“Cry” Detection Report

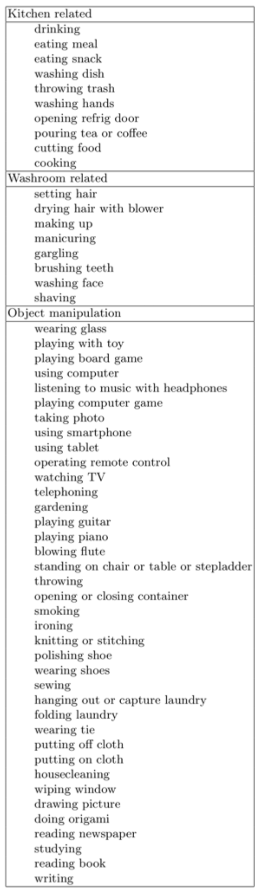
May 23 2020

1. Topic

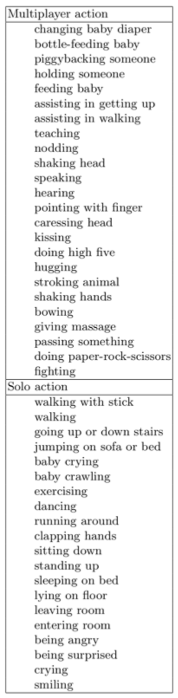
My topic is “crying”.

2. Dataset

I replaced my previous dataset, which is self-crawled and consists of 151433 human facial images extracted from over 200 videos from Youtube, with a part of the STAIR actions dataset (https://actions.stair.center/). STAIR Actions is a video dataset consisting of 100 everyday human action categories. Each category contains around 900 to 1800 trimmed video clips. Each clip lasts 5 to 6 seconds. Clips are taken from YouTube video or made by crowdsource workers.



Kitchen related
drinking
eating meal
eating snack
washing dish
throwing trash
washing hands
opening refrig door
pouring tea or coffee
cutting food
cooking
Washroom related
setting hair
drying hair with blower
making up
manicuring
gargling
brushing teeth
washing face
shaving
Object manipulation
wearing glass
playing with toy
playing board game
using computer
listening to music with headphones
playing computer game
taking photo
using smartphone
using tablet
operating remote control
watching TV
telephoning
gardening
playing guitar
playing piano
blowing flute
standing on chair or table or stepladder
throwing
opening or closing container
smoking
ironing
knitting or stitching
polishing shoe
wearing shoes
sewing
hanging out or capture laundry
folding laundry
wearing tie
putting off cloth
putting on cloth
housecleaning
wiping window
drawing picture
doing origami
reading newspaper
studying
reading book
writing



Multiplayer action
changing baby diaper
bottle-feeding baby
piggybacking someone
holding someone
feeding baby
assisting in getting up
assisting in walking
teaching
nodding
shaking head
speaking
hearing
pointing with finger
caressing head
kissing
doing high five
hugging
stroking animal
shaking hands
bowing
giving massage
passing something
doing paper-rock-scissors
fighting
Solo action
walking with stick
walking
going up or down stairs
jumping on sofa or bed
baby crying
baby crawling
exercising
dancing
running around
clapping hands
sitting down
standing up
sleeping on bed
lying on floor
leaving room
entering room
being angry
being surprised
crying
smiling

I downloaded 1890 video clips from the STAIRS actions dataset, including all crying video clips and an equal amount of negative ones which cover almost all the actions shown in the above picture. I picked 1393 qualified videos from the downloaded ones and extract the face, hand and pose landmarks from the videos using OpenPose. In this way I created a dataset of 177925 rows of face, hand and pose data. After balancing positives and negatives, the size of the dataset became 149102, consisting of 74665 positives and 74437 negatives.

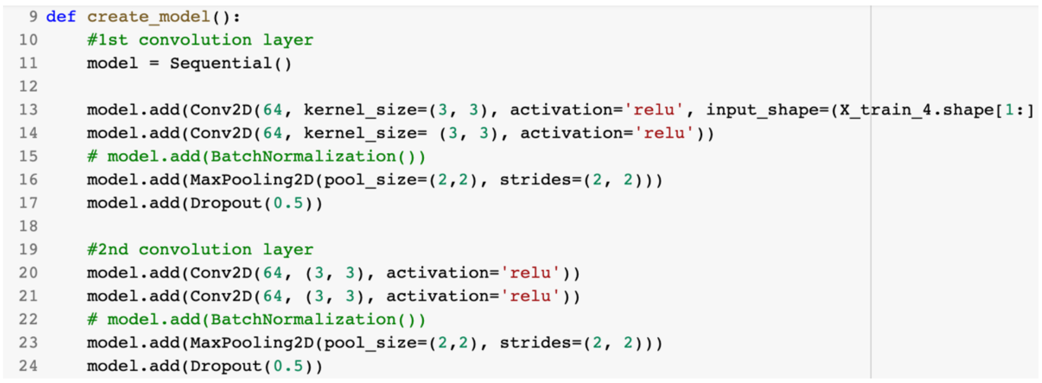
I also created 2 more dataset based on my original dataset, one consisting of pose data only and one consisting of face and hand data only. The pose dataset is of size 82323, consisting of 41165 positives and 41158 negatives. The face and hand dataset is of size 26686, with 16551 positives and 15124 negatives.

3. Model

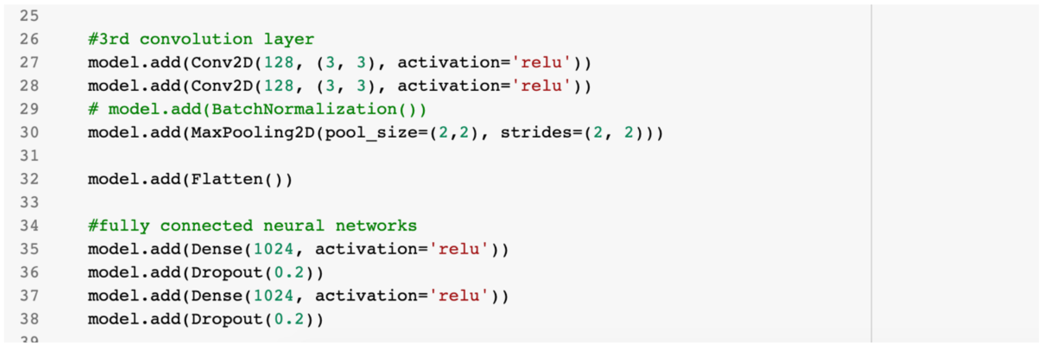
I tried 3 models selected from my previous models and used different data to train these models. At first, I used face-pose-hand data, but the performance is not good. I think the reason is that few positive videos record the whole body of a person, and they either record people above shoulder or the whole body with face and hands difficult to be recognized by OpenPose. Also, if the video dose not contain elbows, the hand data will not be extracted by OpenPose. So, the data extracted contain many 0s, which may disrupt the model. Therefore, I created 2 more dataset based on my original dataset, one consisting of pose data only and one consisting of face and hand data only.

1. 2D CNN model (trained with face-pose-hand data)

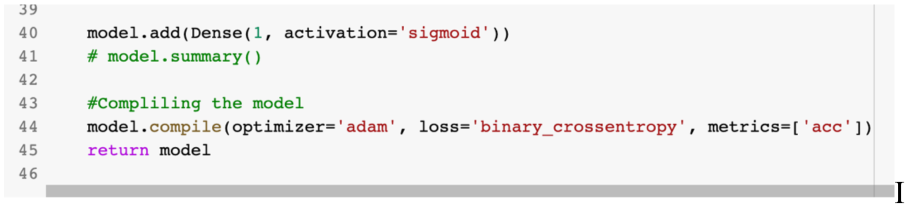
This model has 3 convolutional layers, 2 fully connected layers and 1 binary classification layer. I take every 10 frames as a timestep. My origin input data is of size (149102, 10, 274), in order to fit it into this model, I deleted 24 columns that has the most 0s and dropped several samples that has too many 0s, and get an input of size (126065, 10, 250), and then reshaped it into (126065, 50, 50, 1).



```
def create_model():
    #1st convolution layer
    model = Sequential()

    model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape=(X_train_4.shape[1:]
    model.add(Conv2D(64, kernel_size= (3, 3), activation='relu'))
    # model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
    model.add(Dropout(0.5))

    #2nd convolution layer
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    # model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
    model.add(Dropout(0.5))
```

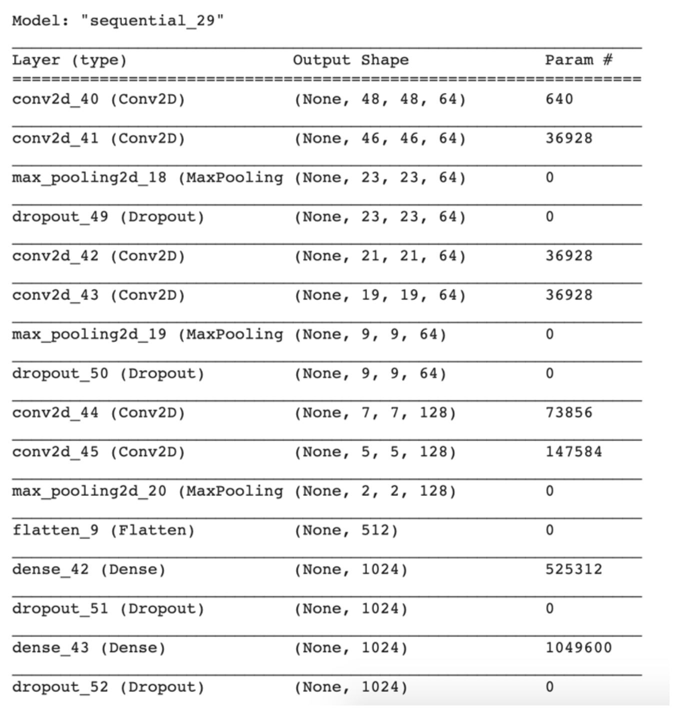


```
    #3rd convolution layer
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    # model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))

    model.add(Flatten())

    #fully connected neural networks
    model.add(Dense(1024, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(1024, activation='relu'))
    model.add(Dropout(0.2))
```

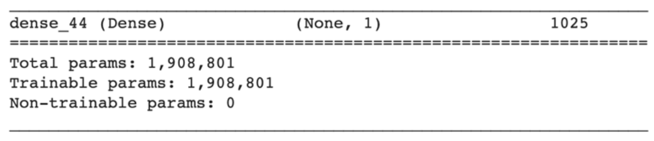


```

    model.add(Dense(1, activation='sigmoid'))
    # model.summary()

    #Compliling the model
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
    return model

```

Model Summary:



```
Model: "sequential_29"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_40 (Conv2D)           (None, 48, 48, 64)        640
_________________________________________________________________
conv2d_41 (Conv2D)           (None, 46, 46, 64)        36928
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 23, 23, 64)        0
_________________________________________________________________
dropout_49 (Dropout)         (None, 23, 23, 64)        0
_________________________________________________________________
conv2d_42 (Conv2D)           (None, 21, 21, 64)        36928
_________________________________________________________________
conv2d_43 (Conv2D)           (None, 19, 19, 64)        36928
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 9, 9, 64)          0
_________________________________________________________________
dropout_50 (Dropout)         (None, 9, 9, 64)          0
_________________________________________________________________
conv2d_44 (Conv2D)           (None, 7, 7, 128)         73856
_________________________________________________________________
conv2d_45 (Conv2D)           (None, 5, 5, 128)         147584
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 2, 2, 128)         0
_________________________________________________________________
flatten_9 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_42 (Dense)             (None, 1024)              525312
_________________________________________________________________
dropout_51 (Dropout)         (None, 1024)              0
_________________________________________________________________
dense_43 (Dense)             (None, 1024)              1049600
_________________________________________________________________
dropout_52 (Dropout)         (None, 1024)              0
```

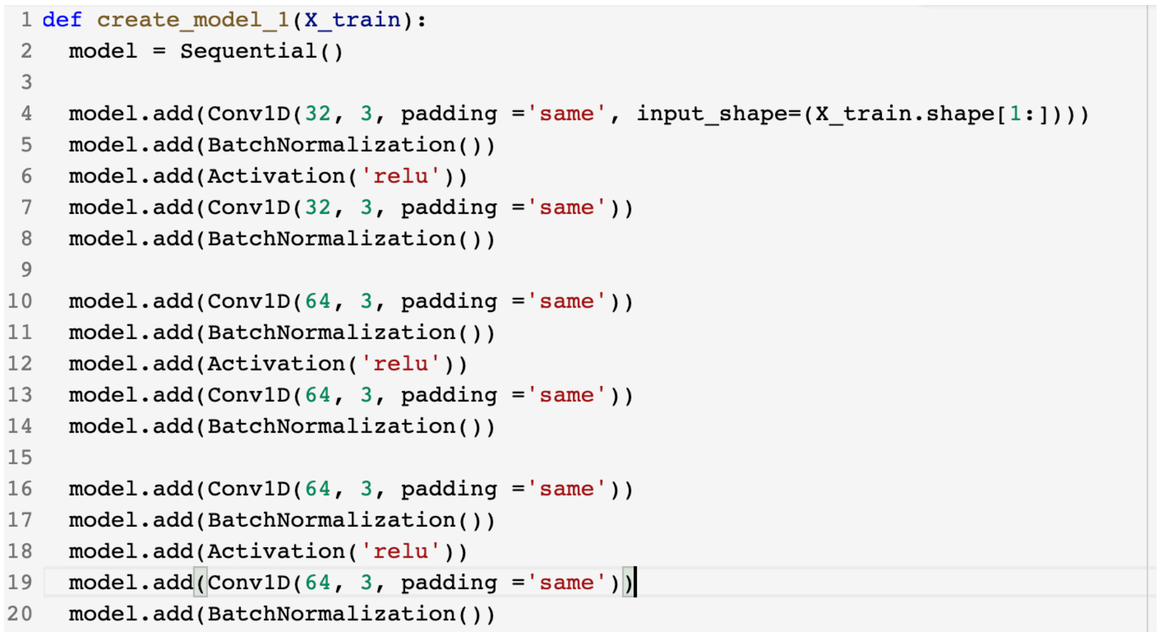


```
_________________________________________________________________
dense_44 (Dense)             (None, 1)                 1025
=================================================================
Total params: 1,908,801
Trainable params: 1,908,801
Non-trainable params: 0
_________________________________________________________________
```

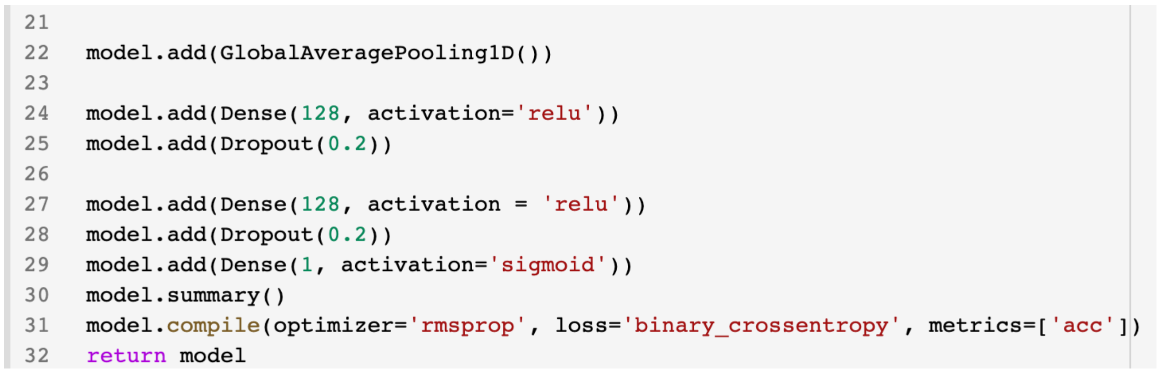
2. 1D CNN model (trained with pose data)

This model has 3 convolutional layers, 2 fully connected layers and 1 binary classification layer.

I tried only pose data to train this model. I deleted several samples that has more than 124 0s, and then take every 15 frames as a timestep, and created an input of size (82323, 15, 50).

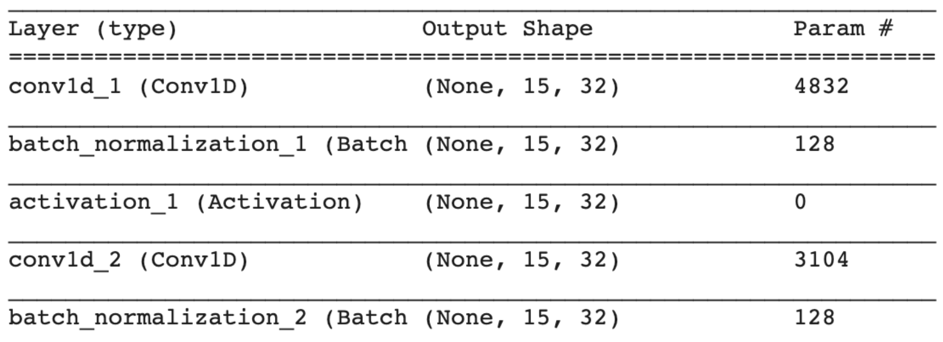


```python
def create_model_1(X_train):
  model = Sequential()

  model.add(Conv1D(32, 3, padding ='same', input_shape=(X_train.shape[1:])))
  model.add(BatchNormalization())
  model.add(Activation('relu'))
  model.add(Conv1D(32, 3, padding ='same'))
  model.add(BatchNormalization())

  model.add(Conv1D(64, 3, padding ='same'))
  model.add(BatchNormalization())
  model.add(Activation('relu'))
  model.add(Conv1D(64, 3, padding ='same'))
  model.add(BatchNormalization())

  model.add(Conv1D(64, 3, padding ='same'))
  model.add(BatchNormalization())
  model.add(Activation('relu'))
  model.add(Conv1D(64, 3, padding ='same'))
  model.add(BatchNormalization())
```

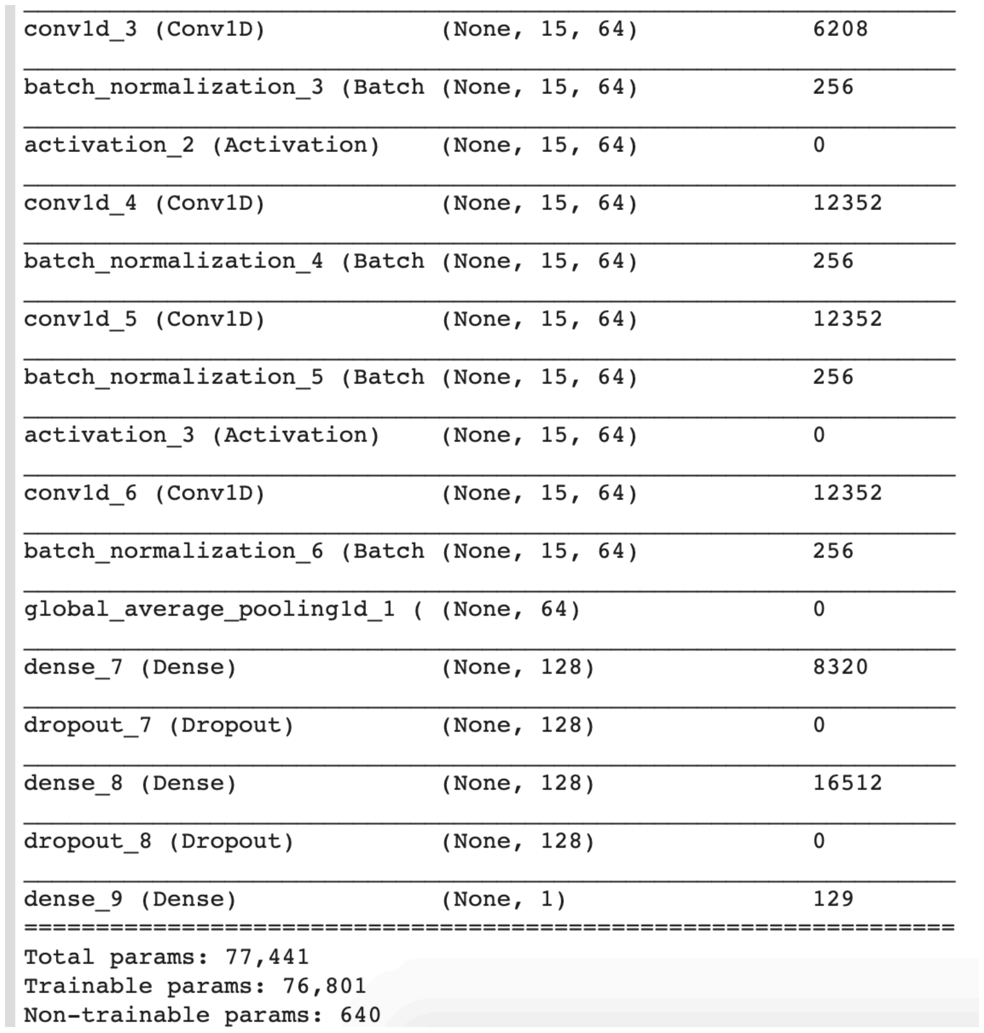


```python

  model.add(GlobalAveragePooling1D())

  model.add(Dense(128, activation='relu'))
  model.add(Dropout(0.2))

  model.add(Dense(128, activation = 'relu'))
  model.add(Dropout(0.2))
  model.add(Dense(1, activation='sigmoid'))
  model.summary()
  model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
  return model
```

Model Summary:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 15, 32)            4832
_________________________________________________________________
batch_normalization_1 (Batch (None, 15, 32)            128
_________________________________________________________________
activation_1 (Activation)    (None, 15, 32)            0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 15, 32)            3104
_________________________________________________________________
batch_normalization_2 (Batch (None, 15, 32)            128
```

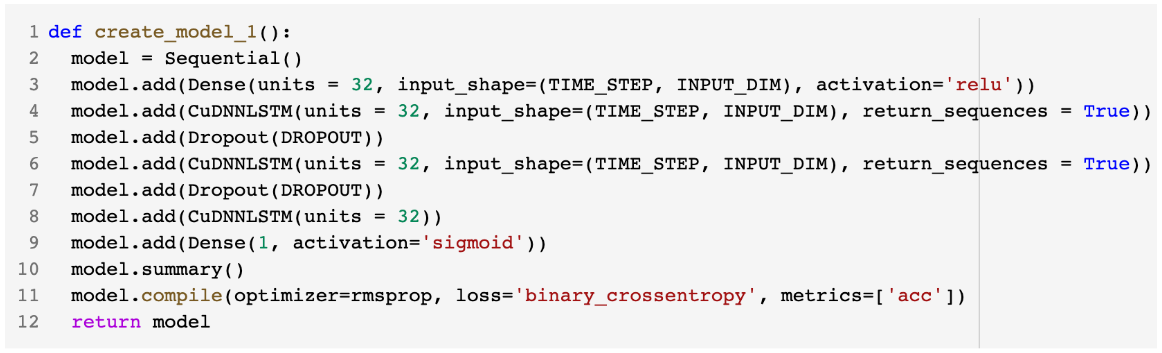


```
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 15, 64)            6208      
_________________________________________________________________
batch_normalization_3 (Batch (None, 15, 64)            256       
_________________________________________________________________
activation_2 (Activation)    (None, 15, 64)            0         
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 15, 64)            12352     
_________________________________________________________________
batch_normalization_4 (Batch (None, 15, 64)            256       
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 15, 64)            12352     
_________________________________________________________________
batch_normalization_5 (Batch (None, 15, 64)            256       
_________________________________________________________________
activation_3 (Activation)    (None, 15, 64)            0         
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 15, 64)            12352     
_________________________________________________________________
batch_normalization_6 (Batch (None, 15, 64)            256       
_________________________________________________________________
global_average_pooling1d_1 ( (None, 64)                0         
_________________________________________________________________
dense_7 (Dense)              (None, 128)               8320      
_________________________________________________________________
dropout_7 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_8 (Dense)              (None, 128)               16512     
_________________________________________________________________
dropout_8 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 129       
=================================================================
Total params: 77,441
Trainable params: 76,801
Non-trainable params: 640
```

3. LSTM model (trained with face-hand data)

   This model has 3 LSTM layers and 1 binary classification layer.

   I tried face and hand data to train this model. I deleted several samples that has more than 84 0s (which means no hand data), and then take every 20 frames as a timestep, and created an input of size (26686, 20, 134).



```python
def create_model_1():
  model = Sequential()
  model.add(Dense(units = 32, input_shape=(TIME_STEP, INPUT_DIM), activation='relu'))
  model.add(CuDNNLSTM(units = 32, input_shape=(TIME_STEP, INPUT_DIM), return_sequences = True))
  model.add(Dropout(DROPOUT))
  model.add(CuDNNLSTM(units = 32, input_shape=(TIME_STEP, INPUT_DIM), return_sequences = True))
  model.add(Dropout(DROPOUT))
  model.add(CuDNNLSTM(units = 32))
  model.add(Dense(1, activation='sigmoid'))
  model.summary()
  model.compile(optimizer=rmsprop, loss='binary_crossentropy', metrics=['acc'])
  return model
```

Model Summary:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_7 (Dense)              (None, 20, 32)            7200      
_________________________________________________________________
cu_dnnlstm_10 (CuDNNLSTM)    (None, 20, 32)            8448      
_________________________________________________________________
dropout_7 (Dropout)          (None, 20, 32)            0         
_________________________________________________________________
cu_dnnlstm_11 (CuDNNLSTM)    (None, 20, 32)            8448      
_________________________________________________________________
dropout_8 (Dropout)          (None, 20, 32)            0         
_________________________________________________________________
cu_dnnlstm_12 (CuDNNLSTM)    (None, 32)                8448      
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 33        
=================================================================
Total params: 32,577
Trainable params: 32,577
Non-trainable params: 0
```

4. Performance and analysis

   1. 2D CNN model

      I tuned the batch size and find that 64 is optimal.

      Result:

```
model_b = create_model()
model_b.summary()
history_b = model_b.fit(X_train_4, y_train, epochs=50, batch_size=64,
                        validation_split=0.15, verbose=2, callbacks=callbacks_list)
```

```
Train on 107155 samples, validate on 18910 samples
Epoch 1/50
 - 45s - loss: 0.3840 - acc: 0.8205 - val_loss: 0.4019 - val_acc: 0.8352
Epoch 2/50
 - 44s - loss: 0.2070 - acc: 0.9128 - val_loss: 0.4120 - val_acc: 0.8570
Epoch 3/50
 - 44s - loss: 0.1441 - acc: 0.9420 - val_loss: 0.4535 - val_acc: 0.8640
Epoch 4/50
 - 44s - loss: 0.1109 - acc: 0.9572 - val_loss: 0.4268 - val_acc: 0.8702
Epoch 5/50
 - 44s - loss: 0.0891 - acc: 0.9662 - val_loss: 0.4439 - val_acc: 0.8780
Epoch 6/50
 - 44s - loss: 0.0741 - acc: 0.9718 - val_loss: 0.6538 - val_acc: 0.8501
Epoch 00006: early stopping
```

Figure about training and validation accuracy:

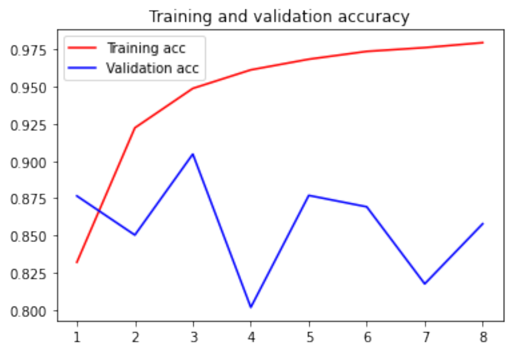
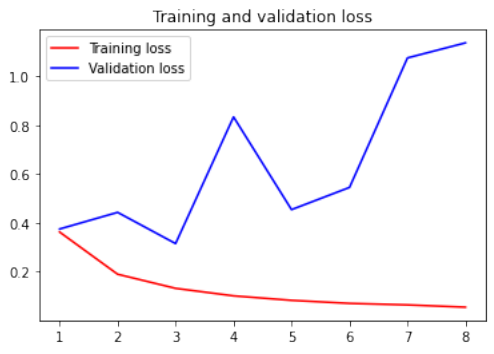


Figure about training and validation loss:



Loss and accuracy on test set:

```
score_b = model_b.evaluate(X_test_4, y_test, verbose=2)
print(score_b)
```

```
[0.9702688392152429, 0.8499804735183716]
```

2. 1D CNN model

I tuned batch size, time step, train-validation split and dropout, and find batch size = 32, time step = 15, train-validation split = 0.2, dropout = 0.2 to be optimal.

Result:

```
# create model
model_1 = create_model_1(X_train)
history = model_1.fit(X_train, y_train, epochs=EPOCH, batch_size=32,
                      validation_split=0.2, verbose=2, callbacks=callbacks_list)
score = model_1.evaluate(X_test, y_test, verbose=2)
print("test loss: ", score[0], "test acc: ", score[1])
```

```
Epoch 1/100
 - 22s - loss: 0.2806 - acc: 0.8874 - val_loss: 0.2119 - val_acc: 0.9192
Epoch 2/100
 - 20s - loss: 0.1724 - acc: 0.9351 - val_loss: 0.3051 - val_acc: 0.8964
Epoch 3/100
 - 21s - loss: 0.1414 - acc: 0.9488 - val_loss: 0.1299 - val_acc: 0.9563
Epoch 4/100
 - 20s - loss: 0.1167 - acc: 0.9576 - val_loss: 0.0714 - val_acc: 0.9725
Epoch 5/100
 - 20s - loss: 0.1042 - acc: 0.9634 - val_loss: 0.0642 - val_acc: 0.9752
Epoch 6/100
 - 21s - loss: 0.0935 - acc: 0.9675 - val_loss: 0.0714 - val_acc: 0.9745
Epoch 7/100
 - 20s - loss: 0.0831 - acc: 0.9711 - val_loss: 0.0670 - val_acc: 0.9759
Epoch 8/100
 - 20s - loss: 0.0743 - acc: 0.9739 - val_loss: 0.0229 - val_acc: 0.9911
Epoch 9/100
 - 21s - loss: 0.0713 - acc: 0.9762 - val_loss: 0.0559 - val_acc: 0.9813
Epoch 10/100
 - 20s - loss: 0.0686 - acc: 0.9773 - val_loss: 0.0462 - val_acc: 0.9834
Epoch 11/100
 - 20s - loss: 0.0642 - acc: 0.9789 - val_loss: 0.0153 - val_acc: 0.9948
Epoch 12/100
 - 21s - loss: 0.0642 - acc: 0.9785 - val_loss: 0.0319 - val_acc: 0.9903
Epoch 13/100
 - 20s - loss: 0.0620 - acc: 0.9804 - val_loss: 0.0361 - val_acc: 0.9902
Epoch 14/100
 - 20s - loss: 0.0556 - acc: 0.9815 - val_loss: 0.0741 - val_acc: 0.9702
Epoch 15/100
 - 21s - loss: 0.0544 - acc: 0.9824 - val_loss: 0.0157 - val_acc: 0.9954
Epoch 16/100
 - 20s - loss: 0.0534 - acc: 0.9825 - val_loss: 0.0485 - val_acc: 0.9895
Epoch 17/100
 - 21s - loss: 0.0508 - acc: 0.9840 - val_loss: 0.0761 - val_acc: 0.9745
```

```
Epoch 18/100
 - 20s - loss: 0.0480 - acc: 0.9847 - val_loss: 0.0128 - val_acc: 0.9968
Epoch 19/100
 - 20s - loss: 0.0502 - acc: 0.9851 - val_loss: 0.0082 - val_acc: 0.9970
Epoch 20/100
 - 20s - loss: 0.0464 - acc: 0.9857 - val_loss: 0.0134 - val_acc: 0.9954
Epoch 21/100
 - 20s - loss: 0.0445 - acc: 0.9866 - val_loss: 0.0601 - val_acc: 0.9850
Epoch 22/100
 - 20s - loss: 0.0462 - acc: 0.9859 - val_loss: 0.0823 - val_acc: 0.9796
Epoch 23/100
 - 20s - loss: 0.0415 - acc: 0.9870 - val_loss: 0.0076 - val_acc: 0.9978
Epoch 24/100
 - 20s - loss: 0.0453 - acc: 0.9864 - val_loss: 0.0105 - val_acc: 0.9972
Epoch 25/100
 - 21s - loss: 0.0393 - acc: 0.9876 - val_loss: 0.0465 - val_acc: 0.9851
Epoch 26/100
 - 21s - loss: 0.0392 - acc: 0.9878 - val_loss: 0.0131 - val_acc: 0.9971
Epoch 27/100
 - 21s - loss: 0.0408 - acc: 0.9878 - val_loss: 0.0111 - val_acc: 0.9966
Epoch 28/100
 - 21s - loss: 0.0377 - acc: 0.9892 - val_loss: 0.0061 - val_acc: 0.9982
Epoch 00028: early stopping
```

Figure about training and validation accuracy:

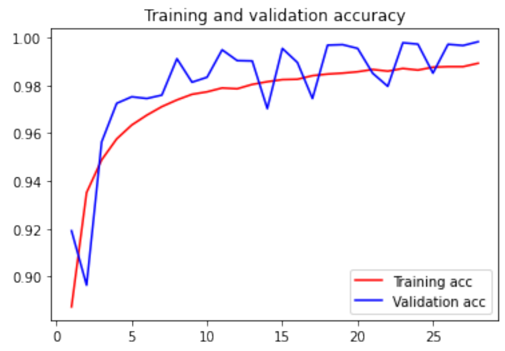
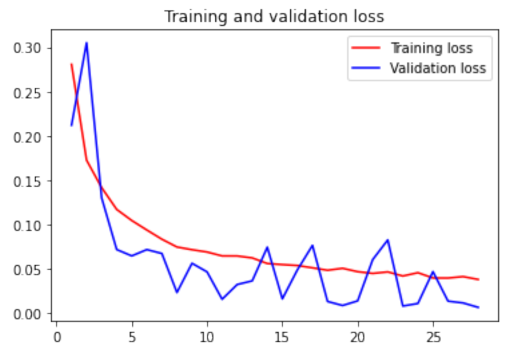
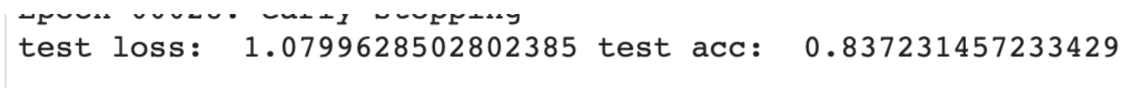


Figure about training and validation loss:



Accuracy and loss on test set:

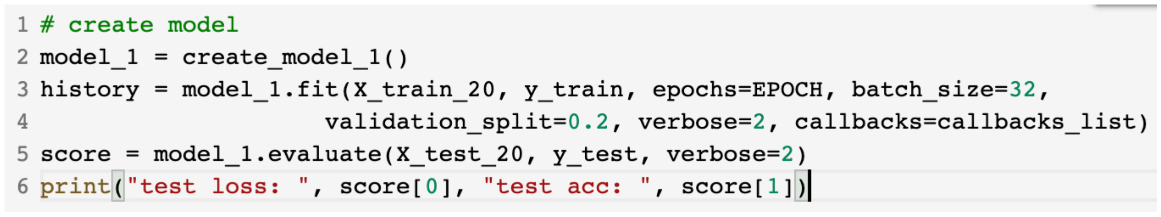


```
test loss:  1.0799628502802385 test acc:  0.837231457233429
```

3. LSTM model

   I tuned Dense units, LSTM units, batch size, time step, train-validation split and dropout, and find units = 32, batch size = 32, time step = 20, train-validation split = 0.2, dropout = 0.5 to be optimal.

   Result:



```
# create model
model_1 = create_model_1()
history = model_1.fit(X_train_20, y_train, epochs=EPOCH, batch_size=32,
                      validation_split=0.2, verbose=2, callbacks=callbacks_list)
score = model_1.evaluate(X_test_20, y_test, verbose=2)
print("test loss: ", score[0], "test acc: ", score[1])
```

```
Epoch 1/100
 - 6s - loss: 0.3732 - acc: 0.8367 - val_loss: 0.4272 - val_acc: 0.8331
Epoch 2/100
 - 5s - loss: 0.2366 - acc: 0.8992 - val_loss: 0.1817 - val_acc: 0.9240
Epoch 3/100
 - 5s - loss: 0.1832 - acc: 0.9209 - val_loss: 0.1494 - val_acc: 0.9368
Epoch 4/100
 - 5s - loss: 0.1509 - acc: 0.9387 - val_loss: 0.1202 - val_acc: 0.9559
Epoch 5/100
 - 5s - loss: 0.1262 - acc: 0.9508 - val_loss: 0.1264 - val_acc: 0.9576
Epoch 6/100
 - 5s - loss: 0.1059 - acc: 0.9610 - val_loss: 0.1362 - val_acc: 0.9535
Epoch 7/100
 - 5s - loss: 0.0878 - acc: 0.9696 - val_loss: 0.0599 - val_acc: 0.9829
Epoch 8/100
 - 5s - loss: 0.0764 - acc: 0.9742 - val_loss: 0.0661 - val_acc: 0.9801
```

```
Epoch 9/100
 - 5s - loss: 0.0690 - acc: 0.9775 - val_loss: 0.0363 - val_acc: 0.9884
Epoch 10/100
 - 5s - loss: 0.0572 - acc: 0.9812 - val_loss: 0.0709 - val_acc: 0.9742
Epoch 11/100
 - 5s - loss: 0.0577 - acc: 0.9830 - val_loss: 0.0333 - val_acc: 0.9867
Epoch 12/100
 - 5s - loss: 0.0505 - acc: 0.9843 - val_loss: 0.0308 - val_acc: 0.9895
Epoch 13/100
 - 5s - loss: 0.0468 - acc: 0.9858 - val_loss: 0.0252 - val_acc: 0.9913
Epoch 14/100
 - 5s - loss: 0.0442 - acc: 0.9864 - val_loss: 0.0186 - val_acc: 0.9935
Epoch 15/100
 - 5s - loss: 0.0503 - acc: 0.9861 - val_loss: 0.0173 - val_acc: 0.9947
Epoch 16/100
 - 5s - loss: 0.0399 - acc: 0.9878 - val_loss: 0.0386 - val_acc: 0.9893
Epoch 17/100
 - 5s - loss: 0.0361 - acc: 0.9893 - val_loss: 0.0119 - val_acc: 0.9959
Epoch 18/100
 - 5s - loss: 0.0375 - acc: 0.9893 - val_loss: 0.0095 - val_acc: 0.9970
Epoch 19/100
 - 5s - loss: 0.0323 - acc: 0.9906 - val_loss: 0.0498 - val_acc: 0.9875
Epoch 20/100
 - 5s - loss: 0.0324 - acc: 0.9906 - val_loss: 0.0352 - val_acc: 0.9880
Epoch 21/100
 - 5s - loss: 0.0369 - acc: 0.9902 - val_loss: 0.0143 - val_acc: 0.9967
Epoch 22/100
 - 5s - loss: 0.0279 - acc: 0.9927 - val_loss: 0.0237 - val_acc: 0.9924
Epoch 23/100
 - 5s - loss: 0.0300 - acc: 0.9917 - val_loss: 0.0114 - val_acc: 0.9963
Epoch 00023: early stopping
```

Figure about training and validation accuracy:

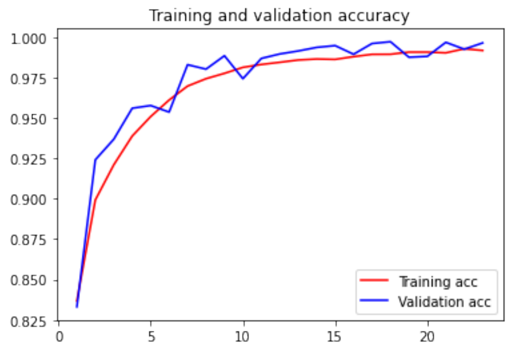
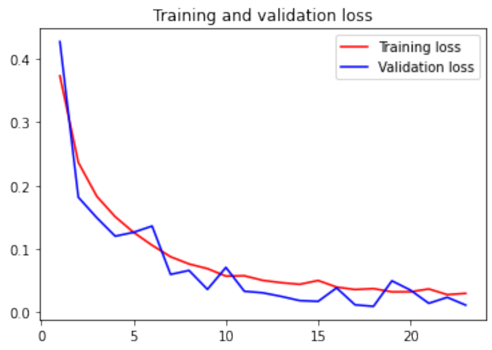


Figure about training and validation loss:



Accuracy and loss on test set:

```
test loss:  0.5815535898602696 test acc:  0.8855501413345337
```

5. Instructions on how to test my code

1. My codes on Github: https://github.com/YiranH/Cry-Detection-in-Real-Time.git
2. My codes and data on Google Colab:

   https://drive.google.com/open?id=17LOBLJLDFgHtDvpQV39D5OBpooPrtlvv

   Demo code on Google Colab:

   https://drive.google.com/open?id=10b7DBiOGlapxjjlYHh6pCj6xQllf8aaB

   Training code:

   https://drive.google.com/open?id=1T3SIWggkeZrPx4Iezw9qOGjeo2iUlU8i

   https://drive.google.com/open?id=1PvS18tl3G0RnAdE_7euRV8MI_lYI6UfS

   https://drive.google.com/open?id=1rch_iFtvEwpTTF8f8Lwd1GsEAgKH1QtM

3. My sample videos are the last 5 test videos sent to me, and the figures and json files are generated by the LSTM model.

   8) 1600_4, https://youtu.be/0mc6IJd0NEw
   9) 1700_2, https://youtu.be/LqRJSNM814g
   10) 1800_2, https://youtu.be/IS3YZ9JSsnc
   11) 1900_2, https://youtu.be/UUtdMuT1EBU
   12) 2000_2, https://youtu.be/NEkWx0pmzq4