EMOTION ANALYSIS OF AUDIO BASED ON WAVE FEATURES WITH LSTM & CNN

Xiaoyan Feng*

University of Maryland
xyan399@umd.edu

December 19, 2022

I. Introduction

As speech recognition systems have matured over the last decades, emotion recognition can be seen as going one step further in the design of natural, intuitive, and human-like computer interfaces. In this paper emotion is automatically detected from the ripple of sound, the frequency of speech. And it is about to classify voice data into different kinds of emotions. Formally, it is Speech Emotion Recognition, abbreviated as SER, which is the act of attempting to recognize human emotion and affective states from speech. This is capitalizing on the fact that voice often reflects underlying emotion through tone and pitch. This is also the phenomenon that animals like dogs and horses employ to be able to understand human emotion.

According to WHO's report about the global prevalence of mental disorders in 2019. We can know 13% of global population is living with mental disorders. Basically, emotions come up everyday, if we can capture the movement of those patients' emotions, it could be used to create some treatment plans. A simple example I can give is to have your device play a relaxing song when it detects a drop in your mood.

*Thank you to read my project

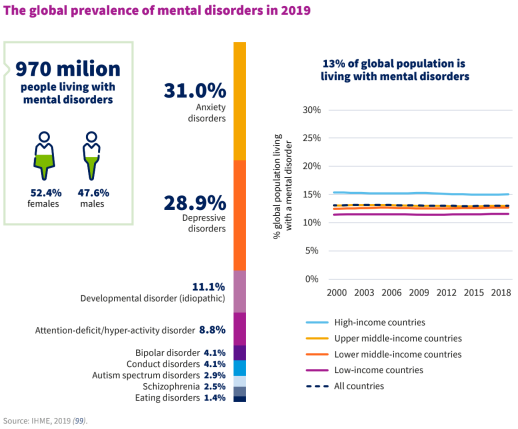

Figure 1: *Mental Disorders in 2019*

II. Dataset

i. Profile

This train data is combined by three file datasets, containing seven kinds of emotions. Each file duration is smaller than 8 seconds.

ii. Details

These datasets were recorded by professional actors.For controlling variables, each dataset uses an identical sentence to be text, so it can maximize the simulation of a scene where you do not understand the language but can distinguish emotions.

- Modality : audio data

- Size : 1.5GB
- Sample Size : 75KB
- Features Size : 168
- Labels : Fearful, Pleasant Surprise, Sad, Angry, Disgust, Happy, Neutral.

iii. Pre-processing

Since my datasets are .wav format, so I have to do pre-process and transfer it into regular data format. And extract features from them.

- Label each sample
  Since I use blended datasets, each dataset has different labelling format. Thus, a quite important step is to get a list of all labels.

- Transfer from wave to spectrum
  For this basic step, I use librosa library to do transfer. After that, I can get an array of HZ of spectrum to stand for each wave.

- Extract features
  Extract the MFCC, CHROMA, and MEL features from a sound file.
  MFCC: Mel Frequency Cepstral Coefficient, represents the short-term power spectrum of a sound
  CHROMA: Pertains to the 12 different pitch classes
  MEL: Mel Spectrogram Frequency

- Scale data to standard format
  I used StandardScaler to standard my features. **without scale**

- Balanced data
  By sns.countplot() function, I know my data is almost balanced. Only 'Neural' label data shows to be about 1% unbalanced, so I think I can ignore it.

- Other processing
  After I trained my model, I found the validation accuracy keep lower than 30%. Then even if I tried very hard on parameters adjustment, there were barely progresses of my training accuracy. So I came back to conduct a further work on my data. I realized there are lots of '0' in the front and end part of those arrays. Thus, I trimmed those parts of the decibel lower than 35 (not in the middle).

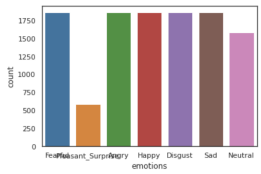

Figure 2: *Unbalanced data*

III. ML Methods

I trained two models, one is RNN model based on LSTM, another is CNN. Overall, CNN model performs better than LSTM model. For one thing, metrics of CNN generally better than LSTM model. For the other, times of parameters adjustment of CNN model are less than LSTM model.

i. RNN Model

- Split Train, Validate and Test Data
  At first, I split my data by 6:2:2 respectively for train, validate and test. While my data didn't train, I rase this split rate of train data to 0.25

- Create LSTM Model
  There is no doubt that I will use LSTM to train my emotion recognition model. Because LSTM performs quite well in processing sequential data.
  Dataset From my previous data preprocessing, you probably have understood that I am converting the input speech signal into the sequential data. Since LSTM need three dimensional input counting time dimension, I reshape my data into three dimensions input first.For label, I use OneHotEncoder to convert

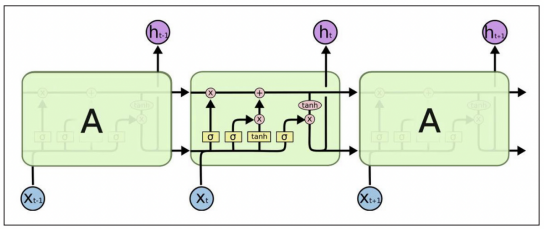

Figure 3: *Sequential LSTM layer internal architecture*

label to fit model.

Library

- import library: Keras
- import models: Sequential
- import layers: Dense, LSTM, Dropout

Loss and Optimizer

- Loss: categorical crossentropy
- Optimizer: Adam with 0.001 learning rate

- Challenges
  My original dataset only have 2000 rows while it has 168 features. It encountered overfitting more easier than I thought. Then, I used three different types of speech datasets for emotion recognition, and the overfitting conditions decreased. However, even I do dropout and keras initializer, it still got overfitted. The accuracy score couldn't break 70%. I thought it blamed on my datasets are slightly different.

ii. CNN Model

- Create CNN Model
  Convolutional Neural Network (CNN) is a multilayer representation learning architecture which has received immense success in multiple applications such as object classification, image segmentation, and natural language processing. This time I use CNN in classification. And it is definitely the best algorithm I have ever used.

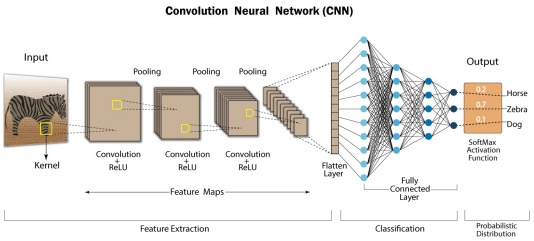

Figure 4: *CNN*

Library

- import library: Keras
- import models: Sequential
- import layers: Conv1D, BatchNormalization, MaxPool1D, Dropout

Loss and Optimizer

- Loss: categorical crossentropy
- Optimizer: Adam with 0.001 learning rate
- Metrics: acc, f1

IV. Results

i. RNN-LSTM Model

I experienced many problems with my LSTM model. For instance, in the very beginning the model was almost in a non-trained state and the **val loss** was surprisingly high. For adding more background, (1) I initially trained with callbacks, and set the patience to 10, which caused my model to always finish training at 60+ epochs. (2) My data at first only have 2800 rows while I have 168 features.
I didn't know what was happened, so I

adjusted parameters as usual. But every time I trained my model, my **train acc** went up to 30 at most.
After that I double-checked my entire codes and outputs, I guessed this outcome probably caused by my input samples with large features. I realized large features will cause time-consuming problem while I didn't wait my model to be fully trained. Thus, I removed callbacks.
On the other hand, I tried to find more train data to solve this problem at its root. But I only add 9000 rows more data into my samples in the end.

In conclusion, my LSTM model parameters were adjusted by followed table based on data limitation.

Hyperparameters

activation function： softmax

	layers	learning_rate	acc	overfitting
1	Sigmoid	0.01	0.16	yes
2	Sigmoid	0.001	0.62	no
3	ReLu	0.01	0.16	yes
4	ReLu	0.001	0.67	no

Figure 5: *LSTM parameters*

Metrics

1. Accuracy

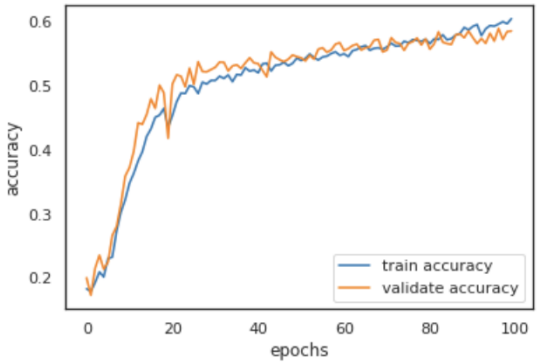

Figure 6: *ACC-CNN*

2. Loss

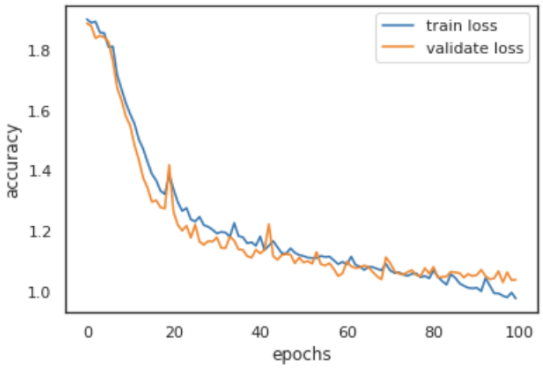

Figure 7: *LOSS-LSTM*

3. Metrics Map

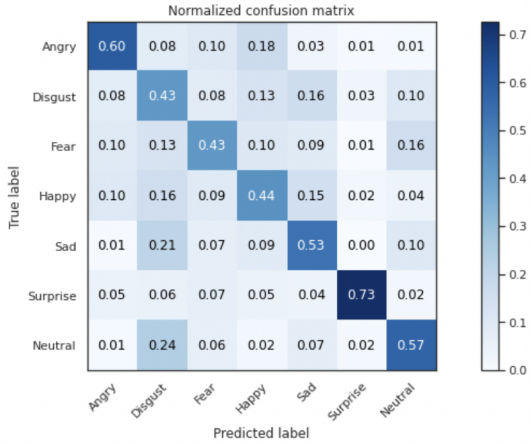

Figure 8: *METRICS-LSTM*

ii. CNN Model

Compared to the bumpy LSTM training experience, my CNN training is smooth. From label processing, I use LabelEncoder to fit my model. Using StandardScaler to normalize my data and split data by 64:16:20.
There are 14 layers in my model. My kernel size is 5, stride of the first layer is 1 and the activation is relu. Although the accuracy of my CNN model stays at 65% or so, I think it acceptable considering 7 labels in my data.

In conclusion, my CNN model parameters were adjusted by followed table based on data limitation.

Hyperparameters

activation function： softmax
dropout: yes
layers :Conv1D-BatchNormalization-MaxPool1D

	layers	learning_rate	acc	overfitting
1	ReLu	0.01	0.61	no
2	ReLu	0.001	0.64	no

Figure 9: *CNN parameters*

Metrics

1. Accuracy

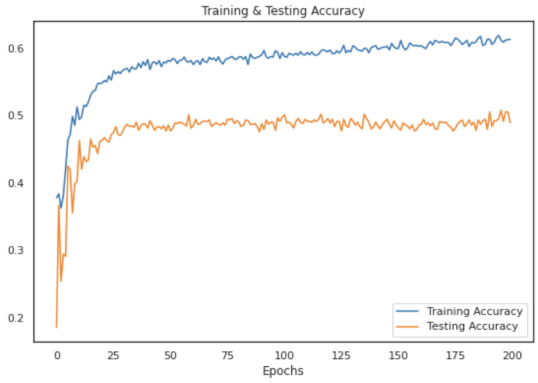

Figure 10: *ACC-CNN*

2. Loss

3. Metrics Map

V. Lessons learned

I learned that real dataset could be pretty complicated, there might be seldom relation between them. As a data scientist or analysis, we should be patient.
Beside, do a fully research before starting training. In my previous proposal, I didn't realize how difficulty audio data processing could be.

Figure 11: *LOSS-CNN*

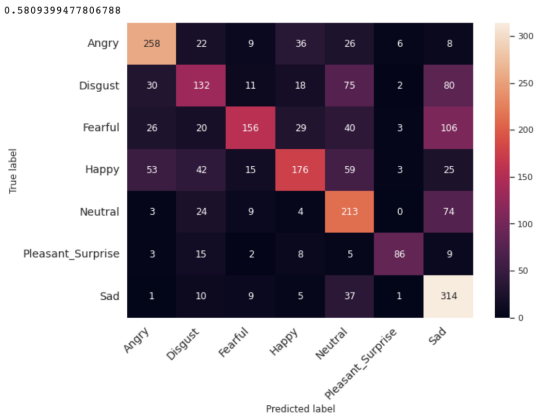

Figure 12: *METRICS-CNN*

So I use less time in extracting features. As a result, my features didn't fit my model and I had to do research again. And in that survey, I doubted LSTM probably was not the best algorithm for my data.
Also, adding more layers will cause overfitting. Large learning rate will give a small accuracy when your data samples are small.
Last but not the least, overfitting is not the only difficult enemy we will have to face in the future, as there are also problems with non-training, data shortage, etc.