Intent Detection in Speech Utterance

Homayoun Afshari
Politecnico di Torino
homayoun.afshari@studenti.polito.it

***Abstract*—In this project, we design a method to extract possible *intents* from speech utterances. Each utterance has an audio signal labeled with information such as language or fluency level. Our method is twofold. First, we use a preprocessing pipeline that extracts Mel Frequency Cepstral Coefficients (MFCCs) from each audio signal as its features. Then, we train a deep neural network that combines the MFCCs of an utterance with its so-called labels and extracts its underlying *intent*. Also, to compensate for the imbalanced distribution of the provided dataset and to improve our method's generalizability, we utilize data augmentation. Based on the evaluation results, our proposed approach offers acceptable performance over the baseline.**

I. Problem Overview

The problem addressed in this project regards the classification of speech utterances based on their *intents*. The provided dataset has two parts. The first one, *development*, contains 9,854 utterances provided for development. The second part, *evaluation*, has 1,456 utterances used for evaluation. In both parts, in addition to an audio signal, each speech utterance is accompanied by some labels. They are *gender*, *langFirst*, *langSecond*, *age*, and *fluency*.[1] The *development* dataset also provides two more: *action* and *object*. The combination of these two labels is the *intent* that is embedded in each utterance. As presented in "Fig. 1", our project aims to discover this *intent* utilizing both audio data and the mentioned labels.

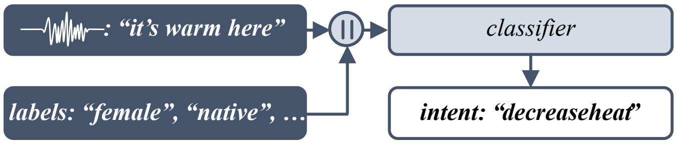

Fig. 1. Summary of the requested task, where || denotes concatenation.

As illustrated in "Fig. 2", the *development* dataset is poorly balanced regarding the possible *intents*. This issue results in under-classifying infrequent *intents* while favoring frequent ones, which leads to performance degradation [1]. For this reason, our idea is to use data augmentation proportionally, i.e., we augment infrequent *intents* more than frequent ones during preprocessing. Although this synthesized data is not actual, we can consider data augmentation as a reasonable attempt for approximately balancing our dataset [2].

Apart from a tiny portion sampled at $22.05kHz$, most of the audio signals in both *development* and *evaluation* datasets have a sampling rate of $16kHz$. This rate enables capturing frequency content slightly above $8kHz$ [3]. Compared to the human ear's frequency range (between $20Hz$ to $20kHz$), having audio data with such a limit is acceptable but could be better [4]. Nonetheless, research has shown that the human voice hardly reaches $8kHz$ in frequency [4]. Therefore, since the provided audio signals only contain human voice, we are assured that each one contains all necessary information that is finally interpreted as its underlying *intent*. However, to unify all the audio signals, we load them into the working space with a default sampling rate of $16kHz$. Note that all the audio files are in WAV format with a quantization depth of $16bit$, which is an acceptable amount for the bit depth as it is traditionally used in the Compact-Disk (CD) technology [5].

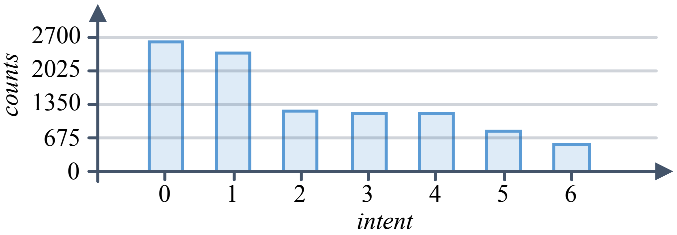

Fig. 2. Distribution of *intents* in the dataset. The numbers 0 to 6 respectively denote "increasevolume", "decreasevolume", "increaseheat", "decreaseheat", "change languagenone", "activatemusic", and "deactivatelights".

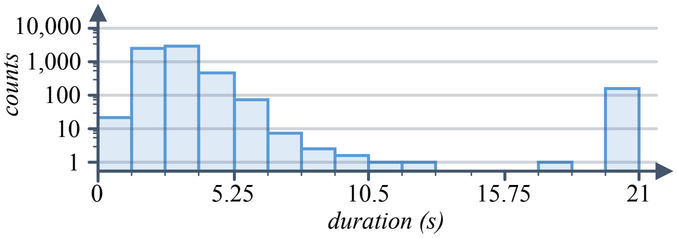

Fig. 3. Histogram of durations in logarithmic scale.

Moreover, the available audio signals have different durations, and most contain silent samples at the beginning or end. As illustrated in "Fig. 3", most of them are shorter than 5s. Also, as illustrated in "Fig. 4", most of them have a considerable amount of silence density, which is the number of silent samples at the beginning and end of an audio signal divided by the total number of samples that it contains. Since the proposed model accepts data in tensors, variation in duration is a problem we must solve in preprocessing. In addition, since silent samples contain no valuable information

[1] Respectively termed "gender", "First Language spoken", "Current language used for work/school", "ageRange", and "Self-reported fluency level" in the original dataset.

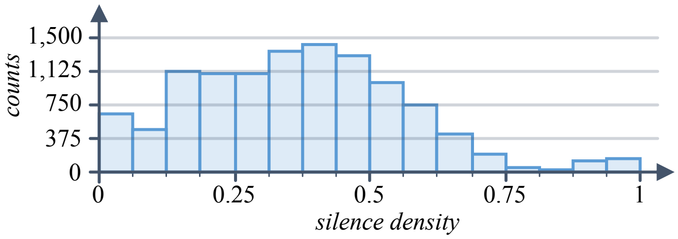

Fig. 4. Histogram of silence density.

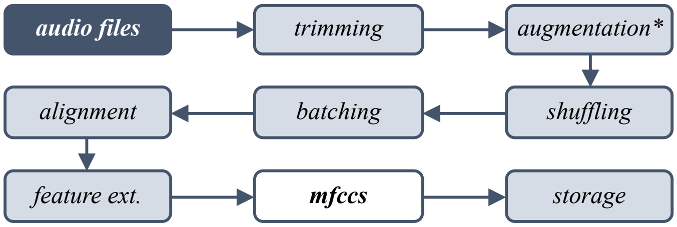

Fig. 5. Preprocessing pipeline. Note that the augmentation block is only active during preprocessing of the development dataset.

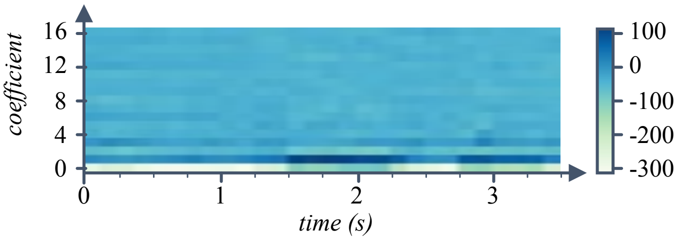

Fig. 6. Heatmap of MFCCs of the utterance “turn down the heat”. The audio is split into 28 time frames, and 17 MFCCs are extracted from each frame.

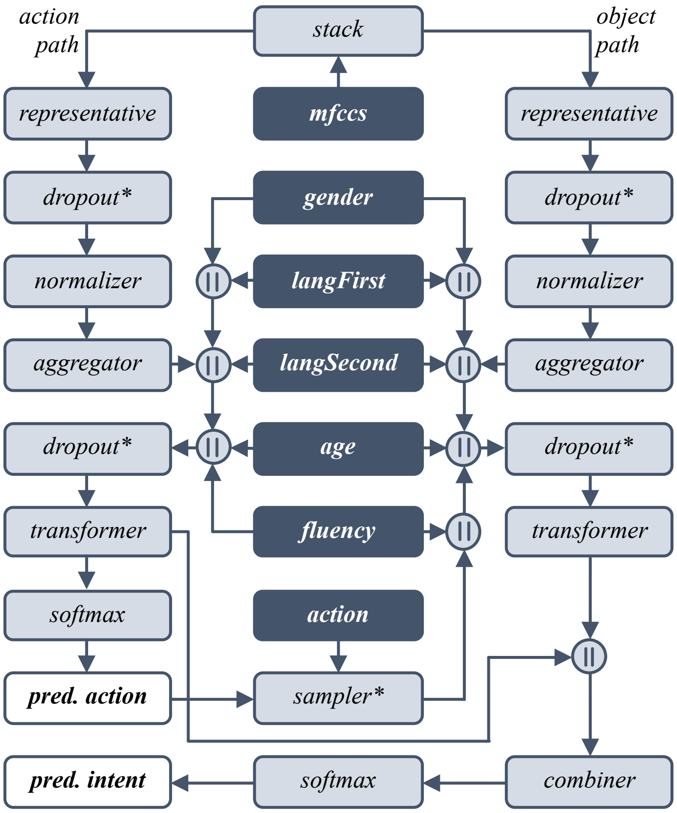

Fig. 7. The architecture of proposed method. Note that the dropout layers are only activated during training. Also, the sampler only selects the predicted action during evaluation.

related to our *intent* discovery task, we also remove this redundancy during preprocessing.

II. Proposed Approach

A. Preprocessing

Before classification, we must clean our dataset. The process is depicted in “Fig. 5”. Consequently, the first step is trimming, i.e., eliminating silent samples from the beginning and end of the audio signals. After that, they are ready for augmentation, which is only applied to the *development* dataset. For this purpose, we employ random pitch shifting, speed changing, time shifting, and noise injection [2]. As mentioned, extending our “development” dataset using this synthesized data is an effort to level the imbalance distribution of the *intents*. However, this also enhances the model by preparing it for low-quality versions of the training data, which reduces overfitting [2].

The third step in the pipeline is shuffling, which is done in both *development* and *evaluation* datasets. However, this step is particularly essential for the *development* dataset because it leads to a perfectly random mixture of actual and augmented data. The next step is putting audio signals in batches. This is done because the proposed model is trained by iterating over batches of tensors. After this step, the audio signals of each batch are aligned, which is a prerequisite for the next step in the pipeline and the proposed model. Note that, without batching, we needed to align all the audio signals in a huge tensor with a considerable amount of padded silent samples due to the difference in durations of the audio signals. Now, with data in batches, we only align the ones that fall in the same batch. This is because the proposed model is almost not sensitive to the dimensions of the tensors that it receives.

At the final step, batched audio signals enter the feature extraction module. Here, we use the well-known MFCC technique to extract temporal and spectral characteristics from each audio signal [6]. For this purpose, we first divide each audio signal into multiple time frames. Then, we calculate the power spectrum of each time frame and map it to the “mel” scale, which resembles how the human ear perceives sound [6]. Finally, we apply the discrete cosine transform to the logarithm of the scaled power spectrum and obtain a group of coefficients called MFCCs [6]. As demonstrated in “Fig. 6”, the result is a matrix where each column contains MFCCs of a time frame. Through this technique, not only can we preserve each signal’s temporal sequence, but we also extract its spectral content. We then use MFCSs as inputs to our model. We also store MFCCs of each batch for further use.

B. Model Selection

The basic idea of our model is adapted from [7]. The original method uses Recurrent Neural Network (RNN) to classify speech utterances only using their audio data. Our contribution is to incorporate their labels in the classification as well. The method is depicted in "Fig. 7". First, each batch of the preprocessed audio data enters a stack of Bidirectional Long-Short Term Memory (Bi-LSTM) units. We chose LSTM because our data is sequential. It is also bidirectional because the *intent* of a speech utterance can flow both from past to future and vice versa. [2] Furthermore, LSTMs are stacked to mine deeper into each batch for complicated patterns [7].

At the next layer, the data moves forward in the network through two parallel paths, one for extracting its underlying *action* and one for the *object*. Each path leads to a layer of again Bi-LSTM unit. We call these units representatives because their responsibility is to search the delivered data of the stack for patterns related to *action* or *object* specifically. At the output of this layer, we use a dropout layer. It replaces a group of random elements in its input tensor with zero. Since this situation is similar to when some neurons are dropped out of the network, the dropout layer acts as a regularization technique forcing the model to learn more robust patterns that are useful in making predictions. It also helps reduce overfitting and improves the generalization of the network [8]. These layers are only active during training. The next layer is batch normalization. It estimates the mean and variance of its input and normalizes it. This measure reduces the internal covariance shift caused by changes in the distribution of the audio data having passed through the previous layers [9].

The data is now ready for aggregation. At this layer, the temporal dimension of the audio data is removed through max-pooling. This enables us to join the output to its corresponding labels. The result is passed through another dropout layer for more regularization. At the next layer, we use a fully connected linear neural network to prepare the combined data for classification. We call it a transformer because it acts as a feature reduction module and transforms its input into the dimensions required by the following layers. From this layer, the paths become different. On the *action* path, a softmax layer classifies the *action* embedded in the output of its corresponding transformer. The predicted *action* is then delivered to the sampler. This module has a vital role. During training, it randomly selects between the real *action* or the predicted one, while the probability of choosing the real *action* decreases gradually. The output is a new label joined to other labels on the object path. Note that the sampler layer only uses the predicted action during evaluation.

The above method is called scheduled sampling [10]. It acts upon the fact that the underlying *action* and *object* in a speech utterance are correlated. To utilize this correlation in classification, while we predict the *action* independently, we make predictions about the *object* conditionally [7]. Finally, we join the transformers' outputs and deliver them to a combiner layer using another fully connected linear neural network. The next layer is a softmax layer that predicts the *intents* embedded in the input batch.

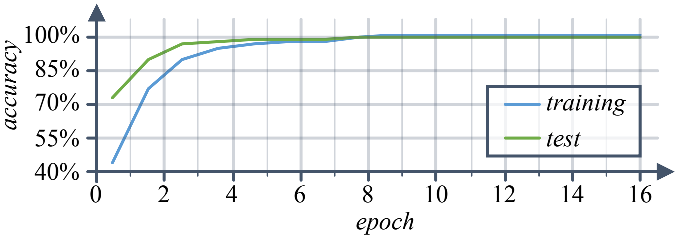

Fig. 8. The accuracy versus epoch during training and test.

C. Hyper-Parameters Tuning

In our model, we have a large number of hyper-parameters. In the preprocessing pipeline, there is the number of MFCCs, the frame size, the augmentation factor (f),[3] and the batch size (s_B). In the network, we have the number of hidden layers in the stack (n_H) and the output size of the LSTM units (s_L). Tuning this number of hyper-parameters via a full grid search is highly time-consuming. Therefore, we limit our options. First, as done in [7], we set the number of MFCCs equal to 40 and the frame size to 512. Now, we only tune the variables provided in "Tab. I", where augmentation time (t_A), training time (t_T), and test accuracy (a_T) are used as our evaluation metrics using an 80-20 training-test split on the *development* dataset. Our method is to fix f and s_B and try multiple values for n_H and s_L. It is done until we achieve an acceptable result that cannot be repeated by further changing the values. Then, considering the previous experience, we increase s_B and repeat the process. After finding the best possible combination for s_B, n_H and s_L, we stop and repeat the search after increasing f. For instance, we set $f = 0$ and $s_B = 64$ and observe that the best result is achieved when $n_H = 3$ and $s_L = 512$. Now, we set $s_B = 128$ and search for new options based on our experience. We continue until we find a satisfying combination for s_B, n_H, and s_L. After that, we repeat the search with higher values of $f = 0$. This is how "Tab. I" is structured.

III. Results

According to "Tab. I", data augmentation has a noticeable positive effect on accuracy. Therefore, selecting $f = 4$, $s_B = 128$, $n_H = 3$, and $s_L = 512$ is probably a good option for the hyper-parameters because it delivers the best accuracy. Nevertheless, due to its relatively high augmentation and training time, we change our option to $f = 3$, $s_B = 256$, $n_H = 3$, and $s_L = 512$, which has a slightly lower accuracy but shorter augmentation and training time. That being the case, we obtain 'Fig. 8". This figure is the result of testing the model using the *development* dataset. We can see that the accuracy converges to almost 99.2% after almost 6 epochs. We

[2] For instance, the flow "decrease volume" in the utterance "lower the music" is from past to future, but it is reversed in the utterance "it is loud".

[3] The factor by which the "development" dataset is augmented. For example, $f = 2$ means its size is tripled. Accordingly, $f = 0$ implies no augmentation.

TABLE I
HYPER-PARAMETERS OF THE PROPOSED METHOD

f	s_B	n_H	s_L	t_A	t_T	a_T
0	64	3	512	$0s$	$95s$	93.92%
		4	256	$0s$	$98s$	92.82%
		4	512	$0s$	$109s$	92.56%
		5	256	$0s$	$118s$	91.02%
	128	3	512	$0s$	$46s$	93.82%
		4	256	$0s$	$50s$	90.30%
		4	512	$0s$	$67s$	92.48%
	256	3	512	$0s$	$32s$	89.93%
		4	512	$0s$	$37s$	88.09%
1	64	3	512	$201s$	$184s$	97.09%
		4	512	$205s$	$207s$	96.69%
	128	3	512	$195s$	$93s$	97.63%
		4	512	$205s$	$101s$	97.59%
	256	3	512	$199s$	$72s$	95.62%
2	128	3	512	$261s$	$136s$	98.15%
	256	3	512	$249s$	$108s$	98.44%
	512	3	512	$237s$	$95s$	97.45%
3	128	3	512	$632s$	$183s$	99.14%
	256	3	512	$646s$	$143s$	99.12%
4	128	3	512	$846s$	$227s$	99.54%
	256	3	512	$832s$	$178s$	99.30%

can also conclude that our precautionary measures to avoid overfitting have been effective, and the model provides good generalizability. This was especially helpful in applying the model to the *evaluation* dataset. Our best score during the evaluation was 99.3%.

The above result is obtained by implementing the model in Spyder v5.2.2 using Python v3.9.15. Among the many useful libraries we utilized, we benefited the most from Torch. In addition to the powerful tools this library provides for machine learning projects, our reason for selecting it is its ability to transfer calculations on Graphical Processing Units (GPUs) to improve performance. For this project, we used GeForce RTX 3050 TI supporting CUDA v11.

IV. DISCUSSION

Considering the obtained accuracy, our proposed *intent* classifier shows excellent performance. However, while further improvement in the accuracy appears unnecessary, there is still room for improving the augmentation and training time. It is clear that without data augmentation, this result was not achievable. Nevertheless, we can still analyze how data augmentation affects the classification of each *intent* separately. In this respect, we could use the f1-measure to decide how unbalanced we perform data augmentation so that the result remains unchanged. Through this, we might find lower augmentation factors that obtain the same result in a shorter time. Moreover, the stacked BiLSTM in the network is connected sequentially, i.e., the output of each hidden layer is the input of its following layer. However, we could employ a residual architecture to check whether it is possible to train the model more quickly. Note that in the residual architecture, the input of each hidden layer can be a combination of the outputs of multiple previous layers.

REFERENCES

[1] G. Zarzar, "Training models on imbalanced data," Oct 2020. [Online]. Available: https://towardsdatascience.com/training-models-on-imbalanced-data-561fa3f842b5

[2] V. Velardo, "Audio data augmentation techniques: The theory," Jan 2022. [Online]. Available: https://www.youtube.com/watch?v=bm1cQfb_pLA

[3] R. Keim, "The nyquist–shannon theorem: Understanding sampled systems - technical articles," May 2020. [Online]. Available: https://www.allaboutcircuits.com/technical-articles/nyquist-shannon-theorem-understanding-sampled-systems/

[4] B. B. Monson, E. J. Hunter, A. J. Lotto, and B. H. Story, "The perceptual significance of high-frequency energy in the human voice," *Frontiers in psychology*, vol. 5, p. 587, 2014.

[5] R. Triggs, "What you think you know about bit-depth is probably wrong," Jul 2021. [Online]. Available: https://www.soundguys.com/audio-bit-depth-explained-23706/

[6] M. H. Shirali-Shahreza and S. Shirali-Shahreza, "Effect of mfcc normalization on vector quantization based speaker identification," in *The 10th IEEE International Symposium on Signal Processing and Information Technology*. IEEE, 2010, pp. 250–253.

[7] E. Palogiannidi, I. Gkinis, G. Mastrapas, P. Mizera, and T. Stafylakis, "End-to-end architectures for asr-free spoken language understanding," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 7974–7978.

[8] J. Brownlee, "A gentle introduction to dropout for regularizing deep neural networks," Aug 2019. [Online]. Available: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

[9] ——, "A gentle introduction to batch normalization for deep neural networks," Dec 2019. [Online]. Available: https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/

[10] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," *Advances in neural information processing systems*, vol. 28, 2015.