“Cry” Detection Report

April 30 2020

1. Topic

   My topic is “crying”.

2. Dataset

   There are barely any datasets about crying detection. There are some in the "STAIR Actions" video dataset, but the data in STAIR focus more on action than facial expression, and in most of the videos about crying, people are just covering their face with hands to pretend crying. I think this is not the data I want to use in this project, so I self-crawled data on many video websites, including youtube.com, bilibili.com, etc.

   In Part 4, I crawled data from a Youtube channel (https://www.youtube.com/user/watchcut), which presents a series about one or two person answering questions in front of a camera against a minimalist background. I crawled the playlist and used some of the videos to generate the dataset. I collected 165 clips from 12 videos about different person with all kinds of facial expressions, adding up to 6781 frames.

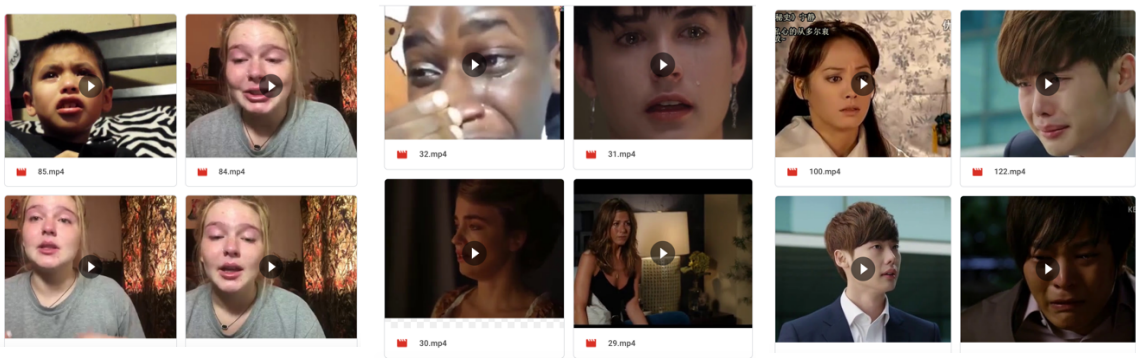
   In Part 5, I made a larger dataset by collecting the crying selfie videos on Youtube and some news videos about celebrities crying in front of the camera. I collected about 50 videos and picked 20 from them where I can tell whether people are crying or not from the picture. Different from the dataset I created in Part 4, I used longer

videos in Part 5, so that there are data about people from cry to not cry, vice versa. In Part 4, I collected many video clips about people crying and people not crying, and there are no data about how they turn to cry or how they managed to hold back their tears, which I think is a progress.

In Part 6, I made the dataset almost twice as large by the same way in Part 5, adding up to 149633 frames, and the proportion of 0 and 1 is 0.55 : 0.45.

Since I make long videos into several clips to be used in the dataset, there are chances that the video clips of one person (not the same video) appear in both test and training set. To avoid this, instead of using sklearn, I manually split the dataset into training and test set, making sure they do not have video clips of the same person.

In Part 8, I collected more videos, making my dataset as large as 151433 frames, in which the number of positive data is 68666, and negative 82767, and the proportion of positive and negative is 0.45. I also made a dataset with 6477 300 * 300 facial images using a part of the large dataset and a tool called autocrop (https://github.com/leblancfg/autocrop), to be used in transfer learning with VGG16 model.



I used openpose to retrieve facial landmarks and body landmarks. I used the 70 facial landmarks and the first 8 body landmarks which are nose, neck, right shoulder, right elbow, right wrist, left shoulder, left elbow, left wrist.

For the dataset, I take every 10 frames as a timestep, which is to say, I use these 10 frames to predict the facial expression in the 10^{th} frame. I tried to use 30 and 50 frames as a timestep but results show that 10 is better than 30 and 50.

To sum up, a total of 151433 frames are processed by OpenPose and transformed into a 3-dimensional array of size (151433, 10, 156), in which 151433 is the number of samples, 10 is the timestep, and 156 is the number of features.

3. Model

1. Model in Part 4

I used a simple LSTM model in Part 4, and to make better use of the small dataset, I adopted K-fold validation.

The first layer is a Dense layer of size 32, which takes input arrays of 3D and output arrays of size (3184, 10, 32), in which 3184 is the size of the training set and uses the activation function ‘relu’ to add non-linearity to the data.

The second layer is an LSTM layer. Since facial expression is time related, and you can predict one’s future facial expression from one’s current facial expression, I consider it reasonable to use LSTM layer in my model, because it can carry information across many timesteps. The input of this layer is (3184, 10, 32), output is (3184, 32).

The third layer is also a Dense layer. Because crying detection is a binary-classification problem, I need to end the network with a single unit and a sigmoid activation. The input of this layer is (3184, 32), output of this layer is (3184, 1), which is the probability of result being 0 or 1.

I chose to tune batch size and epochs. After experimenting batch size of [10, 100], I find 50 to be optimal. After experimenting epochs of [10, 100], I find 30 to be optimal.

```
def create_model():
  model = Sequential()
  model.add(Dense(32, activation='relu'))
  model.add(LSTM(32))
  model.add(Dense(1, activation='sigmoid'))
  model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
  return model
```

```
# callbacks
early_stop = EarlyStopping(monitor='val_loss', min_delta=0.01, patience=10,
                           verbose=2, mode='min')
callbacks_list = [early_stop]
```

K-fold

```
[ ]  1 from sklearn.model_selection import StratifiedKFold
```

```
[23]  1 # fix random seed for reproducibility
      2 seed = 7
      3 np.random.seed(seed)
      4
      5 # define 4-fold cross validation test harness
      6 kfold = StratifiedKFold(n_splits=4, shuffle=True, random_state=seed)
      7 scores = []
      8 i = 0
      9 for train, test in kfold.split(X, y):
     10   print('K-fold: %d\n' % i)
     11   i += 1
     12   model = create_model()
     13   history = model.fit(X[train], y[train], epochs=30, batch_size=50,
     14                       validation_split=0.2, verbose=2, callbacks=callbacks_list)
     15   score = model.evaluate(X[test], y[test], verbose=2)
     16   scores.append(score)
```

```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_39 (Dense)             (None, 10, 32)            6752
_________________________________________________________________
lstm_20 (LSTM)               (None, 32)                8320
_________________________________________________________________
dense_40 (Dense)             (None, 1)                 33
=================================================================
Total params: 15,105
Trainable params: 15,105
Non-trainable params: 0
```

2. Model in Part 5

I tried bidirectional LSTM and 2-layer LSTM model, in which 2-layer LSTM works best.

2-layer LSTM

```
def create_model():
  model = Sequential()
  model.add(Dense(units = LSTM_OUTPUT_DIM, input_shape=(TIME_STEP, INPUT_DIM), activation='re
  model.add(LSTM(units = LSTM_OUTPUT_DIM, input_shape=(TIME_STEP, INPUT_DIM), return_sequence
  model.add(Dropout(DROPOUT))
  model.add(LSTM(units = LSTM_OUTPUT_DIM))
  model.add(Dense(1, activation='sigmoid'))
  model.summary()
  model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
  return model

# callbacks
early_stop = EarlyStopping(monitor='val_loss', min_delta=0.01, patience=10,
                           verbose=2, mode='min')
callbacks_list = [early_stop]
```

```
# create model
model = create_model()
history = model.fit(X_train, y_train, epochs=EPOCH, batch_size=BATCH_SIZE,
                    validation_split=0.2, verbose=2, callbacks=callbacks_list)
score = model.evaluate(X_test, y_test, verbose=2)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 10, 64)            10048
_________________________________________________________________
lstm_3 (LSTM)                (None, 10, 64)            33024
_________________________________________________________________
dropout_2 (Dropout)          (None, 10, 64)            0
_________________________________________________________________
lstm_4 (LSTM)                (None, 64)                33024
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65
=================================================================
Total params: 76,161
Trainable params: 76,161
Non-trainable params: 0
_________________________________________________________________
```

3. Model in Part 6

   I used the 2-layer LSTM model, but the accuracy is low, so I tried to use CNN model. CNN models are developed for image classification, and 1D CNNs work well for the analysis of a time series of sensor data.

   The first 1D CNN layer defines a filter (or also called feature detector) of height 10 (also called kernel size). Only defining one filter would allow the neural network to learn one single feature in the first layer.

   The second layer is a max pooling layer, which is often used after a CNN layer in order to reduce the complexity of the output and prevent overfitting of the data.

   The third layer is a flatten layer that converts the data into a 1-dimensional array for inputting it to the next layer.

   The last 2 layers are dense layers that changes the dimension of data to prediction.

```
def create_model_4():
  model = Sequential()
  model.add(Conv1D(filters=64, kernel_size=2, activation='relu',
                   input_shape=(TIME_STEP, INPUT_DIM)))
  model.add(MaxPooling1D(pool_size=2))
  model.add(Flatten())
  model.add(Dense(50, activation='relu'))
  model.add(Dense(1, activation='sigmoid'))
  model.summary()
  model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
  return model

# callbacks
early_stop = EarlyStopping(monitor='val_loss', min_delta=0.01, patience=10,
                           verbose=2, mode='min')
callbacks_list = [early_stop]
```

```
# create model
model = create_model_4()
history = model.fit(X_train, y_train, epochs=EPOCH, batch_size=BATCH_SIZE,
                    validation_split=0.15, verbose=2, callbacks=callbacks_list)
score = model.evaluate(X_test, y_test, verbose=2)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_15 (Conv1D)           (None, 9, 64)             20032
_________________________________________________________________
max_pooling1d_12 (MaxPooling (None, 4, 64)             0
_________________________________________________________________
flatten_12 (Flatten)         (None, 256)               0
_________________________________________________________________
dense_14 (Dense)             (None, 50)                12850
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 51
=================================================================
Total params: 32,933
Trainable params: 32,933
Non-trainable params: 0
_________________________________________________________________
Train on 108211 samples, validate on 19097 samples
```

4. Model in Part 8

   As my dataset grows larger, the accuracy of my models is decreasing. I tried my original 2-layered LSTM model and 1-layered CNN model, but the accuracy on test set is always around 0.6, no matter how many hyperparameters I tuned. I also tried a 3-layered LSTM model with no improvement on the accuracy. Then I tried multi-layered CNN model, improving my accuracy from around 0.7 in Part 6 to a maximum of 0.78.

This CNN model is initially made for grey-scaled image data, as it has more dimensions and features. I reshaped my data from (, 10, 156) to (, 40, 39, 1) in order to fit to the model. I tried to process my dataset to resize it all into 300 * 300 facial images, but since I do not have my own GPU and Google Colab is too slow, I was not able to do them all, and only transformed 6477 images. The size of facial image dataset is too small, so I decided not to used it in this CNN model but use it in transfer learning with VGG16. And so far, this CNN model works best with my dataset.

```
X_train = X_train.reshape(X_train.shape[0], 40, 39, 1)
X_test = X_test.reshape(X_test.shape[0], 40, 39, 1)
```

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization,AveragePooling2D
from keras.losses import categorical_crossentropy
from keras.optimizers import Adam
from keras.regularizers import l2
from keras.utils import np_utils
##designing the cnn
def create_model():
    #1st convolution layer
    model = Sequential()

    model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape=(X_train.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_b = create_model()
model_b.summary()
history_b = model_b.fit(X_train, y_train, epochs=50, batch_size=128,
                    validation_split=0.15, verbose=2, callbacks=callbacks_list)
```

I tuned the dropout (0.2, 0.4, 0.5, 0.8), batch size (64, 128, 256) and the number of conv2D filters and kernel sizes, and find that dropout = 0.5, batch size = 128, filter = 64 in first layer are the optimum combination.

4. Performance and analysis

1. Part 4

Since dataset is small and balanced, containing frames of the same style, the result turns good.

Figure about training and validation accuracy:

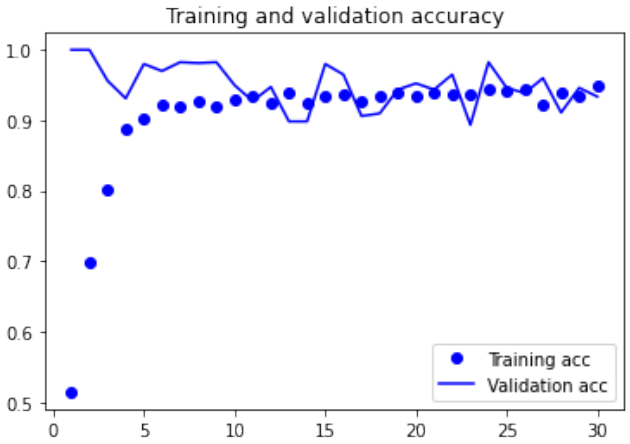
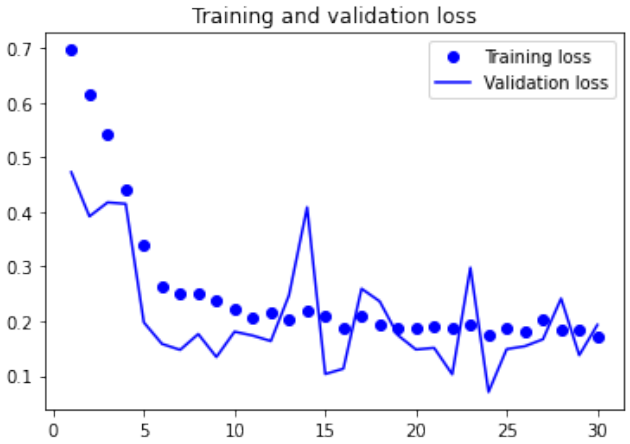


Figure about training and validation loss:



Accuracy and loss on test set:

```
[0.15788169593719406, 0.9562923888470234]]
```

2. Part 5

Figure about training and validation accuracy:

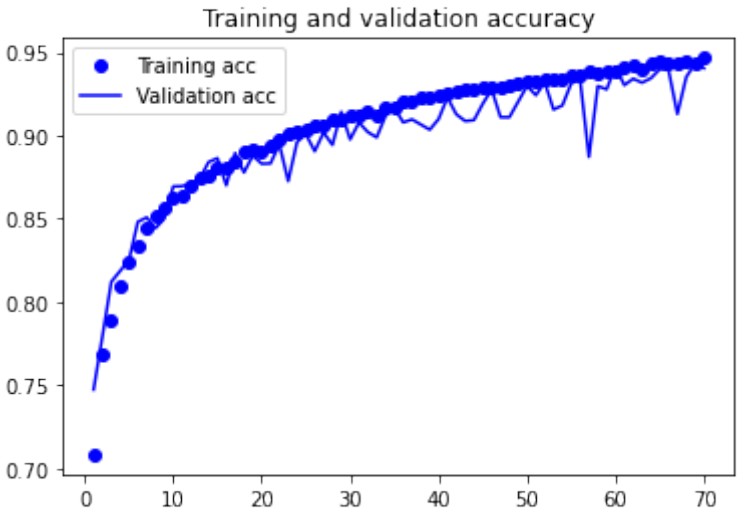
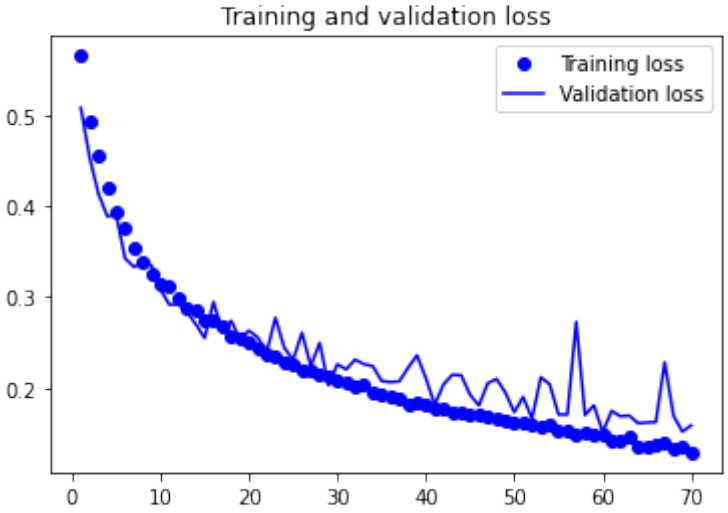


Figure about training and validation loss:



Accuracy and loss on test set:

```
1 score2
```

```
[0.148495629652214, 0.9434428762748532]
```

3. Part 6

   Due to large amount of new data of different angle, light and environment and the fact that I avoided having the video clips of the same person both in training and test set, the accuracy decresed.

   Figure about training and validation accuracy:

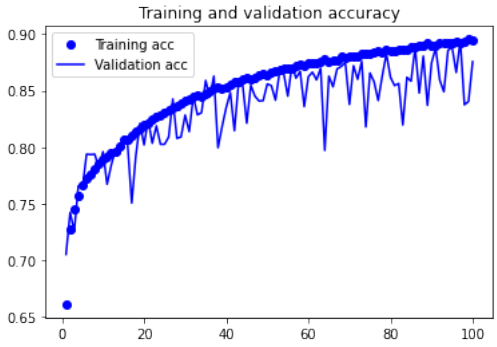
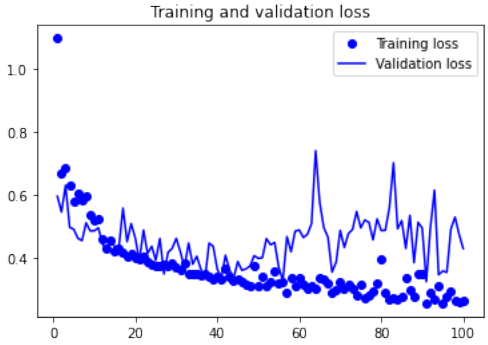


Figure about training and validation loss:



Accuracy and loss on test set:

```
1 print("test loss: ", score[0], "test acc: ", score[1])
```

```
test loss:  2.9137049043930654 test acc:  0.7288689613342285
```

4. Part 8

Result (This is not the best training result, the best one is deleted by me by accident):

```
 - 29s - loss: 0.2680 - acc: 0.8837 - val_loss: 0.2216 - val_acc: 0.9070
Epoch 21/50
 - 29s - loss: 0.2656 - acc: 0.8854 - val_loss: 0.2154 - val_acc: 0.9091
Epoch 22/50
 - 29s - loss: 0.2631 - acc: 0.8898 - val_loss: 0.2144 - val_acc: 0.9081
Epoch 23/50
 - 29s - loss: 0.2613 - acc: 0.8874 - val_loss: 0.2498 - val_acc: 0.8952
Epoch 24/50
 - 29s - loss: 0.2912 - acc: 0.8886 - val_loss: 0.2487 - val_acc: 0.8959
Epoch 25/50
 - 29s - loss: 0.2667 - acc: 0.8883 - val_loss: 0.2175 - val_acc: 0.9051
Epoch 26/50
 - 29s - loss: 0.2963 - acc: 0.8892 - val_loss: 0.1987 - val_acc: 0.9173
Epoch 27/50
 - 29s - loss: 0.4179 - acc: 0.8911 - val_loss: 0.4420 - val_acc: 0.8050
Epoch 28/50
 - 29s - loss: 0.3160 - acc: 0.8659 - val_loss: 0.2111 - val_acc: 0.9100
Epoch 29/50
 - 29s - loss: 0.2492 - acc: 0.8938 - val_loss: 0.1934 - val_acc: 0.9160
Epoch 30/50
 - 29s - loss: 0.2406 - acc: 0.8975 - val_loss: 0.1822 - val_acc: 0.9242
Epoch 31/50
 - 29s - loss: 0.2346 - acc: 0.9000 - val_loss: 0.1794 - val_acc: 0.9252
Epoch 32/50
 - 29s - loss: 0.2802 - acc: 0.8926 - val_loss: 0.1897 - val_acc: 0.9197
Epoch 33/50
 - 29s - loss: 0.2336 - acc: 0.9000 - val_loss: 0.1846 - val_acc: 0.9252
Epoch 34/50
 - 29s - loss: 0.2531 - acc: 0.8993 - val_loss: 0.1801 - val_acc: 0.9258
Epoch 35/50
 - 29s - loss: 0.3248 - acc: 0.8915 - val_loss: 0.1967 - val_acc: 0.9145
Epoch 00035: early stopping

Train on 109404 samples, validate on 19307 samples
Epoch 1/50
 - 30s - loss: 0.6888 - acc: 0.5587 - val_loss: 0.6191 - val_acc: 0.6639
Epoch 2/50
 - 29s - loss: 0.5905 - acc: 0.6979 - val_loss: 0.5689 - val_acc: 0.7156
Epoch 3/50
 - 29s - loss: 0.5429 - acc: 0.7287 - val_loss: 0.4931 - val_acc: 0.7591
Epoch 4/50
 - 29s - loss: 0.4840 - acc: 0.7603 - val_loss: 0.4374 - val_acc: 0.7901
Epoch 5/50
 - 29s - loss: 0.4433 - acc: 0.7844 - val_loss: 0.3921 - val_acc: 0.8124
Epoch 6/50
 - 29s - loss: 0.4115 - acc: 0.8052 - val_loss: 0.3576 - val_acc: 0.8362
Epoch 7/50
 - 29s - loss: 0.3890 - acc: 0.8196 - val_loss: 0.3288 - val_acc: 0.8537
Epoch 8/50
 - 29s - loss: 0.3749 - acc: 0.8282 - val_loss: 0.3375 - val_acc: 0.8453
Epoch 9/50
 - 29s - loss: 0.3532 - acc: 0.8382 - val_loss: 0.3337 - val_acc: 0.8515
Epoch 10/50
 - 29s - loss: 0.3380 - acc: 0.8461 - val_loss: 0.2995 - val_acc: 0.8704
Epoch 11/50
 - 29s - loss: 0.3330 - acc: 0.8510 - val_loss: 0.2768 - val_acc: 0.8785
Epoch 12/50
 - 29s - loss: 0.3317 - acc: 0.8548 - val_loss: 0.2729 - val_acc: 0.8829
Epoch 13/50
 - 29s - loss: 0.3482 - acc: 0.8580 - val_loss: 0.2676 - val_acc: 0.8878
Epoch 14/50
 - 29s - loss: 0.3078 - acc: 0.8645 - val_loss: 0.2544 - val_acc: 0.8882
```

Figure about training and validation accuracy:

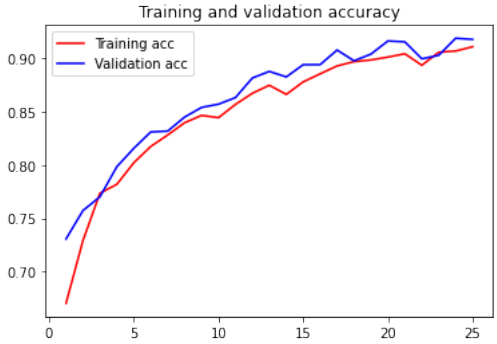
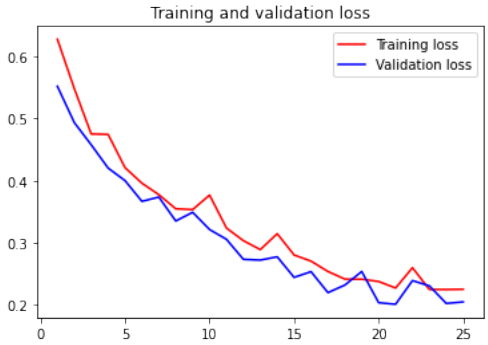


Figure about training and validation loss:



Accuracy and loss on test set:

```
[0.6477290755225208, 0.7586479783058167]
```

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 (Used in tutorial video):

   https://drive.google.com/open?id=1xRaWMwKL9Aax2ipmpg8McmNQrLAyH3f3

   Training code:

   https://drive.google.com/open?id=1w5IQ3hQdapfT3KfFum5BHhJdRBzVVbVl

   https://drive.google.com/open?id=1PE0oKoFVakDWH8Puz7VdKYtp9Iu-ZWAC

   https://drive.google.com/open?id=1Tuvb_6UCW9nZm6oFFHLIWUlmDDjnls6d

   https://drive.google.com/open?id=1nbtbWR_caI0ZUhidnoCt_j1YsAORkyff

3. My sample videos and tutorial video on YouTube (same as Part 5): (https://www.youtube.com/playlist?list=PLzFb8wKb-EHHnxpzFXXUx3iGnDLuOGZDn)