Fabrice Mlili

Report : Speech Audio Recognition

1. Introduction

CREMA-D (Consortium for Research on Emotional Motivations in Adult Development) is a dataset of audio and video recordings of actors performing scripted emotional scenes. The dataset was developed by the University of Colorado Boulder and funded by the National Science Foundation.

The actors in the dataset were asked to perform a variety of emotional scenes, including scenes that depicted happiness, sadness, anger, disgust, surprise, and fear. The scenes were designed to be representative of everyday emotional situations that people might experience in their lives.

The task targeted by experiments using the CREMA-D dataset is typically related to the analysis of emotion in audio and video data. Researchers may use the dataset to develop and evaluate machine learning models for tasks such as emotion recognition, sentiment analysis, and natural language processing. These models may be used to automatically analyze the emotional content of audio or video data and extract relevant information about the emotions being expressed by the actors.

A proposed pipeline for conducting these experiments might involve the following steps:

1. Preprocessing the data: This could involve cleaning and formatting the audio and video data to make it suitable for analysis.

2. Developing machine learning models: This could involve training and testing various machine learning models using the preprocessed data, in order to identify the model that performs the best on the task at hand.

3. Evaluating the performance of the models: This could involve using a variety of evaluation metrics to assess the accuracy and reliability of the models in analyzing emotional content in the data.

4. Fine-tuning the models: This could involve adjusting the model hyperparameters and other settings in order to further improve the performance of the model.

I decided to use the CREMA-D dataset through these experiments because it provides a large and diverse set of data that can be used to train and test machine learning models. The dataset includes a wide range of emotional expressions and situations, which can help to ensure that the models are robust and can generalize well to other speeches dataset.

In digital audio, an audio signal is typically represented as a sequence of numbers, with each number representing the amplitude of the signal at a particular point in time. The sequence of numbers can be thought of as a waveform, with the numbers representing the heights of the waveform at different points in time.

When an audio signal is digitized and stored in a computer, the sequence of numbers representing the signal is typically saved in a file format such as MP3, WAV, or AIFF. The file format specifies how the numbers are encoded and stored, and how they should be decoded and played back when the audio is played.

In the case of the CREMA-D dataset, the audio data consists of recorded audio of actors speaking and performing emotional scenes. Each audio file in the dataset is a sequence of numbers representing the amplitude of the audio signal at different points in time. These numbers can be used to reconstruct the original audio signal and play back the audio when the file is played.

To realize this experiments, I have chosen to use firstly a simple neural network using LSTM layers and creating features from audio sequences with MFCC. MFCC stands for Mel-Frequency Cepstral Coefficients. It is a feature representation commonly used in speech and audio processing tasks. It is derived from the spectrogram of a signal, which is a representation of the frequency content of a signal over time. The MFCCs are derived from the log-magnitude spectrum of a signal, and they capture the spectral envelope of the signal. They are typically used as input features for tasks such as speech recognition, speaker recognition, and music genre classification.

Then, as the previous model doesn't give a good accuracy, I decided to use a more complex model, basically a CNN as we have spectral data. This time the pre-processing is more consequent. I use audio under 3 special types : normal, noised and stretched + pitched. Moreover, features are created under 5 transformations for each audio sample : ZCR, STFT, MFCC, RMS and Mel spectrogram. The zero-crossing rate (ZCR) measures the number of times the audio signal crosses the zero line per second. The chroma short-time Fourier transform (STFT) represents the energy of the audio signal at different pitches. The root mean square (RMS) is a measure of the power of the audio signal. Finally, The Mel spectrogram represents the spectral power of the audio signal across different frequency bands.

About pre-processing, there is no need to clean data because this is a cleaned data commonly used in research.

2. Experimental methodology

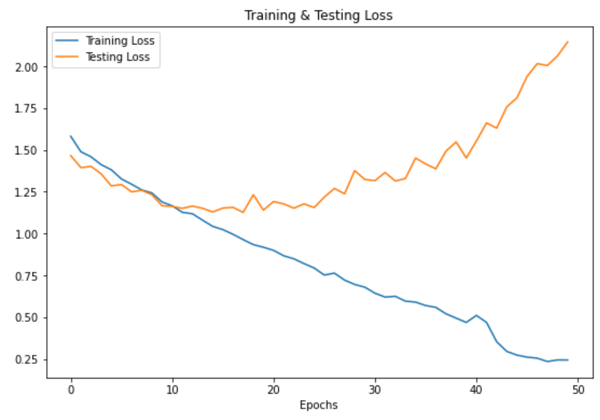
The choice done on the way to test performance of models are very classical. As a metrics evaluated, I took the accuracy. About loss, I used 'sparse_categorical_crossentropy' avoiding to set up a oneHotEncoding to categorise the labels. I didn't use shuffle split because each part (train, test and val) are already inside the structure of the loaded dataset. Finally, the selected numbers of neurons are are purely abitrary and their architecture derive from very often used models. Nota Bene : a StandardScaler() function is used with the CNN model in order for it to make easy data comparison and thus potentially reduce the time of reach the max accuracy.

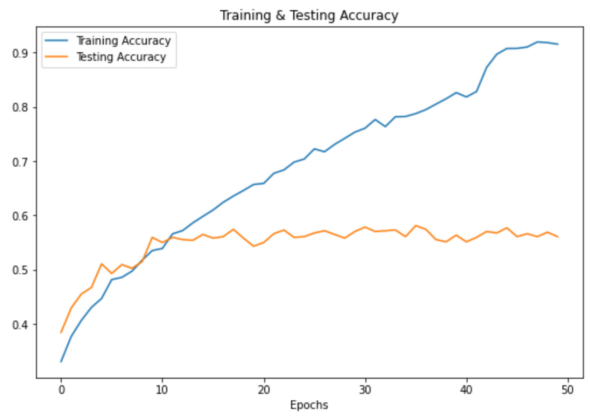
To get more information and get in details, here is the notebook of the experiment :

https://github.com/fabricemlili/Supervised-learning/blob/main/Project/final_project.ipynb

3. Results and discussion

Now, the main relevant results are going to be discuss. The LSTM model seems to reach its maximum accuracy after 10 epochs. More of 50 % of prediction that it makes are correct. Let’s plot the graph to have a better visualisation.

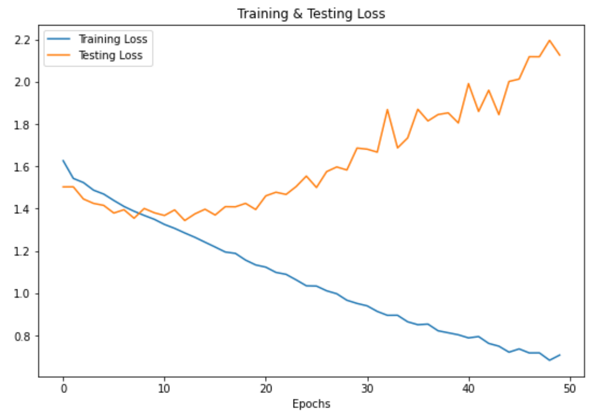


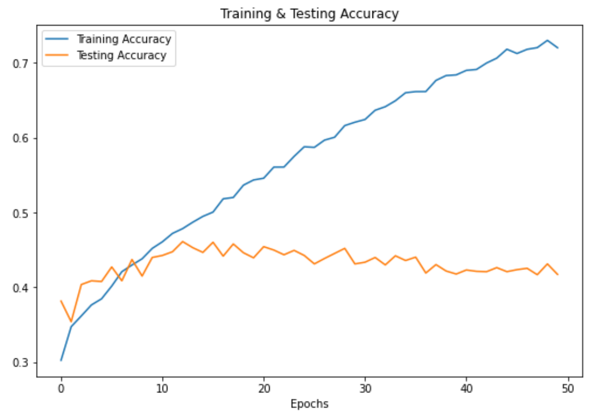


The maximum accuracy is reached from 10 epochs. Beyond it, the training is overfitting. But unfortunately, the accuracy isn’t as nice as we could expected. So let’s try the next model to make better.

When you need to build an AI with a good ability in communication with humans, you need to have a model with a very high accuracy in emotion recognition. Knowing that the sound can be modelised as an image, I chose to completely change the model using a convolutional neural network (CNN). Basically, I need to know if this way of thinking is the better for the task.

I obtained an accuracy only slightly more than 40 % with the CNN. Let’s plot to see what is its behavior :





The crossing between the two curves show that starting from the 5th the model is overfitting. There are maybe several reasons to as the difference between the two ways of process each traning with the two different models, but the result of CNN surprised me as the model is often used with the dataset CREMA-D, the accuracy reaching more than 60 %.

Difficulties encountered : Load the dataset, understand that I don't really need librosa.load() and thus no need to download the data, find the matching sample_rate without which I could not have predicted anything, problems of memories and lack of GPU (all used on google colab).

4. Conclusion and next steps

Finally, as we have seen the best model and the one to use for the task is .

Here is a non-exhaustive list about what I had could be better :

1. Use tf.keras.layers.BatchNormalization layers to normalize the activations of the layers. This can help improve the performance of the model and speed up the training process.

2. Use the tf.keras.callbacks.EarlyStopping callback to stop the training early if the validation loss does not improve after a certain number of epochs. This can prevent overfitting and improve the generalization of the model.

3. Use the tf.keras.callbacks.ModelCheckpoint callback to save the best model weights during training. This can help ensure that the model does not lose the best weights if the training is interrupted.

4. Use the tf.keras.callbacks.TensorBoard callback to visualize the training and validation loss and metrics in TensorBoard. This can help identify trends and patterns in the data and improve the model.

5. Use the tf.keras.optimizers.Adam optimizer to train the model. This optimizer is generally a good choice for many tasks and can help improve the performance of the model.

6. Use the tf.keras.losses.SparseCategoricalCrossentropy loss function for training the model. This is a good choice for multi-class classification tasks.

7. Use the tf.keras.metrics.SparseCategoricalAccuracy metric to evaluate the performance of the model. This metric is a good choice for multi-class classification tasks.

8. Use the tf.keras.layers.GlobalAveragePooling1D or tf.keras.layers.GlobalMaxPooling1D layers to aggregate the features from the convolutional layers before the fully connected layer. This can help reduce the number of parameters in the model and improve its performance.

9. Use the tf.keras.layers.Conv1D layer with a larger kernel size and/or a larger number of filters to extract more meaningful features from the audio data. This can help improve the performance of the model.

10. Use the tf.keras.layers.MaxPooling1D layer with a larger pool size to down-sample the feature maps and reduce the dimensionality of the data. This can help improve the performance and efficiency of the model.

11. Use the tf.keras.layers.Bidirectional layer with an LSTM or GRU layer to capture the temporal dependencies in the data.

12. Consider using a more complex model, such as a stacked LSTM or a hybrid model combining CNN and RNN layers, to improve the performance of the model.

13. Use the tf.keras.layers.Concatenate layer to combine the features from different layers or models to improve the performance of the model.

Also, in the case of real-time inference or embedded inference, the time of compute need to be more efficient. One of the solution would be to use with tf.device("/cpu:0"): or with tf.device("/gpu:0"): to specify the device to be used for running TensorFlow operations. This can help improve the performance of the model. Another solution would be to train on a bigger dataset to no more have need to train and only focus on the predictions made on the new data.

4. Reference

https://www.kaggle.com/datasets/ejlok1/cremad