LoopQPrize: My solution for the Speech Emotion Recognition task

Alessio Serra

May 2022

Abstract

In this documentation I present my solution for the Speech Emotion recognition task of the LoopQPrize challenge, in which I propose six different architectures that exploits combination of Convolutional Neural Network (CNN), Long Short Term Memory (LSTM) and also the Self-attention mechanism.
In particular, I present three families of models, i.e. CNN-LSTM, LSTM-CNN and CNN-Transfomer, which are finally combined in an ensemble that consider an average voting to get the final predictions, where the best combination of weights is found with a Genetic Algorithm (GA).
The model that can provide the best results is the CNN2D-LSTM, that achieves 71.4% of accuracy and 73.3% of F1-macro, while the ensemble achieves 74.5% of accuracy and 76.3% of F1-macro. These performance are computed on a validation set created with a stratified sampling of the training set.
I also do an analysis of the dataset and of the results achieved with the best single model and the ensemble, highlighting their positive features and limits.

1 Introduction

The Speech is the fastest way humans use to communicate and therefore is one of the most common ways to enable **human-machine interaction (HCI)**. However, to increase the naturalness and efficiency of some applications, e.g. in dialogue system or in on board vehicle driving system, the machine should also be able to understand the emotional state of the speaker.
This challenge aims at recognize the main **emotional content of a spoken utterance**, which is an inherently difficult task, since it implies a high subjectivity, in fact each individual can be influenced *in a different way and with different intensity from the same stimuli*. For example, [5, 14] suggested that women experience and express emotions more intensely than men, while [7] also proved that the age affects the way humans are able to recognize emotion.

Furthermore, the concept of emotion itself is not easy to categorize, in fact in the scientific literature there are several ways to represent it:

- **Dimensional approach:** uses a continuous representation in a three-dimensional space where emotions are mapped based on their valence (positive versus negative), arousal (calm versus excited) and control (or dominance). However, the control dimension has little effect, therefore, it is often considered a 2D emotional space with valence-arousal axis.
- **Category approach:** uses a discrete set of basic emotions that can also be assigned to a point in the Valence-Arousal-Control space, so it can be seen as a quantized version of the dimensional approach.

The latter approach is the one used in the challenge that follows the Ekman's division of emotions, in which a label among seven categories, namely *'fear', 'angry', 'happy', 'disgust', 'sadness', 'neutral'* and *'surprise'*, is assigned to each audio file.

To automatically recognize the emotion having only audio information, earlier approaches extract features from speech and recognize emotions utilizing Gaussian Mixture Models (GMMs) [10] and Hidden Markov Models (HMMs) [12]. However, recently the most used technique mostly employs deep learning models [3, 4, 15], thanks to their great ability to automatically discover the representations needed for the classification from the inputs received.

Used approach. The first step to follow for any machine learning algorithm is an accurate analysis of the data to fully understand the problem and the possible difficulties that may be encountered, followed by a feature engineering process to extract the most suitable representation to be given as input to the model.
The same logical path used to address the problem is followed by this documentation, which is composed by six main sections:

1. Dataset Analysis.
2. Feature Extraction.
3. Understanding the model's input.
4. Method Used.
5. Models Ensemble.
6. Analysis of the Results.

2 Dataset Analysis

2.1 Dataset sources

The dataset consists of audio files in the WAV format merged from four distinct sources:

1 - TESS. It is composed by 2800 audio files recorded by 2 female speakers (aged 26 and 64 years), which repeat with 7 different emotions the phrase *"Say the word _"*, where _ is taken from a set of 200 target words.
It has the advantage of being very clean, but it has low variability, since the audios have all almost the same duration (about 2 seconds) and the same structure.

2 − SAVEE. It is composed by 480 audio files recorded by 4 male speakers (aged from 27 to 31) which express 15 TIMIT sentences per emotion: 3 common, 2 emotion-specific and 10 generic sentences that were different for each emotion and phonetically-balanced. The 3 common and $2 \times 6 = 12$ emotion-specific sentences were recorded as neutral to give 30 neutral sentences. This resulted in a total of 120 utterances per speaker.
The variability of sentences and also their *"emotion-specific"* choice is the main advantage of this dataset, as it is increases the naturalness of the recorded speech.

3 - RAVDESS. It is composed by 2452 audio files recorded by 12 male and 12 female speakers that repeats only two statements, namely "*Kids are talking by the door"* and *"Dogs are sitting by the door"* with eight different emotions which are expressed with two different intensities (normal and strong), except for the neutral class.
It has two main advantages, in fact it does not contain noise, thanks to post processing and standardization of recordings, and it has a high level of emotions validity, as it was labeled by 319 distinct raters. On the other hand, one drawback is that the 24 actors involved always repeat the same two sentences, thus limiting its generality.

4 − CREMA. It is composed by 7442 audio files recorded by 48 male and 43 female speakers (aged from 20 to 74) of different races and ethnicities (African America, Asian, Caucasian, Hispanic, and Unspecified), that repeats 12 statements with 6 different emotions and 4 emotional levels (low, mid, high and unspecified). The labelling is highly reliable as it is obtained through crowd-sourcing involving 2,443 raters of different ages and with 40.5% male and 59.5% female.
The strengths of this dataset are the number and the variability of speakers and raters involved, that allows to increase the generality of the dataset and the reliability of the ground truth.

2.2 Advantages and limits of the whole dataset

The dataset is well balanced both in terms of classes (see next section for details) and for the speaker's gender. In fact the RAVDESS and the CREMA dataset already have balanced male and female speaker, while the SAVEE dataset, which is male-only, is compensated with TESS, which is female only.
Another good feature of the dataset is that it contains high quality data with clean voice recordings, which were recorded without noise.

However, there are also some limits:

- In real speech audio emotions are often **weakly expressed, mixed, and hard to distinguish** from each other and for this reason the dataset should involve the possibility of *multiple label and/or an associated intensity* of the emotion expressed, e.g. in a discrete range from 1 to 10.

- The model trained using this dataset would perform **significantly worse on real-world data** since recordings are done by actors that has to mimic the emotions, thus resulting in artificial samples where *speakers exaggerate the expression of feelings.*

- **Audio-only emotion recognition in many case cannot be sufficient to predict the correct emotion**. In fact all the dataset, except for TESS, are multi-modal and also associate a video clip with the audio to give more clues to the classification model. *The lack of visual data seriously degrades the performances*, as instance the authors of the CREMA dataset [2] shows that the human recognition rate for audio-only data is 40,9%.

- Due to the subjectivity in the human emotion recognition process, is needed a **large number of raters to have reliable labels** and it is reached only by RAVDESS and CREMA dataset, while the 10 and 56 raters of, respectively, the SAVEE and the TESS dataset may be insufficient.

- The majority of the recordings have **low variability both in the duration and in the structure of the sentence**, in fact there is a small fixed set of sentences that are repeated with different emotions, except for RAVDESS that has higher sentence variability.

2.3 Dataset statistics

The training set contains 10111 samples and is almost balanced, as shown in Figure 1, except for the *"surprise"* class which contains only 560 elements, while the average is 1444 per class. This is mainly due to the fact that the CREMA dataset, from which comes the majority of samples, does not contain *"surprise"* samples.

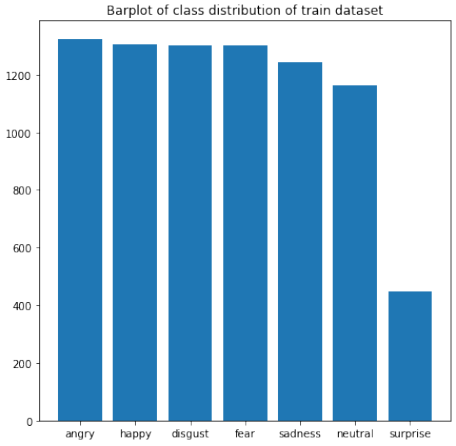


Figure 1: Class distribution in the training set.

Intra-dataset class distribution. Classes are almost balanced in all the sub-dataset, except for the SAVEE. In fact, as can be seen in Figure 2, twice as many samples belong to the *"angry"* and *"neutral"* classes, compared to the classes *"disgust"*, *"fear"*, *"surprise"* and *"happy"* (note that the class *"sadness"* is not present).

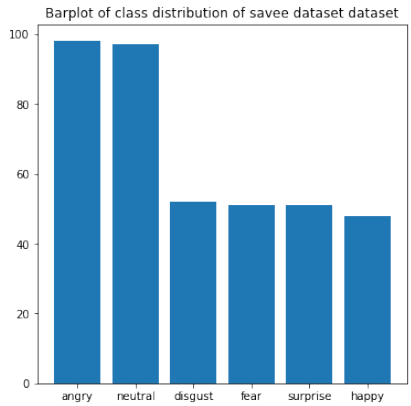
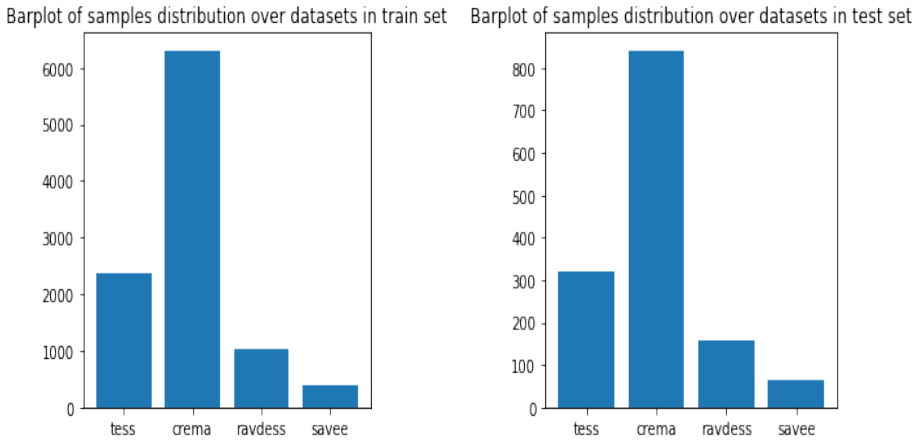


Figure 2: Class distribution in the SAVEE dataset.

Dataset origin distribution. In Figure 3 the origin of the training samples is shown and it can be seen that there is a large imbalance, in particular between the CREMA dataset (6306) and the SAVEE (397).
The same consideration can be done on the test set, which contains 1385 samples.



(a) Dataset distribution in the training set. (b) Dataset distribution in the test set.

Audio duration. Table 1 contains duration statistics of the dataset, also considering their origin, and shows that almost all the samples are quite short, with a median duration of around 2 seconds.

Dataset	**#samples**	**avg_dur**	**max_dur**	**min_dur**	**median_dur**
Training	**10111**	2.596986	**7.138730**	**1.254104**	2
Test	1385	2.620571	6.129524	1.312426	2
Tess	2367	2.054052	2.984807	**1.254104**	2
Crema	6306	2.541316	5.005034	1.267982	2
Savee	397	**3.866647**	**7.138730**	1.630930	3
Ravdess	1041	3.684523	5.271973	2.969660	3

Table 1: Information on the duration (in seconds) of the data set.

3 Feature Extraction

The extraction of robust and discriminatory features from audio data is crucial to build a machine learning model that can efficiently solve the task.
After a study of the recent literature [2, 9, 13], I decided to extract three widely used and effective features for the speech analysis, i.e. the *Mel Frequency Cepstral Coefficients (MFCCs)*, the *Log Mel Spectrogram* and the *Chromagram*.

Mel-Frequency Cepstral Coefficients (MFCCs). MFCC is a mathematical method which transforms the power spectrum of an audio signal to a small number of coefficients representing power of the audio signal in a frequency region (a region of pitch) taken w.r.t. time.
The *"mel"* prefix derives from the fact that the powers of the spectrum is mapped on the non-linear mel scale, which *approximates the human auditory system's response more closely* than the linearly-spaced frequency bands used in the normal spectrum.
To obtain the MFCC from the audio signal I have taken the following steps:

1. Apply the Short-Time Fourier Transform (STFT) on the audio signal and obtain its power spectogram.
2. Apply mel filterbanks (triangular window functions) to each STFT power spectrogram to map the spectrogram to the mel-scale.
3. Take the logs of the powers at each of the filter-banks energy.
4. Apply the discrete cosine transform (DCT) on them and use the DCT coefficients as the MFCC.

Regarding the implementation I used the python's Librosa audio preprocessing library to extract 40 MFC coefficient from each audio sample.

Log-mel spectrograms. Log mel-spectogram are obtained following the same procedure for the MFCC extraction up to step 3, thus obtaining a 2 dimensional feature that depends on the n_mels and the audio duration. It can be seen as *a simple analog of the power spectrogram* with the frequency scale in mels.
In particular I used 128 n_mels and I padded all the audio files with the same duration of 8 seconds, to *avoid losing information* since that the longest audio file is 7,14 s, thus obtaining a feature of (128, 251) dimension for each audio sample.

The chromagram. While this feature is used more frequently in music audio data rather than speech, I also extract this type of feature to train some models. In particular, to obtain the chromagram, it is computed the logarithmic Short-Time Fourier transform of the signal and is mapped into 12 bins that represents the 12 semitones (or chroma) of musical octave.

3.1 Features scaling

Normalization of features led to many advantages and should always be included in the audio processing pipeline, in fact:

- It makes the *training faster.*
- It gives a *better error surface shape.*
- It prevents the optimization from getting *stuck in local optima.*

The two most common choices are Min-Max normalization, which maps the entire set of values to a desired new range of values, and the standard scaling that **center the features to zero mean and unitary standard deviation.** Since I don't know the optimal target distribution of the scaled value, I preferred to use the standard scaler which subtracts the mean of each feature and divides it by the standard deviation of that feature.
This way extreme values will have less impact on the model's learned weights, i.e. the model is **less sensitive to outliers**.

To demonstrate the positive effects of feature scaling I performed an ablation study, which showed that training time decreased and accuracy increased when scaled features are used, as shown in Table 2.

Models	**Accuracy**	**Loss**	**F1-macro**	**Inference time**
cnn1D_lstm_no_scaled.h5	0.593472	1.041217	0.614307	0.910028
cnn1D_lstm.h5	0.622156	1.006559	0.639534	**0.733104**
cnn2D_lstm_no_scaled.h5	0.650841	0.971584	0.665839	3.247346
cnn2D_lstm.h5	**0.661721**	**0.968594**	**0.678884**	3.181240

Table 2: Ablation study to show the positive effects of feature scaling.

4 Understanding the model's inputs

In this section I try to visualize the inputs received by the models, to understand what are the points in common between samples of the same classes.
I have randomly selected two samples of similar duration of each class from the TESS dataset, then I plotted both their *log mel spectogram* and *chromagram*.

In this documentation I just show the classes *angry, happy* and *neutral* to avoid being too redundant.
However, similar considerations can be made by looking at the other plots, which can be found in the Jupiter Notebook.

4.1 Log mel spectogram

Observing all the pairs of samples of the same class it is easy to notice a **recurring pattern**, while looking at different classes there are clear differences.
Each couple shows a similar behavior of the frequencies (y-axis) with the passage of time (x-axis):

- **Neutral** class samples show *flat behavior* throughout time and can be spotted a little *"hill"* around 0.9 s.
- **Happy** class samples show three hills, around 0.5s, 1.2s and 1.6s. In particular the one on the left is narrower, but higher, while the one in the middle is wider and with a lower height, and the one on the right has almost the same height, but a different length as *different words are pronounced*.
- **Angry** class samples show a *wavy* behavior with almost the same pattern.

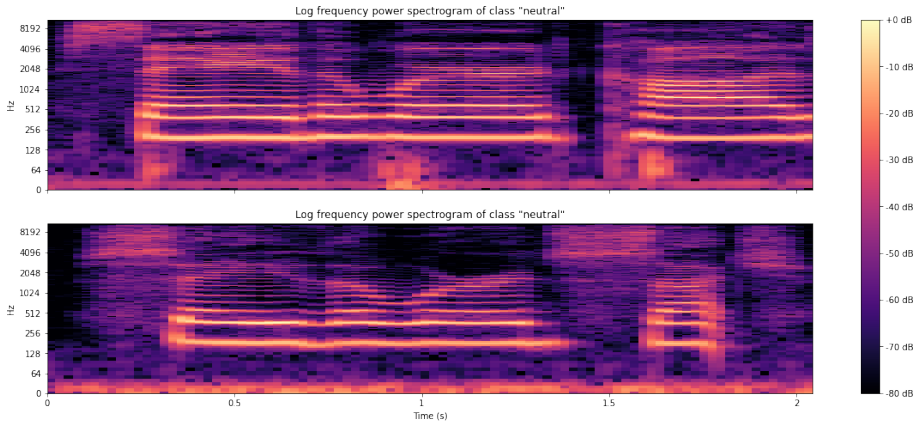


Figure 4: Log mel spectogram obtained with two samples of the ***neutral*** class.

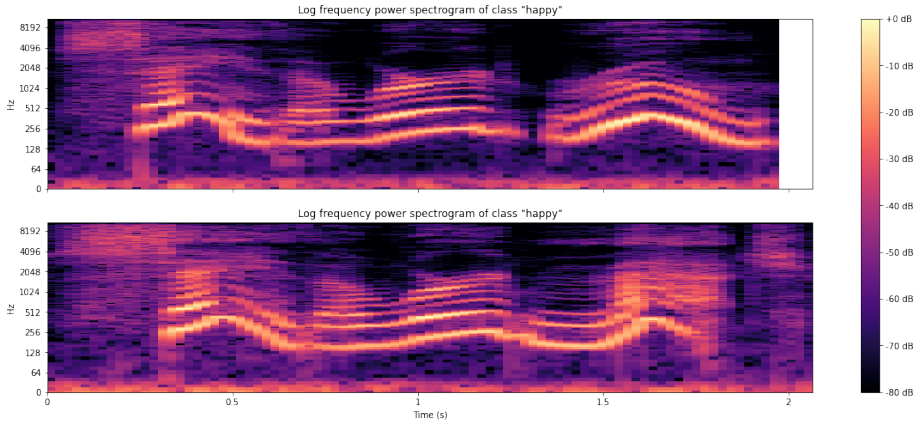


Figure 5: Log mel spectogram obtained with two samples of the ***happy*** class.

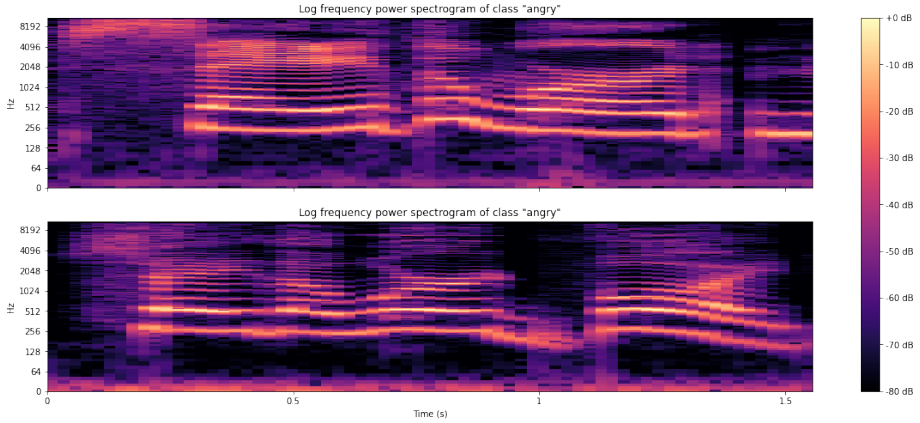


Figure 6: Log mel spectogram obtained with two samples of the ***angry*** class.

4.2 Chromagram

Looking at the chromagram plot can be seen that the pairs of samples of the same class have a very similar behavior for the pitch class (y-axis) with the passing of time (x-axis), while the differences between distinct classes are still many:

- **Neutral** class samples keep almost always the same pitch, i.e. *G*.
- **Happy** class samples show three hills almost at the same time of the log mel spectogram.
- **Angry** class samples start both with a pitch of class B, then around [0.8s-1.2s] they follow the same path of classes, i.e. *A-G-F-E-D-C*.

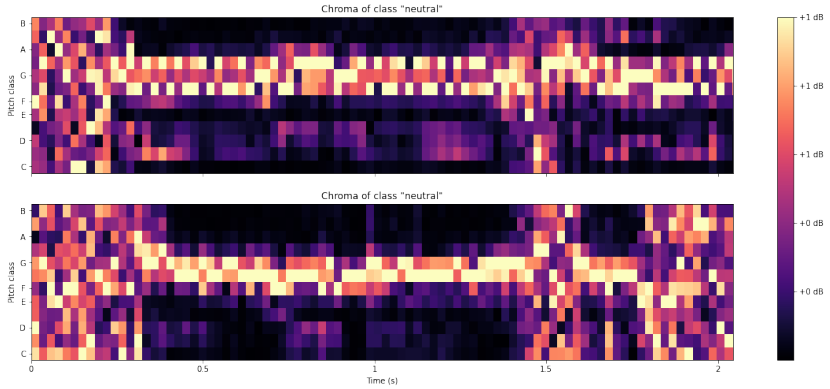


Figure 7: Chromagram obtained with two samples of the ***neutral*** class.

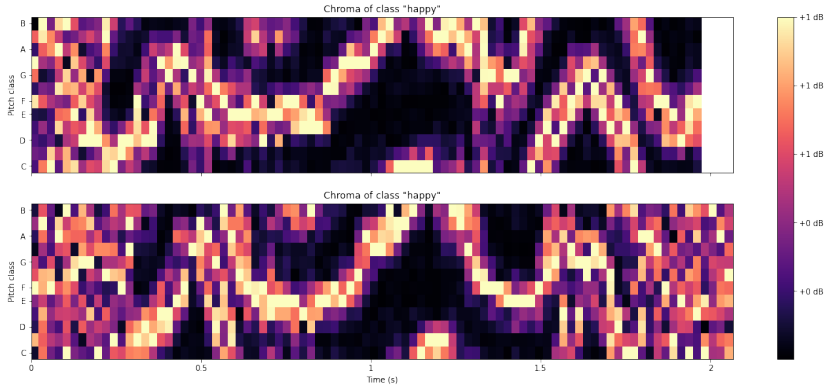


Figure 8: Chromagram obtained with two samples of the ***happy*** class.

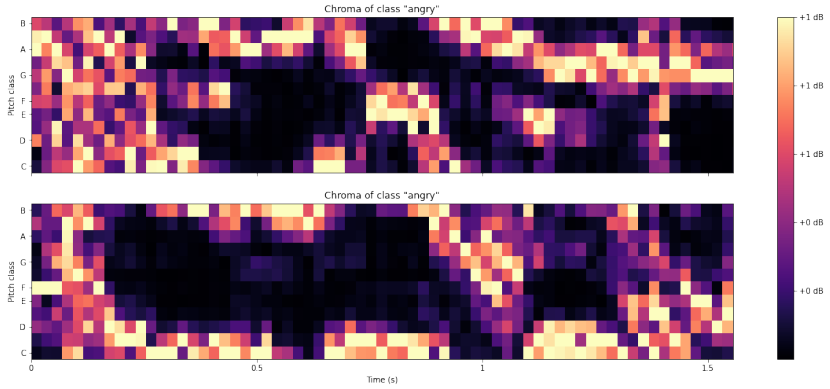


Figure 9: Chromagram obtained with two samples of the ***angry*** class.

5 Method Used

Different types of input's features and architectures were tested to achieve better performance, among them three main families can be distinguished: *CNN-LSTM, LSTM-CNN* and *CNN-Transformer*.

Model validation. To compare the performances of my models I used a **validation set**, which contains the 20% of the training set, so 2022 samples, with a balanced distribution of the seven classes, in fact it was obtained with a stratified sampling from the training set.
All the results reported in the following tables were done on the validation set, which is included in the submitted code, in order to make all the results of this documentation reproducible.

Metrics used. For each model i report as performance metrics the accuracy, the loss, the F1-macro and the *inference time*, i.e. the time to predict *all the samples* of the validation set.

Hyperparameter optimization. Different hyperparameters significantly affect the network performance, for this reason, for all the trained model, I first followed a trial and error approach to understand good ranges of parameters, but then I implemented a systematic search using the **Hyperband algorithm** [11].
This algorithm runs the Successive Halving several times initializing it with different parameters to find a **good compromise between exploration of the search space, and exploitation of the best solutions already found**.

Successive Halving algorithm. It requires as parameters the number of configuration to try (n) and the allocated resources (r), i.e. how many epoch of training can be done. So the increase of n favors exploration, while the increase of r favors exploitation.
After the initialization, it randomly samples n configurations from the search space and train them for n/r epochs. Then it selects only the best $n/\ \eta$ configurations, where η is the branching factor (default equal to 3), to further training them for a longer time. This is repeated until just one configuration survives and is selected as the current best solution.

5.1 CNN-LSTM

The architecture proposed is inspired by the work of [8], in which the authors built a deep neural network consisting of four convolutional blocks followed by one LSTM layer. In my work I propose a similar, but smaller network that has been optimized to cope with the LoopQPrize dataset, in particular it tries both 1D and 2D inputs and then a *fused network* that parallelizes the two CNN-LSTM architecture.

5.1.1 Building block of the architectures

CNN layer. The input passes through three convolutional blocks, which are used *to extract local features* from the input presented to the network. In particular each block is composed by:

- **Convolutional layer**, in which inputs are convolved with kernels (their number varies from 128 to 256 and have size equal to 3 or 5) to produce a feature map, which has the task of identifying a specific characteristic of the input.
- **Batch Normalization (BN) layer**, which normalizes the feature maps at each batch, thus increasing the stability and reducing the time of the training process, handling the *internal covariance shift problem.*
- **ELU (Exponential Linear Unit) activation function**, which has several advantages in common with ReLu, as it is not affected by the vanishing and the exploding gradient problem, but has also a faster convergence and does not have the dead ReLu problem.
- **Max-pooling layer**, which is used to reduce the size of feature maps *extracting only the most relevant elements*, i.e. dividing the input into a set of non-overlapping regions and returning only the maximum value of each such sub-region.

The output of convolutional blocks becomes progressively more abstract, in fact each block, except the first, receives as input the features produced by the previous block, thus extracting feature from feature maps and **increasing the level of abstraction**.

LSTM layer. It is a Recurrent Neural Network (RNN), that has been optimized to cope with long-term dependencies, as they often lead to the *vanishing gradient problem.* They are used to learn global features from sequence data that receives as input, which, in this case, coincides with the local features extracted from the signal. In particular, they try to *capture contextual dependencies* producing as output a single vector with a number of components equal to the number of units that compose the LSTM.

Classification layer. It is a single layer perceptron with a number of units equal to the number of classes, which produces the logits. The latter are then transformed into a probability distribution thanks to the softmax activation function, which maps all the elements (seven as the number of classes) in the range [0, 1] and to have their sum equal to 1.
The models presented in the following sections also have a classifier of the same type, placed as the last block of the network.
Finally the most probable class is selected as the proposed network's prediction.

5.1.2 CNN2D-LSTM

Input layer. From the audio files are extracted the *Log Mel Spectrogram* of the signal, as described in section 3, thus producing a two dimensional input, whose size depends on the length of the audio (each one has been padded with "0" to have features of the same size) and the number of FFT (128).

Search space and hyperparameter found. First I have defined the search space, which depends on the hyperparameters to test and their possible values, producing a 7-dimensions space to tune:

- **Convolutional kernel size:** choosing among 3, 5 (*"C_block"*).
- **Number of convolutional block:** searching in the discrete range [1, 4] (*"Ker_size"*).
- **Number of convolutional kernels of first block:** choosing among 32, 64, 128 (*"Ker_1"*).
- **Number of convolutional kernels of following blocks:** choosing among 64, 128, 256 (*"Ker_seq"*).
- **Number of LSTM units:** searching in the discrete range [32, 64] with step 4 (*"LSTM"*).
- **Learning rate:** choosing among 1e-3, 1e-4, 1e-5 (*"Lr"*).
- **Type of optimizer:** choosing between Adam and RMSprop (*"Opt"*).

The best hyperparameters found are shown in Table 3. For the following models I have left implicit the definition of the search space and of the range of values that they can assume, which in most cases are similar to the previous ones, and I just present the table with the best Hyperparameters.
The results obtained are summarized in Table 4.

Ker_size	#C_block	#Ker_1	#Ker_seq	LSTM	Lr	Opt
5	3	128	256	48	0.0001	Adam

Table 3: Best Hyperparameters found with the Hyperband algorithm for the CNN2D-LSTM model.

Accuracy	Loss	F1-macro	Inference time
0.71365	0.914559	0.732899	4.12555

Table 4: Performances obtained with the CNN2D-LSTM model with the validation set.

5.1.3 CNN1D-LSTM

Input layer. It can receive three types of inputs:

- Only 40 Mffc (model called *"_no_mel_no_chromagram"*).
- 40 mfcc concatted with 12 chroma features (model called *"_no_mel"*).
- 40 mfcc concatted with 12 chroma features and 128 mel spectrogram (model called *"all"*).

The hyperparameter optimization done with a preliminary trial and error and then with the Hyperband algorithm is summarized in Table 5.

Ker_size	**#C_block**	**#Ker_1**	**#Ker_seq**	**LSTM**	**Lr**	**Opt**	**dropout**
3	2	64	128	48	0.0002	Adam	0.5

Table 5: Best Hyperparameters found for CNN1D-LSTM with the Hyperband algorithm.

Models	**Accuracy**	**Loss**	**F1-macro**	**Inference time**
cnn1D_lstm_all.h5	0.606825	1.064544	0.623260	0.834419
cnn1D_lstm_no_mel.h5	0.622156	1.006559	0.639534	**0.733104**
cnn1D_lstm_no_mel_chro.h5	**0.625618**	**0.992354**	**0.641172**	0.733976

Table 6: Performances obtained with the CNN1D-LSTM model with the validation set.

The results obtained are summarized in Table 6 and analyzing them it can be seen that:

- The additional 128 Mel Spectrogram degrades the performances, as no further useful knowledge can be extracted from them, perhaps because the *MFC coefficients already contain the necessary information.*
- The chromagram features does not improve the results, maybe because they are *more useful for the analysis of music file.*

For this reason the best CNN1D-LSTM, which will be used in the fused model, is the one with **just MFCC as input**.

5.1.4 Fused model

The fused model parallelizes the two previous networks, in particular it concatenates the outputs of the two LSTMs before feeding it to the classification layer, thus producing the structure illustrated in Figure 10.

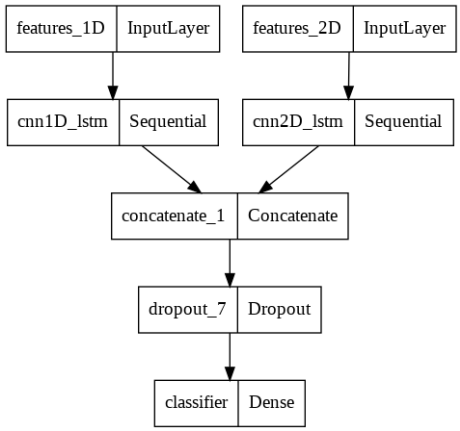


Figure 10: Fused CNN1D-2D-LSTM structure.

As can be seen from the results in Table 7 the parallelization *does not provide any useful information to the single layer perceptron* that receives the concatted outputs, on the contrary this also leads to a degradation of the performances and to an increment of the number of parameters of the model and of the inference time.

Accuracy	**Loss**	**F1-macro**	**Inference time**
0.710682	0.832017	0.725428	11.13292

Table 7: Performances obtained with the fused CNN1D-2D-LSTM model with the validation set.

5.2 LSTM-CNN

This type of architecture have the same building blocks of the CNN-LSTM described in the section 5.1.1, but the main differences are:

- Now the input are directly received by a LSTM layer, which processes the sequence of features in input, to extract contextual information from them.
- Now the LSTM layer does not only return the last state of hidden units, so a mono-dimensional output equal to the number of units, but returns the states of all the units in all the timestamps, so a bi-dimensional output (num_timestamps, units).

The second change has been applied due to an empirical evidence, in fact after many trials with the mono-dimensional output of the LSTM, I tried to return *all the states through time* of all the units obtaining an increment of 5% of accuracy. Also in this case I tried both the 1-dimensional and the 2-dimensional features in input, however I do not report the performances of the fused approach since it does not provide satisfying results.

5.2.1 1D-input LSTM-CNN

Also in this case the hyperparameters optimization was carried out with the Hyperband algorithm, obtaining the hyperparameters shown in Table 8.

Ker_size	**#C_block**	**LSTM**	**Lr**	**Opt**	**dropout**
3	1	52	0.0002	Adam	0.2

Table 8: Best Hyperparameters found for 1D-input LSTM-CNN with the Hyperband algorithm.

The results obtained are summarized in the Table 9 below and analyzing them it can be seen that:

- This time the performance differences between the different types of inputs are not significant.
- The model that receives MFCCs and Chromagram as input (*"_no_mel"*) slightly improves both the accuracy and the F1-macro.

Model	Accuracy	Loss	F1-macro	Inference time
lstm_cnn_no_mel_ch.h5	0.566271	**1.075563**	0.583135	1.506166
lstm_cnn_no_mel.h5	**0.569733**	1.084049	**0.586485**	**0.782807**
lstm_cnn_all.h5	0.562315	1.112019	0.583201	2.075511

Table 9: Performances obtained with the 1D input LSTM-CNN model with the validation set.

5.2.2 2D-input LSTM-CNN

Also in this case the hyperparameter optimization was carried out with the Hyperband algorithm, obtaining the hyperparameters shown in Table 10.

Ker_size	#C_block	LSTM	#Ker_1	#Ker_seq	Lr	Opt	dropout
3	2	64	128	256	0.0002	Adam	0

Table 10: Best Hyperparameters found for 2D-input LSTM-CNN with the Hyperband algorithm.

Accuracy	Loss	F1-macro	Inference time
0.553907	1.171493	0.567925	1.639491

Table 11: Performances obtained with the 2D input LSTM-CNN model with the validation set.

Comparing the results, in Table 11, of the 2D-input model with the ones, in Table 9, of the 1D-input model we can see that:

- The performance differences between the 1D and 2D input model **are not significant**, while in the CNN-LSTM model the 2D-input led to an improvement of the 4% in accuracy respect to the 1D input.

5.3 CNN-Transformer

The last type of models I tried have in common the same structure as the CNN-LSTM model, but this time, instead of having a LSTM to extract the global features from the feature maps obtained with the CNNs, it is used a Multi-headed Self-Attention block.

Multi-Head Self Attention. This block has the ability to attend to different positions of the input sequence and to compare each element to every other element, including itself, finally producing a new representation of the sequence.
In particular, the inputs are multiplied by three weight matrices, i.e. the Query (Q), the Key (K) and the Value (V) matrix, producing three vectors which are then combined to compute the output of the layer.
In order to focus on **different aspects of the input at the same time**, multiple heads, with different Q, K and V matrices, are used. In particular, all the outputs of the heads are joined and projected into a smaller space to keep the computational cost similar to basic single-head attention.

The architecture used contains:

- **Four convolutional block**, composed by one convolutional layer, one Elu activation function, one Batch normalization layer and one Max pooling layer. The convolutional block grows in the number of filter, having, respectively 32, 64, 128, 256 kernels.
- **A transformer block**, which is identical to the one introduced by Vaswani et al. in the well-known paper *"Attention is all you need"*.

As can be seen from the results of Table 12, the performance is slightly worse compared to the CNN2D-LSTM model, in fact the accuracy goes from 71.4% to 70.0% and the F1-macro from 73.3% to 72.1%. However the transformer model is lighter in terms of parameters, as they go from 3.8M to 2.1M, and this also affects the *inference time* that is almost **halved**.

Model	**Accuracy**	**Loss**	**F1-macro**	**Inference time**	**#Par**
cnn_tranformer.h5	0.700791	1.704515	0.72123	**2.148409**	**2.1M**
cnn_lstm_2D.h5	**0.71365**	**0.914559**	**0.732899**	4.12555	3.8M

Table 12: Performances obtained with the CNN-Transformer model compared with the CNN2D-LSTM with the validation set.

6 Models Ensemble

6.1 Aggregation of final models' predictions

All previous models **approach the task significantly differently** and, for this reason, may be more or less able to recognize a certain class or type of sample. Therefore, by **combining the predictions** made by each model, thus creating a model "ensemble", can be exploited each one's strengths and improve the coverage of the solution space.
The best model for each type was selected, in particular in the Table 13 below are summarized the performance of the 7 selected models.

Model	Accuracy	Loss	F1-macro	Inference time
cnn_lstm_fused.h5	0.710682	**0.832017**	0.725428	11.13292
cnn_lstm_1D.h5	0.625618	0.992354	0.641172	**0.733976**
cnn_lstm_2D.h5	**0.71365**	0.914559	**0.732899**	4.12555
2D_lstm_cnn.h5	0.553907	1.171493	0.567925	1.639491
1D_lstm_cnn.h5	0.569733	1.084049	0.586485	0.782807
cnn_transformer.h5	0.700791	1.704515	0.72123	2.148409

Table 13: Summary of the performances obtained with the best models, which are selected to join the ensemble.

6.1.1 Average Voting

I initially used an **unweighted average** as a baseline for choosing the weights to assign to each prediction.
With this method I was able to achieve the performance shown in Table 14.

Accuracy	F1-macro
0.7315	0.7493

Table 14: Performances obtained Ensemble using average voting.

6.1.2 Weighted Average Voting with Genetic Algorithm

To find the best combination of weights to assign to each model's prediction, I implemented a **Genetic Algorithm**, i.e. an optimization algorithm which uses *biologically inspired operators* such as *mutation, crossover* and *selection* to find a good trade-off between exploration and exploitation of the search space.

Problem definition. The problem is defined as follows:

- Each **chromosome** is composed of a gene for each model in the ensemble.
- Each **gene** represents a weight (real encoding).

- The **initial population** of candidate solutions is randomly generated and then each individual is normalized in order to ensure that the sum of each gene in a chromosome is always 1.
- As **fitness function** was used the *F1-Macro*.
- As **selection technique for recombination**, was implemented a *k-tournament selection*.
- As **crossover operator**, average crossover has been used. Each new chromosome is normalized to ensure the sum of its genes adds up to 1.
- For **mutation**, a single gene is chosen at random and is replaced with a randomly generated real number in the range [0, 1], then applying normalization.
- An **elitism mechanism** was implemented to ensure that the best solutions of each generation are maintained in the subsequent generation.

The hyperparameters of such genetic algorithm, i.e. *population size, number of generations, probability of crossover, probability of mutation, percentage of genes to use during crossover* and *the elitism percentage*, have been **chosen experimentally**.

The genetic algorithm has been run multiple times with different initial random populations. At the end, the best solution found is reported in Table 15 and led to the performance shown in Table 16.

Model	**Weight**
cnn_lstm_fused.h5	0.1864
cnn_lstm_1D.h5	0.2012
cnn_lstm_2D.h5	**0.2791**
2D_lstm_cnn.h5	0.1252
1D_lstm_cnn.h5	0.0745
cnn_transformer.h5	0.1335

Table 15: Weights assigned to each model.

Accuracy	**F1-macro**
0.7453	0.7630

Table 16: Performances obtained with the models' Ensemble using genetic algorithm to assign better weights.

6.2 Stacking Ensemble

Stacking is an ensemble technique to produce a hierarchical model with two levels:

- **Base models:** there are several models trained on the same dataset that have different capabilities or different architectures.
- **Meta model:** it receives the predictions of the base models and use them to determine the final label to assign to the sample.

I decided to adapt this technique, training an ensemble composed by seven base models, which have the same architecture, i.e. the CNN2D-LSTM, but they are all ***binary classifier*** specialized in recognizing a certain class, and a Multi-layer perceptron as meta model.

Training details. The seven models were previously trained **separately to transform the multi-class classification into a binary one**, simply substituting the current considered category with the code *"1"* and all the others with *"0"* and changing the MLP classifier. In particular, the latter has just one unit and a sigmoid activation function in the last perceptron.
Then I loaded the trained model **freezing all their parameters**, and I use them to build the ensemble.
It is important to freeze all their layers, otherwise the representations that were previously learned by the base model would get modified during training and, since the meta classifier is randomly initialized, *very large weight updates would be propagated through the network*, **effectively destroying** the representations previously learned.

The final architecture is shown in Figure 11, in which the meta classifier is composed by an MLP with the first perceptron with 512 units, followed by a dropout layer with 0.4 as dropout rate and finally a perceptron with seven units, as the number of classes, and a softmax activation function.

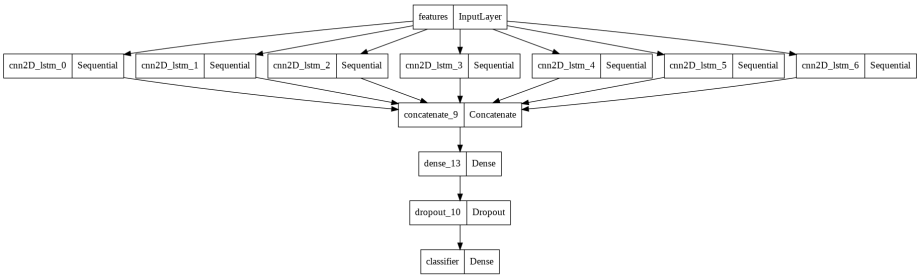


Figure 11: Stacking ensemble of CNN2D structure.

Comments on results. As shown in Table 17, this ensemble outperformed the results obtained with the CNN2D-LSTM model, even if just by a little, especially considering the F1-macro.
I think that the improvement obtained does not worth the complexity introduced, in fact this model is almost 7 times larger than the previous one and is also slower in making predictions.
So following the **KISS** principle I consider that the CNN2D-LSTM model is still the best trade-off between complexity and accuracy.

Model	**Accuracy**	**Loss**	**F1-macro**	**Inference time**
CNN2D_LSTM.h5	0.71365	**0.914559**	0.732899	**4.12555**
stacking_ensemble.h5	**0.720079**	1.174485	**0.734486**	26.680516

Table 17: Comparison between performance obtained with the CNN2D-LSTM model and the stacking ensemble with the validation set.

7 Analysis of the Results

I will analyse more deeply the results obtained with the best single model, i.e. the CNN2D-LSTM model and with the ensemble of models with average voting, where weights are chosen using the genetic algorithm.
Performance on single origin. Table 18 shows the accuracy obtained on samples that comes from the same origin, in fact, as described in section 2, the dataset is composed merging four distinct datasets.

Model	**Tess**	**Crema**	**Savee**	**Ravdess**
CNN2D-LSTM	0.9980	0.6128	0.5946	0.6716
Ensemble	**1.0**	**0.6442**	**0.6892**	**0.7562**

Table 18: Accuracy on samples from the same origin.

Looking at the table it can be seen that:

- **The TESS dataset contains the most simple samples,** in fact the ensemble perfectly recognize all its audio and it is in line also with other results on the complete TESS dataset, such as [1].
- Also the results on the RAVDESS dataset are very good, around 75%.
- There is a drop in the accuracy with the other two datasets.

Maybe possible **limits** of the proposed model are:

- It **is not able to generalize well to spoken utterances of different length and with a different structure,** in fact it loses around the 10% of accuracy on dataset which have an higher variability. While it works well with the RAVDESS and the TESS dataset, which have only three different *template* of sentence, all with a small duration, around 2 seconds.
- **Samples with different origins are predicted with the same model,** in fact having a different model for each origin would very likely lead to better performance. However, I intentionally avoided implementing this distinction, as it would only be beneficial for the purposes of the challenge, but would be useless for real data.
- **The ensemble model has an high complexity:** the ensemble is able to provide the best results, but it would be too slow and complex to be used on resource-limited device or when the response time is a strict requisite.

Performance on single classes. Table 19 and Table 20 shows, respectively, the accuracy and the F1 obtained on single classes and it can be noted that:

- **The *Surprise* class is the simplest to recognize**, even if it is the minority class, which includes about a third of the samples compared to the other classes.
- **The *Neutral* class is the one with most misclassified samples**, as can be seen from the confusion matrix in Figure 12 it is often confused with *happy* or *angry* samples.
- **The *Happy* class has the lowest F1**, in fact often *neutral* and *disgust* samples are classified as *happy*, thus reducing the **precision**, while true *happy* samples are confused almost with the same proportion with all the other classes, except for sadness and surprise, thus reducing the **recall**.

These results are in line with other studies, e.g. [8], which also shows better performances with *surprise* audios, may be because it is intrinsically simpler to recognize from the voice.
However other studies, e.g. [6], underlined that *surprise* is the most challenging emotion to detect looking only at the facial expression, while other, like *anger*, are easier to detect from visual input.
So it would be beneficial to integrate the point of strength of both the modalities **implementing a multi-modal model**.

Model	**fear**	**angry**	**happy**	**disgust**	**sadness**	**neutral**	**surprise**
CNN2D-LSTM	0.7866	0.7031	0.6373	0.6814	0.7676	0.6444	0.9099
Ensemble	**0.8076**	**0.7101**	**0.6744**	**0.7405**	**0.8014**	**0.6790**	**0.9279**

Table 19: F1 on single classes.

Model	**fear**	**angry**	**happy**	**disgust**	**sadness**	**neutral**	**surprise**
CNN2D-LSTM	0.7818	0.7301	**0.7331**	0.6147	0.7690	0.5884	0.9018
Ensemble	**0.8333**	**0.7362**	0.7147	**0.6850**	**0.7931**	**0.6495**	**0.9196**

Table 20: Accuracy on single classes.

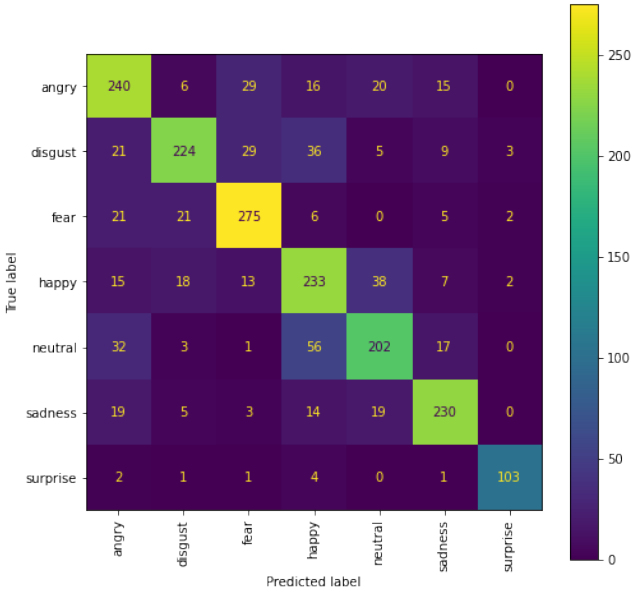


Figure 12: Confusion matrix obtained with the Ensemble model.

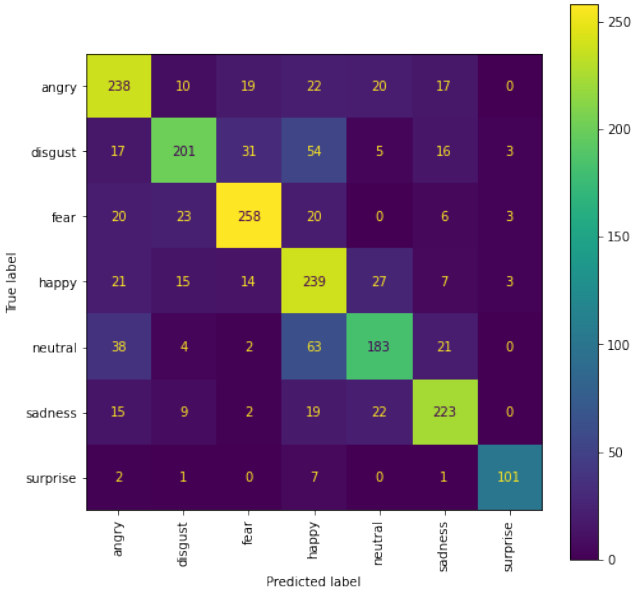


Figure 13: Confusion matrix obtained with the CNN2D-LSTM model.

8 References

1. Aggarwal, Apeksha, et al. "Two-Way Feature Extraction for Speech Emotion Recognition Using Deep Learning." Sensors 22.6 (2022): 2378.

2. Akçay, Mehmet Berkehan, and Kaya Oğuz. "Speech emotion recognition: Emotional models, databases, features, preprocessing methods, supporting modalities, and classifiers." Speech Communication 116 (2020): 56-76.

3. Chen, Mingyi, et al. "3-D convolutional recurrent neural networks with attention model for speech emotion recognition." IEEE Signal Processing Letters 25.10 (2018): 1440-1444.

4. de Pinto, Marco Giuseppe, et al. "Emotions understanding model from spoken language using deep neural networks and mel-frequency cepstral coefficients." 2020 IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS). IEEE, 2020.

5. Deng, Yaling, et al. "Gender differences in emotional response: Inconsistency between experience and expressivity." PloS one 11.6 (2016): e0158666.

6. Guarnera, Maria, et al. "Facial expressions and ability to recognize emotions from eyes or mouth in children." Europe's journal of psychology 11.2 (2015): 183.

7. Grosbras, Marie-Hélène, Paddy D. Ross, and Pascal Belin. "Categorical emotion recognition from voice improves during childhood and adolescence." Scientific reports 8.1 (2018): 1-11.

8. Kerkeni, Leila, et al. "Automatic speech emotion recognition using machine learning." Social media and machine learning. IntechOpen, 2019.

9. Latif, Siddique, et al. "Survey of deep representation learning for speech emotion recognition." IEEE Transactions on Affective Computing (2021).

10. Lee, Chul Min, et al. "Emotion recognition based on phoneme classes." Eighth international conference on spoken language processing. 2004.

11. Li, Lisha, et al. "Hyperband: A novel bandit-based approach to hyperparameter optimization." The Journal of Machine Learning Research 18.1 (2017): 6765-6816.

12. Schuller, Björn, Gerhard Rigoll, and Manfred Lang. "Hidden Markov model-based speech emotion recognition." 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03).. Vol. 2. Ieee, 2003.

13. Swain, Monorama, Aurobinda Routray, and Prithviraj Kabisatpathy. "Databases, features and classifiers for speech emotion recognition: a review." International Journal of Speech Technology 21.1 (2018): 93-120.

14. Swerts, Marc, and Emiel Krahmer. "Gender-related differences in the production and perception of emotion." Ninth Annual Conference of the International Speech Communication Association. 2008.

15. Zhao, Jianfeng, Xia Mao, and Lijiang Chen. "Speech emotion recognition using deep 1D & 2D CNN LSTM networks." Biomedical signal processing and control 47 (2019): 312-323.