and matched label for body landmarks of each frame.
For Multiple Cameras Fall dataset, I marked the frame using loop.

(4) Up-sampling

There are 17938 frames are not fall(0) and 11261 frames of fall(1). To make the data balance, I used up-sampling. After up-sampling, both fall frames and not fall frames are 17938.

```
[ ]
# 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, fall vs not 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=17938)    # 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, fall vs not fall:
0    17938
1    11261
Name: label, dtype: int64
begin to re sample...
after re-sampling...
1    17938
0    17938
Name: label, dtype: int64
```

3. LSTM model

3.1. Input: Shape of Tensor

x_train shape is (31890, 1, 63)
x_test shape is (3986, 1, 63)

3.2. Output: Shape of Tensor

y_train shape is (31890,)
y_test shape is (3986,)

3.3. Shape of Output Tensor of Each layer

Output of first hidden layer: (31890, 32)
Output of output layer: (3986, 1)

3.4. Model improving

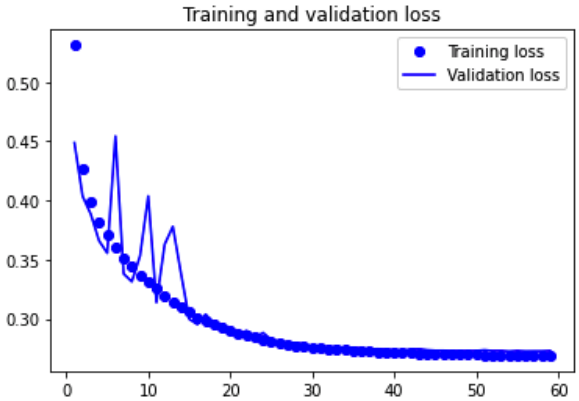
(1) Version 1

I used one hidden layer of LSTM and one dense layer as output layer.

```python
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)

x_train = array(x_train)
x_train = x_train.reshape((len(x_train), 1, len(x_train[0])))


model = Sequential()
model.add(LSTM(32, input_shape=(1, 63)))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(x_train, y_train,
                    epochs=100,
                    batch_size=32,
                    validation_split=0.2,
                    callbacks=[callbacks.EarlyStopping(monitor='val_loss', patience=5),
                    callbacks.LearningRateScheduler(scheduler)])

```

```python
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 32)                12288
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 12,321
Trainable params: 12,321
Non-trainable params: 0
_________________________________________________________________
```




```
2160/2160 [==============================] - 0s 84us/step
test loss:
0.283822970478623
test accuracy:
0.8685185185185185
```

(2) version 2

1) Epochs

I used learning rate decay(function scheduler), which means after the 10th epoch, learning rate will decrease gradually. I also use early stopping, which means when validation loss is not decrease for 5 epochs in a row.

2) Batch size

I tried different batch size and decided 16 is the best one. By choosing batch size 16, I improved the test accuracy from 0.86 to 0.88.

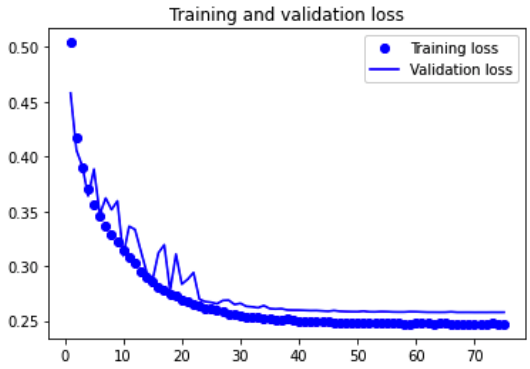
The performance of each batch size is showed as follows:

- Batch size: 16

Test score (test loss and test accuracy):

```
541/541 [==============================] - 1s 2ms/step - loss: 0.2491 - acc: 0.8883
[0.24909274280071259, 0.888271689414978]
```

Loss:

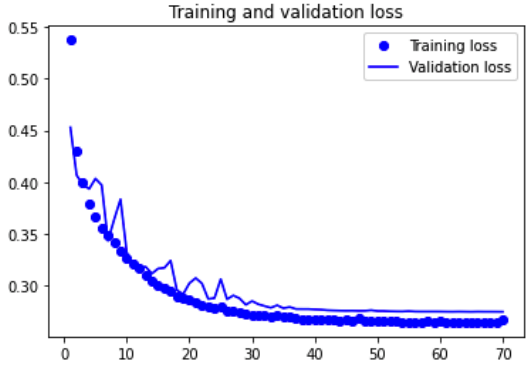


- Batch size: 32

Test score (test loss and test accuracy):

```
541/541 [==============================] - 1s 2ms/step - loss: 0.2668 - acc: 0.8789
[0.2668079137802124, 0.8788983225822449]
```

Loss:

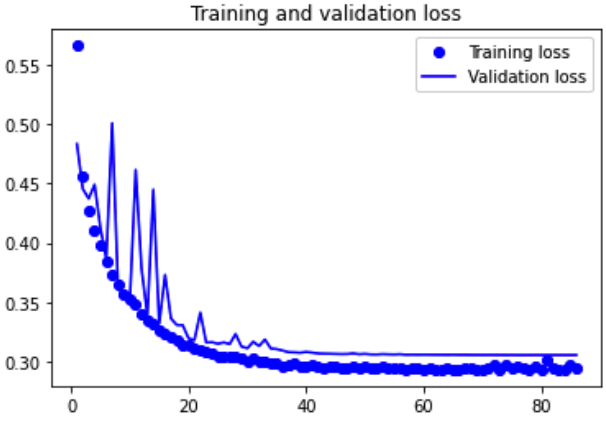


- Batch size: 64

Test score (test loss and test accuracy):

```
541/541 [==============================] - 1s 2ms/step - loss: 0.2962 - acc: 0.8624
[0.29618409276008606, 0.8623502850532532]
```

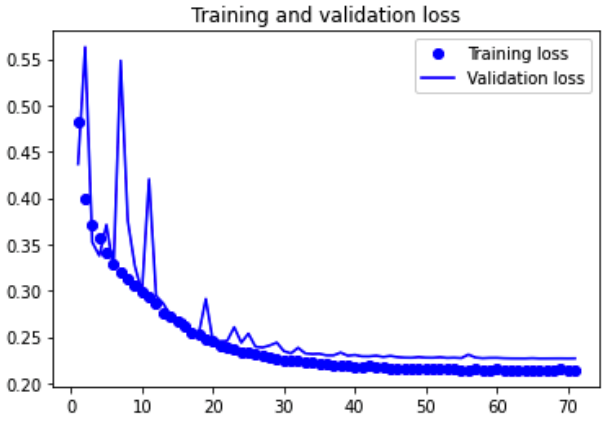
Loss:



3) Add one layer—stack recurrent layers

```
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)
```

The test accuracy improved from 0.88 to 0.90.

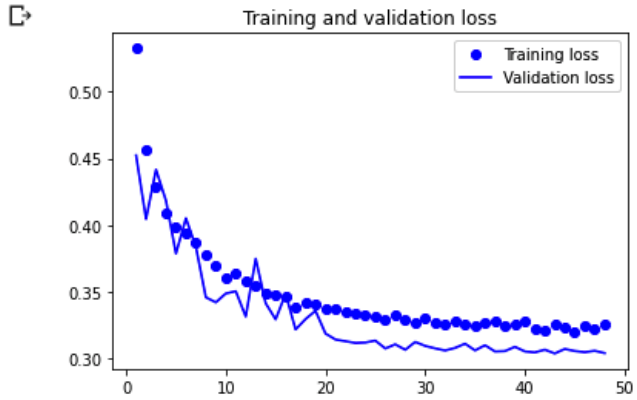



```
541/541 [==============================] - 1s 2ms/step - loss: 0.2164 - acc: 0.9047
[0.21643199026584625, 0.9047040343284607]
```

4) Drop out

Though the model doesn't show much overfitting, I tried to add *dropout = 0.1* and *recurrent_dropout = 0.5*.

```
27
28 # imrpove log: add one more lstm layer
29 model = Sequential()
30 model.add(LSTM(32,
31                input_shape=(1, 63),
32                return_sequences=True,
33                dropout = 0.1,
34                recurrent_dropout = 0.5))
35 model.add(LSTM(32,
36                dropout = 0.1,
37                recurrent_dropout = 0.5))
38 model.add(layers.Dense(1, activation='sigmoid'))
39 model.compile(optimizer='rmsprop',
40               loss='binary_crossentropy',
41               metrics=['acc'])
42 history = model.fit(x_train, y_train,
43                     epochs=100,
44                     batch_size=16,
45                     validation_split=0.2,
46                     callbacks=[callbacks.EarlyStopping(monitor='val_loss', patience=5),
47                     callbacks.LearningRateScheduler(scheduler)])
48
```

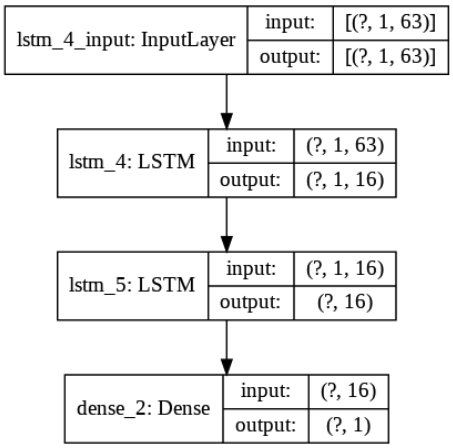



```
541/541 [==============================] - 2s 3ms/step - loss: 0.2884 - acc: 0.8676
[0.28841814398765564, 0.8676155805587769]
```

The performance does not improve, so I removed dropout.

(3) The final LSTM model

Architecture



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

Batch Size	16
Epochs	35
Dropout	(unnecessary)

Training and Testing performance of LSTM model are as follows:

```
Epoch 1/100
1595/1595 [==============================] - 14s 9ms/step - loss: 0.4169 - acc: 0.8105 - auc_3: 0.8322 - false_negatives_5: 1360.3173 - recall_3: 0.7113 - precision_3: 0.7575 - false_negatives_6: 1360.3173 -
Epoch 2/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.2995 - acc: 0.8744 - auc_3: 0.9095 - false_negatives_5: 2982.2815 - recall_3: 0.8654 - precision_3: 0.8103 - false_negatives_6: 2982.2815 -
Epoch 3/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.2745 - acc: 0.8876 - auc_3: 0.9276 - false_negatives_5: 4517.9707 - recall_3: 0.8818 - precision_3: 0.8343 - false_negatives_6: 4517.9707 -
Epoch 4/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.2592 - acc: 0.8948 - auc_3: 0.9353 - false_negatives_5: 5707.4170 - recall_3: 0.8947 - precision_3: 0.8406 - false_negatives_6: 5707.4170 -
Epoch 5/100
1595/1595 [==============================] - 14s 9ms/step - loss: 0.2441 - acc: 0.9029 - auc_3: 0.9415 - false_negatives_5: 6856.0483 - recall_3: 0.9024 - precision_3: 0.8488 - false_negatives_6: 6856.0483 -
Epoch 6/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.2345 - acc: 0.9071 - auc_3: 0.9461 - false_negatives_5: 7958.9785 - recall_3: 0.9077 - precision_3: 0.8557 - false_negatives_6: 7958.9785 -
Epoch 7/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.2260 - acc: 0.9103 - auc_3: 0.9496 - false_negatives_5: 9103.4736 - recall_3: 0.9109 - precision_3: 0.8620 - false_negatives_6: 9103.4736 -
Epoch 8/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.2187 - acc: 0.9122 - auc_3: 0.9522 - false_negatives_5: 10069.8535 - recall_3: 0.9148 - precision_3: 0.8651 - false_negatives_6: 10069.8535
Epoch 9/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.2133 - acc: 0.9147 - auc_3: 0.9545 - false_negatives_5: 11170.7559 - recall_3: 0.9167 - precision_3: 0.8692 - false_negatives_6: 11170.7559
Epoch 10/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.2075 - acc: 0.9169 - auc_3: 0.9566 - false_negatives_5: 12200.3477 - recall_3: 0.9187 - precision_3: 0.8726 - false_negatives_6: 12200.3477
Epoch 11/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.2024 - acc: 0.9187 - auc_3: 0.9582 - false_negatives_5: 13194.7031 - recall_3: 0.9206 - precision_3: 0.8749 - false_negatives_6: 13194.7031
Epoch 12/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1962 - acc: 0.9207 - auc_3: 0.9598 - false_negatives_5: 14187.0898 - recall_3: 0.9221 - precision_3: 0.8777 - false_negatives_6: 14187.0898
Epoch 13/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1921 - acc: 0.9216 - auc_3: 0.9608 - false_negatives_5: 15422.8535 - recall_3: 0.9221 - precision_3: 0.8800 - false_negatives_6: 15422.8535
Epoch 14/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1880 - acc: 0.9240 - auc_3: 0.9621 - false_negatives_5: 16366.5166 - recall_3: 0.9235 - precision_3: 0.8822 - false_negatives_6: 16366.5166
Epoch 15/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1849 - acc: 0.9244 - auc_3: 0.9633 - false_negatives_5: 17295.5527 - recall_3: 0.9248 - precision_3: 0.8841 - false_negatives_6: 17295.5527
Epoch 16/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1823 - acc: 0.9260 - auc_3: 0.9643 - false_negatives_5: 18223.9727 - recall_3: 0.9259 - precision_3: 0.8858 - false_negatives_6: 18223.9727
Epoch 17/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1795 - acc: 0.9267 - auc_3: 0.9653 - false_negatives_5: 19091.1992 - recall_3: 0.9271 - precision_3: 0.8873 - false_negatives_6: 19091.1992
Epoch 18/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1782 - acc: 0.9271 - auc_3: 0.9661 - false_negatives_5: 20009.6836 - recall_3: 0.9280 - precision_3: 0.8888 - false_negatives_6: 20009.6836
Epoch 19/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1756 - acc: 0.9291 - auc_3: 0.9669 - false_negatives_5: 20852.7480 - recall_3: 0.9291 - precision_3: 0.8900 - false_negatives_6: 20852.7480
Epoch 20/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1739 - acc: 0.9303 - auc_3: 0.9676 - false_negatives_5: 21701.3066 - recall_3: 0.9300 - precision_3: 0.8912 - false_negatives_6: 21701.3066
Epoch 21/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1723 - acc: 0.9291 - auc_3: 0.9683 - false_negatives_5: 22454.0996 - recall_3: 0.9311 - precision_3: 0.8920 - false_negatives_6: 22454.0996
Epoch 22/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1709 - acc: 0.9305 - auc_3: 0.9689 - false_negatives_5: 23309.6582 - recall_3: 0.9318 - precision_3: 0.8930 - false_negatives_6: 23309.6582
Epoch 23/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1699 - acc: 0.9316 - auc_3: 0.9694 - false_negatives_5: 24141.9629 - recall_3: 0.9325 - precision_3: 0.8940 - false_negatives_6: 24141.9629
Epoch 24/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1684 - acc: 0.9319 - auc_3: 0.9699 - false_negatives_5: 24926.3516 - recall_3: 0.9333 - precision_3: 0.8949 - false_negatives_6: 24926.3516
Epoch 25/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1681 - acc: 0.9324 - auc_3: 0.9704 - false_negatives_5: 25709.4727 - recall_3: 0.9340 - precision_3: 0.8957 - false_negatives_6: 25709.4727
Epoch 26/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1666 - acc: 0.9316 - auc_3: 0.9709 - false_negatives_5: 26536.3105 - recall_3: 0.9346 - precision_3: 0.8965 - false_negatives_6: 26536.3105
Epoch 27/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1661 - acc: 0.9323 - auc_3: 0.9713 - false_negatives_5: 27330.3730 - recall_3: 0.9352 - precision_3: 0.8973 - false_negatives_6: 27330.3730
Epoch 28/100
1595/1595 [==============================] - 14s 9ms/step - loss: 0.1652 - acc: 0.9330 - auc_3: 0.9717 - false_negatives_5: 28094.6660 - recall_3: 0.9358 - precision_3: 0.8979 - false_negatives_6: 28094.6660
Epoch 29/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1648 - acc: 0.9333 - auc_3: 0.9721 - false_negatives_5: 28854.4004 - recall_3: 0.9364 - precision_3: 0.8986 - false_negatives_6: 28854.4004
Epoch 30/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1637 - acc: 0.9335 - auc_3: 0.9724 - false_negatives_5: 29635.8730 - recall_3: 0.9369 - precision_3: 0.8992 - false_negatives_6: 29635.8730
Epoch 31/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1636 - acc: 0.9335 - auc_3: 0.9728 - false_negatives_5: 30400.5410 - recall_3: 0.9374 - precision_3: 0.8998 - false_negatives_6: 30400.5410
Epoch 32/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1633 - acc: 0.9340 - auc_3: 0.9731 - false_negatives_5: 31178.7793 - recall_3: 0.9379 - precision_3: 0.9003 - false_negatives_6: 31178.7793
Epoch 33/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1626 - acc: 0.9336 - auc_3: 0.9734 - false_negatives_5: 31959.4082 - recall_3: 0.9383 - precision_3: 0.9009 - false_negatives_6: 31959.4082
Epoch 34/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1624 - acc: 0.9344 - auc_3: 0.9737 - false_negatives_5: 32716.5645 - recall_3: 0.9387 - precision_3: 0.9014 - false_negatives_6: 32716.5645
Epoch 35/100
1595/1595 [==============================] - 13s 8ms/step - loss: 0.1618 - acc: 0.9346 - auc_3: 0.9739 - false_negatives_5: 33465.4062 - recall_3: 0.9391 - precision_3: 0.9018 - false_negatives_6: 33465.4062
finish training lstm model
```

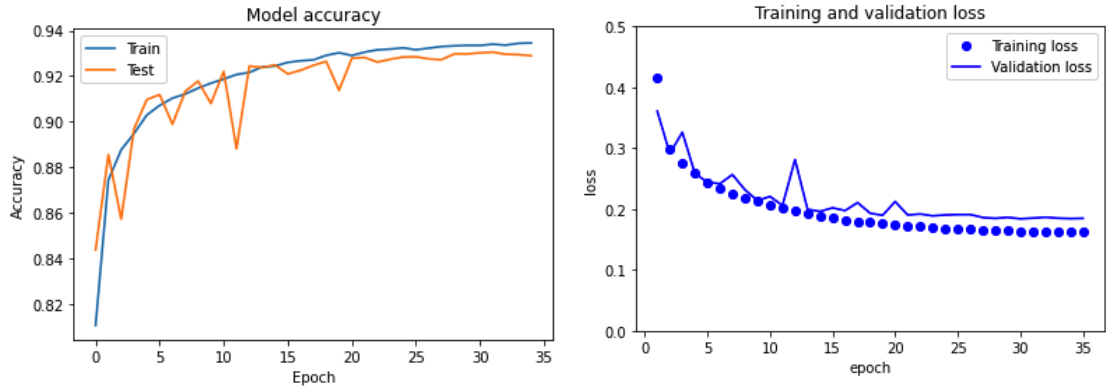
```
[0.16936977207660675, 0.9312009811401367, 0.9721036553382874, 26913.107421875, 0.9362621307373047, 0.8967313766479492, 26913.107421875, 377305.71875, 45530.30859375, 395358.34375, array([0.66638786], dtype=float32)
```

Testing Loss	0.16936977207660675
Testing Accuracy	0.9312009811401367
AUC	0.9721
Sensitivity /Recall $(Sensitivity = \frac{TP}{TP + FN})$	0.9362660472
Specificity $(Specificity = \frac{TN}{TN + FP})$	0.8923220642

Model Summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 1, 16)             5120
_________________________________________________________________
lstm_5 (LSTM)                (None, 16)                2112
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 17
=================================================================
Total params: 7,249
Trainable params: 7,249
Non-trainable params: 0
_________________________________________________________________
```

Model plotting



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")
```

4. CNN model

4.1. Input: Shape of Tensor

x_train shape is (31890, 3, 21,1)
x_test shape is (3986, 3, 21,1)

4.2. Output: Shape of Tensor

y_train shape is (31890,)
y_test shape is (3986,)

4.3. Shape of Output Tensor of Each layer

Output of first hidden layer: (31890, 32)
Output of output layer: (31890, 1)

4.4 Model improving log

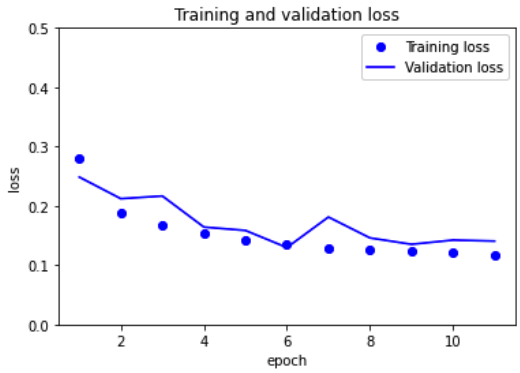
(1) version 1

I tried 2D CNN to see if it is better than LSTM.

```
[ ]
# solution 2: train model using CNN
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, MaxPooling2D

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)

x_train = array(x_train)
x_train = x_train.reshape((len(x_train), 3, int(len(x_train[0])/3), 1))

y_train = array(y_train)
```

```
23
24 #create model
25 model = Sequential()
26 model.add(Conv2D(64,
27                  kernel_size=3,
28                  activation='relu',
29                  input_shape=(3,21,1),
30                  padding='same'))
31 model.add(Conv2D(32,
32                  kernel_size=3,
33                  activation='relu',
34                  padding='same'))
35 model.add(MaxPooling2D(2,2))
36 model.add(Flatten())
37 model.add(Dense(128, activation = 'relu'))
38 model.add(Dense(1, activation='sigmoid'))
39 # compile and fit
40 model.compile(optimizer='rmsprop',
41               loss='binary_crossentropy',
42               metrics=['acc'])
43 history = model.fit(x_train, y_train,
44                     epochs=100,
45                     batch_size=16,
46                     validation_split=0.2,
47                     callbacks=[callbacks.EarlyStopping(monitor='val_loss', patience=5),
48                     callbacks.LearningRateScheduler(scheduler)])
49
```

```
Epoch 1/100
1595/1595 [==============================] - 8s 5ms/step - loss: 0.2790 - acc: 0.8816 - val_loss: 0.2485 - val_acc: 0.8935 - lr: 0.0010
Epoch 2/100
1595/1595 [==============================] - 7s 5ms/step - loss: 0.1883 - acc: 0.9253 - val_loss: 0.2121 - val_acc: 0.9169 - lr: 0.0010
Epoch 3/100
1595/1595 [==============================] - 7s 5ms/step - loss: 0.1668 - acc: 0.9336 - val_loss: 0.2166 - val_acc: 0.9219 - lr: 0.0010
Epoch 4/100
1595/1595 [==============================] - 8s 5ms/step - loss: 0.1526 - acc: 0.9392 - val_loss: 0.1642 - val_acc: 0.9363 - lr: 0.0010
Epoch 5/100
1595/1595 [==============================] - 7s 5ms/step - loss: 0.1412 - acc: 0.9423 - val_loss: 0.1585 - val_acc: 0.9454 - lr: 0.0010
Epoch 6/100
1595/1595 [==============================] - 7s 5ms/step - loss: 0.1342 - acc: 0.9445 - val_loss: 0.1297 - val_acc: 0.9483 - lr: 0.0010
Epoch 7/100
1595/1595 [==============================] - 7s 5ms/step - loss: 0.1292 - acc: 0.9501 - val_loss: 0.1811 - val_acc: 0.9389 - lr: 0.0010
Epoch 8/100
1595/1595 [==============================] - 7s 5ms/step - loss: 0.1260 - acc: 0.9496 - val_loss: 0.1459 - val_acc: 0.9429 - lr: 0.0010
Epoch 9/100
1595/1595 [==============================] - 7s 5ms/step - loss: 0.1226 - acc: 0.9513 - val_loss: 0.1352 - val_acc: 0.9484 - lr: 0.0010
Epoch 10/100
1595/1595 [==============================] - 7s 5ms/step - loss: 0.1209 - acc: 0.9532 - val_loss: 0.1423 - val_acc: 0.9475 - lr: 0.0010
Epoch 11/100
1595/1595 [==============================] - 7s 5ms/step - loss: 0.1168 - acc: 0.9537 - val_loss: 0.1406 - val_acc: 0.9505 - lr: 0.0010
```




```
[38]  1 model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 3, 21, 64)         640
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 3, 21, 32)         18464
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 1, 10, 32)         0
_________________________________________________________________
flatten (Flatten)            (None, 320)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               41088
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
=================================================================
Total params: 60,321
Trainable params: 60,321
Non-trainable params: 0
_________________________________________________________________
```

Test score of CNN 2D is better than LSTM.
The test accuracy is 0.95, better than 0.9 of LSTM.

```
# train the final model on all non-test data avaliable
# for cnn 2d
x_test = []
y_test = []

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

x_test = array(x_test)
x_test = x_test.reshape((len(x_test), 3, int(len(x_test[0])/3), 1))

y_test = array(y_test)

test_score = model.evaluate(x_test, y_test)
print(test_score)
```

```
997/997 [==============================] - 2s 2ms/step - loss: 0.1164 - acc: 0.9559
[0.11636932939291, 0.9558795690536499]
```

(2) version 2

Improvement of CNN model:

1. I added batch normalization between 2 convolution layers to adaptively normalize data as the mean and variance change over time during training. (line 32, 37)

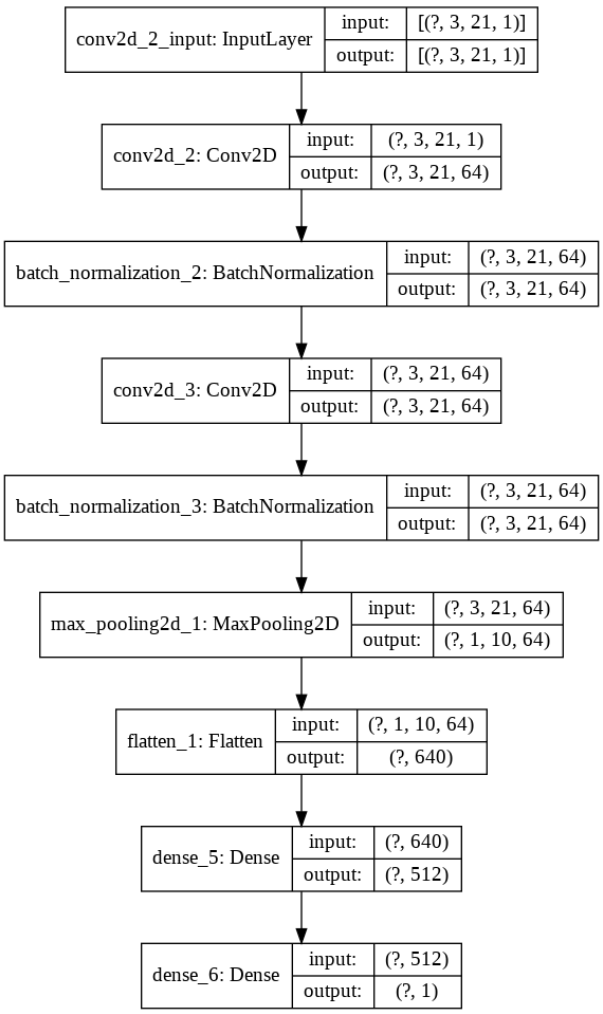
2. I changed optimizer to Adam. (line 44)

3. I changed number of units in the dense layer from 128 to 512. (line 40)

```
def tain_cnn_model(x_train, y_train):
  x_train = array(x_train)
  x_train = x_train.reshape((len(x_train), 3, int(len(x_train[0])/3), 1))

  y_train = array(y_train)

  #create model
  cnn_model = Sequential()
  cnn_model.add(Conv2D(64,
                  kernel_size=3,
                  activation='relu',
                  input_shape=(3,21,1),
                  padding='same'))
  cnn_model.add(layers.BatchNormalization(1))
  cnn_model.add(Conv2D(64,
                  kernel_size=3,
                  activation='relu',
                  padding='same'))
  cnn_model.add(layers.BatchNormalization(1))
  cnn_model.add(MaxPooling2D(2,2))
  cnn_model.add(Flatten())
  cnn_model.add(Dense(512, activation = 'relu'))
  cnn_model.add(Dense(1, activation='sigmoid'))

  # compile and fit
  cnn_model.compile(optimizer='Adam',
                loss='binary_crossentropy',
                metrics=['acc',
                        metrics.AUC(),
                        metrics.FalseNegatives(),
                        metrics.Recall(),
                        metrics.Precision(),
                        metrics.FalseNegatives(),
                        metrics.TrueNegatives(),
                        metrics.FalsePositives(),
                        metrics.TruePositives()])
  cnn_history = cnn_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)])

tain_cnn_model(x_train, y_train)
```

(3) The final CNN model

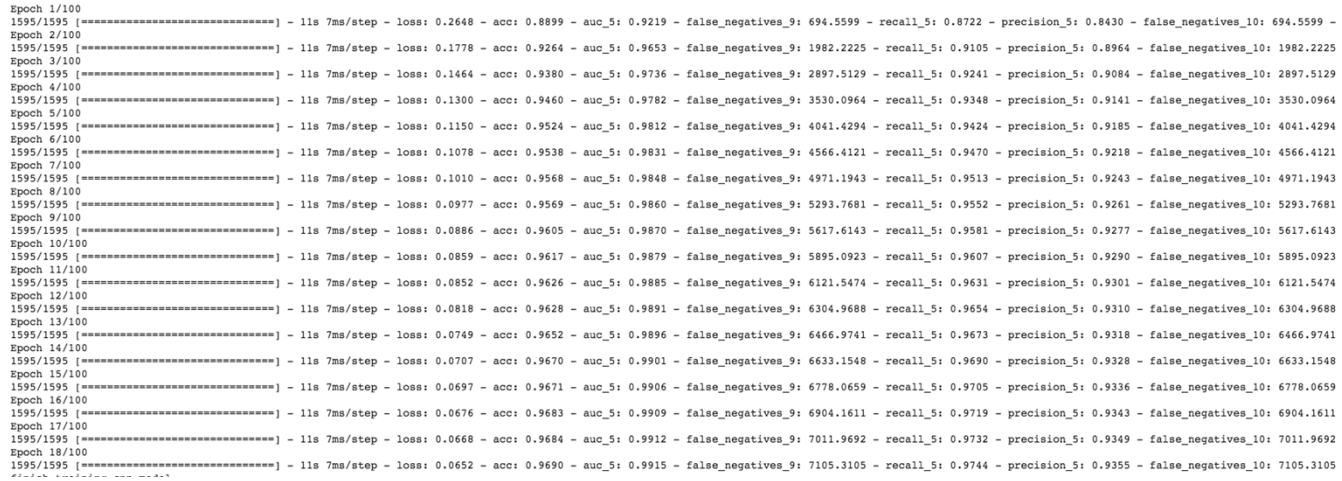
Architecture

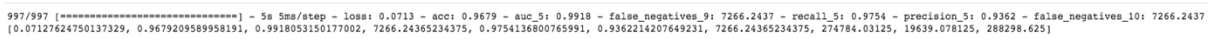


According to the above discussion, Hyperparameters of CNN model are as follows:

Batch Size	16
Epochs	18
Dropout	(unnecessary)

Training and Testing performance of CNN model are as follows:





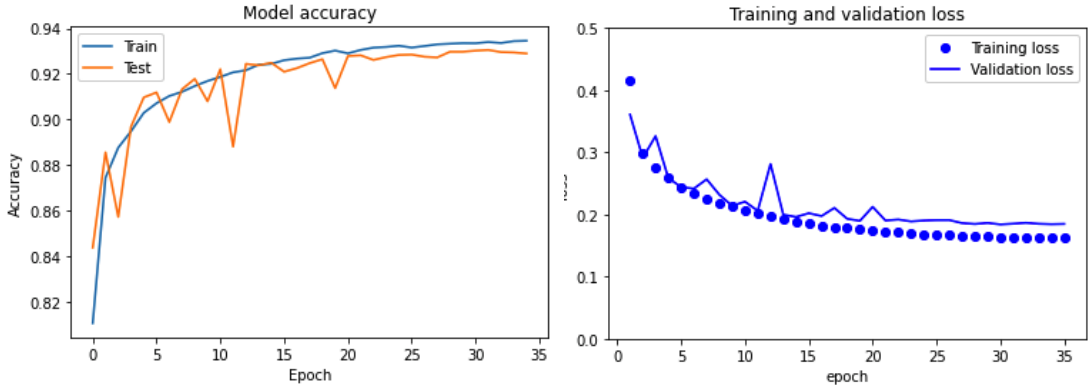
Testing Loss	0.0713
Testing Accuracy	0.9679
AUC	0.9918
Sensitivity / Recall $(Sensitivity = \frac{TP}{TP + FN})$	0.9754
Specificity $(Specificity = \frac{TN}{TN + FP})$	0.9333

Model Summary

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

Model plotting



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

5. Model ensembling

The LSTM model and CNN model are good models and they prediction from different aspects. I emsemble these two models.

```
prediction = []

for i in range(len(cnn_probality)):
  p = cnn_probality[i] * 0.5 + lstm_probality[i] * 0.5

```

```
prediction = []

for i in range(len(cnn_probality)):
  p = cnn_probality[i] * 0.5 + lstm_probality[i]

```

6. Fail case analysis

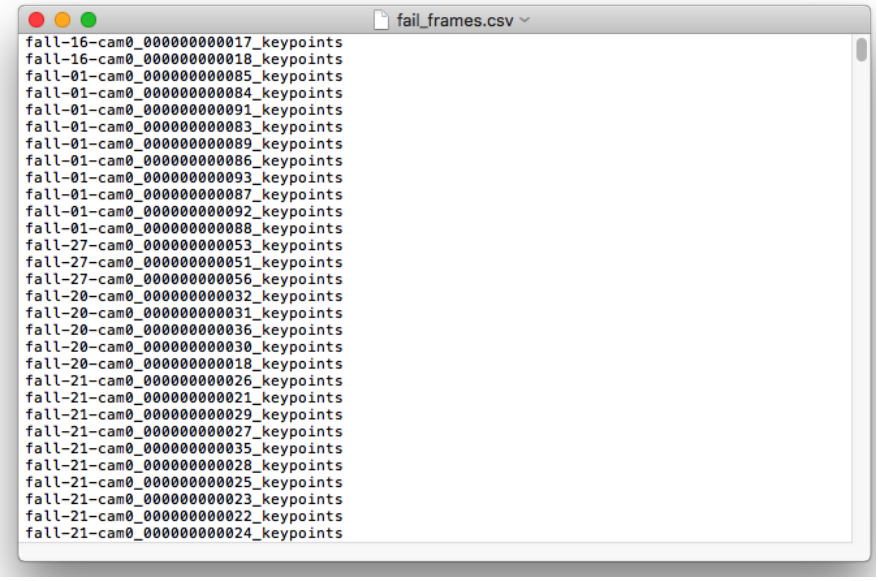
```
# prepare data to detect and analyse fail cases of the final models
sample_data_for_fail = []
for i in range(len(all_data)):
  toAdd = []
  for record in all_data[i]:
    toAdd.append(record)
  toAdd.append(frame_name[i])
  sample_data_for_fail.append(toAdd)

print(sample_data_for_fail[0])
```

```
# get wrong predictions(fail cases) of cnn model
from numpy import array
import tensorflow as tf

sample_data_for_fail

cnn_model = tf.keras.models.load_model('/content/drive/My Drive/636project/model_improved_cnn.h5')

x_sample = []
y_sample = []

for record in sample_data_for_fail:
  x_sample.append(record[1:-1])
  label = int(record[0])
  y_sample.append(label)

x_sample = array(x_sample)
x_sample = x_sample.reshape((len(x_sample), 3, int(len(x_sample[0])/3), 1))

y_sample = array(y_sample)

probality = cnn_model.predict(x_sample)

fails = []
fn = []
fp = []

for i in range(len(probality)):
  if probality[i] >= 0.5:
    x = 1
  else:
    x = 0
  if x != y_sample[i]:
    fails.append(i)
    if x == 1:
      fn.append(i)
    else:
      fp.append(i)

print("number of fail cases of CNN model:", len(fails))

```

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

for fail in fails:
  fail_frames.append(sample_data_for_fail[fail][-1])

for fail in fp:
  false_positive_frames.append(sample_data_for_fail[fail][-1])


for fail in fn:
  false_negative_frames.append(sample_data_for_fail[fail][-1])

print(fail_frames[0])
print(len(false_positive_frames))
print(len(false_negative_frames))
```

```python
df_fail_frames = pd.DataFrame.from_records(fail_frames)
df_fail_frames.to_csv (r'/content/drive/My Drive/636project/fail_frames.csv', index = False, header=False)
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)

```

For a total of 29199 frames, there are 1137 fail cases. In these fails frames, there are 974 false-positive frames(actual fall but predicts not fall) 163 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 the frame.



I found that for every 30 frames of each video, there are less than 15 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.

By checking the image of fail video frames. I found that there are three main reasons of fail prediction.

First, the uncertainty of falling action.

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 falling 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. The uncertainty of falling action will cause the model cannot reach 100% accuracy of prediction.

video 59 from Multiple Camera Dataset
left: marked as fall; right: 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. If there are enough video data of different kinds of fall and confusion actions, my training structure will get a more precise model.

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

For the above reasons, I will try to find more data of fall in the upcoming improvement.

7. Annotated Code

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

Training code:
636model_improved_Apr16.py

Execute code:
LSTM model: execute_model_lstm.py
CNN model: execute_model_cnn.py

8. Instruction on how to test trained model

Instruction:
https://github.com/YJZFlora/Fall_Detection_Deep_Learning_Model/blob/master/README.md

8.1. Install Dependencies

- python 3
- Keras
- Tensorflow
- pandas
- Numpy

8.2. Execution

Execute new LSTM model:

```
python3 execute_model_lstm.py <directory of body landmarks for a video>
```

Execute CNN model:

```
python3 execute_model_cnn.py <directory of body landmarks for a video>
```

video demo: https://www.youtube.com/watch?v=r2CNC9QNPMg

video demo: https://www.youtube.com/watch?v=r2CNC9QNPMg

8.3. Video Link

https://www.youtube.com/playlist?list=PLYkX3-wtcreE-4tlAK8zQ4hFrgpoR_OUC

9. Future improvement

9.1. Improve test data

Currently, the test data come from the same datasets of the training data. In the next improvement, I will use video data from other datasets to see the test score. Then analyze the fail case and further improve the model.