Different deep learning approaches for speech command recognition: attention mechanisms, residual connections, and a transformer's encoder

Camilo Betancourt Nieto

***Abstract*—This work explores different network architectures for keyword spotting (KWS), which has applications in technologies such as smart home devices, automotive systems, and virtual assistants like Siri and Alexa. Since many of the devices that implement these systems demand small memory footprints and low computational requirements, I focused on models that comply with these characteristics. I devised an architecture that combines convolutional operations, recurrent neural networks with an attention mechanism, and deep residual networks. Following a common feature extraction module, I compared this design against two models: the powerful `AttRNN` from the literature, and a transformer's encoder adapted for speech command recognition, which I also developed. Despite its reduced parameter count, one version of my hybrid approach outperformed the `AttRNN` in terms of accuracy and F-score, although the latter model exhibited shorter training times. Additionally, two other variants of my proposed approach demonstrated competitive performance, comparable to the one in the literature, with 20% fewer parameters.**

***Index Terms*—Speech recognition, keyword spotting, attention mechanism, deep residual networks, transformers.**

I. Introduction

The keyword spotting task plays a key role in human-computer interfaces (HCI) like Google assistant, Apple's Siri, Amazon's Alexa or Microsoft's Cortana, which rely on speech recognition systems [1] [2]. An interaction that requires full automatic speech recognition would typically be performed in the cloud and would require transferring the audio recordings from the device, potentially raising privacy concerns. On the other hand, the KWS systems can run locally on the devices to detect a relatively small set of simple commands such as ”on”, ”off”, ”stop”, ”go”, etc. that trigger actions on the device, for example enabling a full automatic speech recognition interaction [3].

Since the KWS systems usually run on mobile devices, it is desirable that the implemented models have a small memory footprint and require low computational power [1] [3]. With that in mind, there have been several examples in research that tackle this problem effectively by using different artificial neural network (ANN) architectures. Some approaches to this problem include convolutional neural networks (CNNs) [1], deep residual networks (ResNets) [3], and recurrent neural networks (RNNs) with attention mechanisms [2], however, to the best of my knowledge, no architecture has combined these approaches for the keyword spotting task.

In this work, I propose a hybrid approach, incorporating elements from the aforementioned methodologies. I utilized an RNN with attention mechanisms (`AttRNN`) [2] as a benchmark, while specifically aiming to develop lightweight models that could achieve comparable performance in terms of accuracy, F-score, and prediction time. Additionally, to test this mixture against a different type of architecture, I also implemented a model that consists of a compact adaptation of the transformer's architecture and its self-attention for classification [4] [5]. This comparison allows for the assessment of which architectures work best for this problem, opening up the possibility of explaining the reasons in further research, and serving as an example for other implementations tackling similar problems, for example within the domain of audio analysis and signal processing. Moreover, the best-performing models are comparable to state-of-the-art results and could potentially be applied in real-world situations.

Therefore, the contributions of this work could be summarized as:

1) Definition of a novel model architecture that combines features that have previously been proven effective for this kind of task;
2) Definition of a model based on the transformer architecture, proven effective in other types of scenarios, but specifically adapted for keyword spotting;
3) A comparison between a benchmark (the `AttRNN`) and these two proposed architectures in terms of model weight (number of parameters), accuracy, F-score, training time, and prediction time.

This report is structured as follows. I describe previous approaches in the literature in Section II. The processing methodology and the data preparation are presented in Sections III and IV, respectively. The proposed learning architectures are described in detail in Section V, and their performance evaluation is carried out in Section VI. In Section VII, I provide the concluding remarks.

II. Related Work

Different architectures of CNNs were devised by Sanaith and Parada [1] to obtain small memory footprint models for keyword spotting. In that paper, they applied a 40-dimensional log-mel filterbank feature extraction module and grouped some frames to obtain a two-dimensional representation of the input (with time and frequency). Then, they tried different network configurations and types of filters -in both dimensions- to limit the amount of parameters and multiplies of their models.

Apart from this, to define an architecture for the same task, Tang and Lin [3] used deep residual learning techniques based on ResNets [6] in addition to convolutional layers with two-dimensional dilated convolutions [7]. In doing so, they were able to adjust the extent of their networks to achieve the desired balance between model footprint and performance, obtaining a variant that outperforms Google's previously best compact CNN networks. An aspect worth mentioning about that paper is that, instead of using log-mel filterbanks, they used Mel-Frequency Cepstrum Coefficients (MFCCs) as input for the networks.

A different kind or approach was made by Coimbra de Andrade et al. [2], as they designed the `AttRNN`, an architecture based on RNNs and the attention mechanism [8] [4]. In this case, they started from an 80-band Mel-scale spectrogram instead of the MFCCs and applied convolutions along the time dimension only before passing the result to the layers with recurrent units. This work achieved a better performance than the previous examples with less trainable parameters. However, these authors did not use the deep residual learning techniques described above, allowing for the possibility to test how these two methods would work together.

Another paper worth mentioning is by Chorowski et al. [9], where they applied adapted attention mechanisms for speech recognition, though intended for the phoneme recognition task. Yet, one of the adaptations that the authors included is relevant for this work. After making certain modifications to the attention mechanism, they observed that the model's performance worsened for short utterances and identified the need to add a smoothing effect to the attention focus. This effect was achieved by replacing the unbounded exponential function of the softmax $(\alpha_i = e^{z_i}/\sum_{j=1}^{n} e^{z_j})$ with the bounded logistic sigmoid, such that $a_i = \sigma(z_i)/\sum_{j=1}^{n} \sigma(z_j)$. This result invites to test how a similar smoothing effect could perform on a keyword spotting task.

On a different note, Song et al. [5] proposed an architecture based on the transformer, adapted for using clinical time-series data in multiple diagnosis tasks. They employed an input embedding, positional encoding, and an attention module similar to those in the transformer to learn representations of the sequential data. To generate a concise code for the entire sequence, they used a dense interpolation module, which was then passed to the final layer for classification or regression. Given that their study allows for classification, their architecture could also be adapted for the speech command recognition tasks addressed in this work.

As the source of audio data for training and evaluating the effectiveness of the proposed networks, I utilized the 12-command variation of Google's Speech Commands V2 dataset [10]. This dataset contains short audio clips of common spoken words with the following target labels: ”Yes”; ”No”; ”Up”; ”Down”; ”Left”; ”Right”; ”On”; ”Off”; ”Stop”; ”Go”; a label for ”silence,” representing background noise; and a label for ”unknown word,” indicating utterances that do not match the other 10 keywords. Various versions of this dataset have been used as a standard benchmark for evaluating speech recognition models in limited-vocabulary scenarios, including some of the previously referenced papers.

III. Processing Pipeline

In this work, after a short preprocessing stage of the raw data, the audio signals were transformed by extracting the MFCCs for audio frames of 25 ms. This step can be seen as a feature extractor to obtain a more compact representation of the input before passing it to the networks. This step was performed with three objectives in mind. First, by reducing the size of the network inputs, the number of parameters could also be reduced, resulting in lighter models. Second, as the MFCCs procedure yields a two-dimensional representation of the inputs (the time dimension and the MFCCs dimension), I could perform different types of convolutions, for example the ones described in the previous section. Lastly, since I use the `AttRNN` as a benchmark to devise and test my models, but the paper that introduced this approach utilized a different feature extraction technique, I tested that architecture with MFCCs to make the results comparable.

Once the MFCCs are obtained, they are passed to each of the neural networks. More specifically, three types of models are considered in this report. The first one is simply the `AttRNN` without any modifications. The second type is the hybrid approach that mixes deep residual learning techniques and RNNs with the attention mechanism. From now on, this kind of models are referred to as `ResAttRNN`. Finally, the third type of models is based on the transformer's architecture, but adapted for a classification task. These models are labeled as `TransfClassifier`.

The `ResAttRNN` architecture is inspired by the `AttRNN`, but it replaces the first convolutional layers with residual blocks that resemble those used by Tang and Lin [3]. The idea is to obtain a deeper -and hopefully better- hidden representation of the sequential data before passing it to the bidirectional recurrent units. An additional modification that was tested, was to apply a *bounding function* before the attention mechanism as an attempt to achieve a similar smoothing effect as the one described in the previous section. From that core idea, different configurations were empirically tested in order to check the performance of the network when the number of parameters was changed and when the attention smoother was included.

On the other hand, the `TransfClassifier` is based on the transformer's architecture [4] and the paper by Song et al. [5]. More specifically, starting directly from the sequence of MFCCs, it takes the encoder part of the transformer to obtain a contextualized representation of the data. Next, a dense interpolation layer [5] is added before the last classification layer. Similar to what I did with the `ResAttRNN`, I experimented with different configurations of the encoder, varying the number of parameters and aiming for lightweight networks.

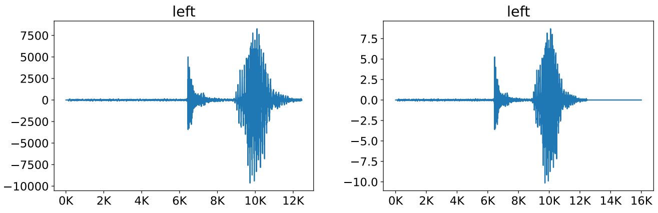
IV. Signals and Features

The Speech Commands dataset was directly downloaded from the TensorFlow repository [11] [12], which allows loading the data with the preestablished balanced splits for training, validation, and test sets. These sets comprise 85511, 10102, and 4890 observations, respectively. Each observation is an audio signal with a 16 kHz sample rate and a maximum length of 1 second.

As some of the audio clips were shorter than one second, the first preprocessing step was to pad the missing information with zeros. This ensured that all the processed examples had the same length. After that, while still working with audio signals, I normalized them by using a mean-and-variance approach across samples. The idea behind this was to maintain a balanced level among observations. The result after these two steps is shown in Fig. 1.

From that point, the data was transformed into MFCCs. In a nutshell, this technique summarizes the audio information by segmenting it into short frames (25 ms with a 10 ms time step in my case) and computing key features for each frame. These coefficients should effectively represent the linguistic content of the audio signal [13], facilitating subsequent classification with neural networks. In this study, the MFCCs were computed using the library implemented by Lyons et al. [14], which allows to retain 12 cepstral coefficients plus an additional feature corresponding to the $\log$ of the total frame energy, resulting in 13 features for each frame.

Even though the networks include normalization layers, as described in Sec. V, I wanted to ensure that the data was adequately normalized before passing it to the models to facilitate convergence [15]. Therefore, after obtaining the MFCCs, I applied Cepstral Mean and Variance Normalization (CMVN) [16] [17] using a per-sample approach, which involved taking the mean and variance of each feature across the different frames of each sample. The result of this process is illustrated in Fig. 2.



(a) Raw audio

(b) Processed audio

Fig. 1: Audio padding and normalization

After all these feature extraction steps, the final input for the neural networks was obtained. Each observation consists of 99 frames (the time dimension) and 13 MFCCs (the features at each time step).

V. Learning Framework

The AttRNN was implemented exactly as described by the authors [2], without any design changes. The only difference lies in the input that the network receives, as I used a different feature extraction approach. Consequently, the number of parameters and the performance of the network differ from the results obtained in their publication. This procedure is done to address the effectiveness of the networks only. My implementation resulted in a model with **166K** trainable parameters. For the architectural details, please refer to the original paper.

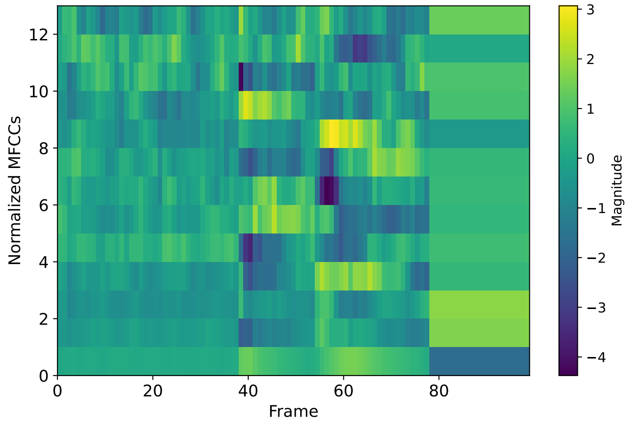


Fig. 2: Result of the feature extraction module

A. The ResAttRNN architecture

As previously mentioned, the ResAttRNN architecture builds upon the AttRNN design, but modifies various aspects of it.

The first and most significant modification is at the very beginning of the network. The AttRNN starts with a set of convolutions in the time dimension to capture local relations in the input. In my case, based on the architecture proposed by Tang and Lin [3], I designed residual blocks to replace these initial convolutions. These blocks start with a convolutional layer to resize the input. Then, there is a sequence of layers composed of batch normalization, ReLU activation, and dilated convolutions. By incorporating dilation, as described by Yu and Koltun [7], I aim to achieve a larger receptive field without increasing the parameter count. For each block, the dilation was defined as $d = 2^i$ at layer i. Finally, at the end of each block, there is a skip connection [18] intended to allow deeper layers in the network by facilitating the flow of information and potentially avoiding vanishing gradients.

After the convolutions, as in the AttRNN, the resulting sequences are passed to two bidirectional recurrent layers to extract the two-way long-term dependencies in the signal. The only modification I made for the ResAttRNN in this regard is that I used GRU cells instead of LSTM units to reduce the number of parameters. Then, one of the output vectors (the last one in my case) from the second recurrent layer is projected with a dense layer and used as the query vector of a dot-product attention mechanism to identify the most relevant parts of the sequence. The adaptation I made in this case was to include what I refer to as a *bounding function* before the softmax of the attention mechanism as an attempt to achieve a similar smoothing effect as the one described in Sec. II. This approach was tested with both the logistic sigmoid σ and the $\tanh$ function.

From that point onwards, the ResAttRNN is identical to the AttRNN: there is a sequence of fully connected layers followed by the final classification layer. Regularization was addressed by adding dropout layers [19] throughout the last part of the network.

To test the effectiveness of my proposed architectural designs, I experimented with five versions of the model that vary in the number of residual blocks, the type of filters (1D or 2D), the bounding function, and the overall amount of parameters. The different versions are detailed as follows:

- **ResAttRNN1:** This model closely mimics the AttRNN but incorporates the new architecture, with one residual block replacing each convolutional layer and no attention smoother. The convolutions are performed along the time dimension only. It is a lighter model compared to the AttRNN.
- **ResAttRNN2:** Identical to the ResAttRNN1, but incorporating the logistic sigmoid σ as a bounding function.
- **ResAttRNN3:** Identical to the ResAttRNN1, but incorporating the $\tanh$ as a bounding function.
- **ResAttRNN4:** An alternative that adheres more closely to the residual architecture by going deeper with more residual blocks. While not identical, these blocks more closely resemble those proposed by Tang and Lin [3]. The convolutions are still computed along the time dimension only. It uses the same number of residual units as all the previously described models. Additionally, as in some validation runs the $\tanh$ function performed well as an attention smoother, it was also added to this model. This is the only ResAttRNN variation that has more parameters than the AttRNN.
- **ResAttRNN5:** Another version that follows the residual architecture closely, but also incorporates efforts to reduce the number of recurrent units by improving the feature representation fed into the recurrent layers. This is the only case where the convolutions are two-dimensional (also along the MFCCs dimension). It includes the $\tanh$ function as a bounding function as well.

The key architectural features and the total amount of trainable parameters are summarized in Tab. 1. Notation-wise, there are b_j residual blocks of the j-th type, each consisting of l_j dilated convolutional layers with weights $\mathbf{W} \in \mathbb{R}^{(m_j \times r_j) \times n_j}$, where m_j represents the width, r_j represents the height, and n_j represents the number of feature maps. Additionally, u refers to the number of GRU cells in each bidirectional recurrent layer, while f denotes the bounding function. An example of an architecture diagram is shown in Appendix A, but for complete details about the implementation, please refer to the code.

All ResAttRNN models were trained with the same optimizer that was defined for the AttRNN in the original paper [2].

TABLE 1: Architectural summary for the ResAttRNN instances.

ResAttRNN #	**1**	**2**	**3**	**4**	**5**
b_1	1	1	1	6	5
l_1	5	5	5	3	3
m_1	3	3	3	3	3
r_1	1	1	1	1	3
n_1	5	5	5	45	25
b_2	1	1	1	1	1
l_2	5	5	5	3	3
m_2	3	3	3	3	3
r_2	1	1	1	1	3
n_2	1	1	1	1	1
u	64	64	64	64	48
f	-	σ	$\tanh$	$\tanh$	$\tanh$
Total par.	132K	132K	132K	237K	155K

B. *The TransfClassifier architecture*

This design mirrors the encoder part of the transformer's architecture with minor adjustments to make it suitable for classification in this context.

In this case, to preserve the temporal dependencies in the data, the positional encoding was added to the networks' input (which is detailed in Sec. V) in a similar fashion as the one described by Vaswani et al. [4]. Then, a set of encoder layers were implemented with the multi-head self-attention, the Feed-Forward Network, and the layer normalization components without modifications from the original design.

Next, based on the work by Song et al. [5], I added a dense interpolation layer as described by these authors. Here, the idea is to obtain a compact version of the contextualized representation of the input before passing it to the final classification step. This process involves the *dense interpolation factor* M, which controls the size of the input before the last classification layer. For this work, M was fixed at 60 and I experimented with different configurations of the encoder's shape:

- **TransfClassifier1:** A model with fewer parameters than AttRNN and a similar count with respect to ResAttRNN1, ResAttRNN2, and ResAttRNN3.
- **TransfClassifier2:** The only version with more parameters than AttRNN.
- **TransfClassifier3:** The lightest variation.

For a general architectural layout, see Appendix B. For full implementation details, please consult the code. The main characteristics, including the total number of trainable parameters, are detailed in Tab. 2, where h indicates the number of heads in the attention mechanism, FFU denotes the inner-layer dimensionality of the Feed-Forward Networks, and E refers to the number of encoder layers.

The optimizer for these networks was implemented in the same way as in the original transformer paper [4].

TABLE 2: Architectural summary for the `TransfClassifier` instances.

TransfClassifier #	**1**	**2**	**3**
h	8	10	8
FFU	896	1024	768
E	4	6	4
Total par.	129K	219K	116K

VI. Results

All the models were trained for a maximum of 40 epochs, but training was halted if there was no improvement in validation loss for 6 consecutive epochs. For each architecture, the model with the best validation loss was saved and then tested on the test set. The results are presented in Tab. 3.

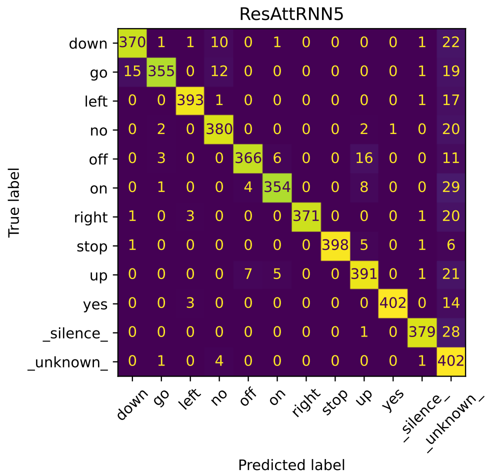
TABLE 3: Performance comparison of different models

Model	Accuracy	F-score	Train time (s)	Pred. time (s)
AttRNN	0.926	0.929	197.8	5.2
ResAttRNN1	0.914	0.919	347.5	3.8
ResAttRNN2	0.920	0.925	412.2	5.3
ResAttRNN3	0.916	0.921	337.1	3.4
ResAttRNN4	0.939	0.942	1812.6	4.7
ResAttRNN5	0.933	0.936	1134.0	5.3
TransfClassifier1	0.847	0.861	1886.2	4.2
TransfClassifier2	0.874	0.883	3279.9	3.0
TransfClassifier3	0.859	0.868	1424.9	2.5

These results show that the `ResAttRNN4` was the best performing model in terms of accuracy and F-score. However, this is also the model with the highest number of parameters, which resulted in the longest training time out of all the `ResAttRNN` networks. Yet, the prediction time was comparable to that of the other networks.

Also, it is noteworthy that the `ResAttRNN5` achieved better results than the original `AttRNN` with respect to accuracy and F-score, even if it uses less parameters and has a comparable prediction time. For these reasons, this network is the one that showcased the best trade-off between performance and complexity out of all my proposed models, making it a competitive potential candidate for real-world applications. Nonetheless, this approach showed a significantly worse training time in comparison to the `AttRNN`, which still outperforms every other model in this regard.

In line with this reasoning, in terms of computational efficiency, it is worth noting that `ResAttRNN1`, `ResAttRNN2`, and `ResAttRNN3`, the "shallower" versions of the `ResAttRNN`, achieved performances comparable to the best models and required significantly less time to train compared to their deeper variants (`ResAttRNN4` and `ResAttRNN5`). This fact makes them potentially suitable for practical implementations where lighter models are key, as they have approximately 20% fewer parameters than the `AttRNN` and only around 1% (or less) difference in accuracy and F-score.



	down	go	left	no	off	on	right	stop	up	yes	_silence_	_unknown_
down	370	1	1	10	0	1	0	0	0	0	1	22
go	15	355	0	12	0	0	0	0	0	0	1	19
left	0	0	393	1	0	0	0	0	0	0	1	17
no	0	2	0	380	0	0	0	0	2	1	0	20
off	0	3	0	0	366	6	0	0	16	0	0	11
on	0	1	0	0	4	354	0	0	8	0	0	29
right	1	0	3	0	0	0	371	0	0	0	1	20
stop	1	0	0	0	0	0	0	398	5	0	1	6
up	0	0	0	0	7	5	0	0	391	0	1	21
yes	0	0	3	0	0	0	0	0	0	402	0	14
silence	0	0	0	0	0	0	0	0	1	0	379	28
unknown	0	1	0	4	0	0	0	0	0	0	1	402

Fig. 3: Confusion matrix for the `ResAttRNN5`

Another remark regarding these models is that, although the `tanh` function performed better during some validation runs, the final model with the logistic sigmoid σ as the bounding function achieved slightly better accuracy and F-score during the test run, despite being slower to train and predict. Still, both bounding functions did seem to improve the results with respect to the version with no attention smoother.

On the other hand, the `TransfClassifier` networks performed consistently worse than the other architectures with respect to accuracy, F-score, and train time. Only the prediction time seems to beat the other models, which is a good feature for the keyword spotting, also recalling that the `TransfClassifier3`, for example, is the lightest network in terms of number of parameters and still achieves an accuracy and F-score over 85%. However, the negative difference in performance in comparison the other models makes this architecture less attractive for real-use cases.

Finally, to assess the type of errors made during this experiment, Fig. 3 shows the confusion matrix for the `ResAttRNN5`. The matrix for this model is shown because it demonstrated higher accuracy than the original `AttRNN` while utilizing fewer parameters. For the other confusion matrices, please refer to the Jupyter Notebook.

From the confusion matrix, it can be seen that words with similar vowel sounds such as "no", "go" and "down", or "up" and "off" are the most frequently misclassified, which makes sense even from a human perspective. Additionally, a common type of error was the misclassification of utterances as unknown words. This could potentially be improved by further enhancing the models' generalizing capabilities in future studies, for example, by using data augmentation techniques.

VII. Concluding Remarks

In the context of a speech command recognition task, I devised and tested different configurations of two types of network architectures: the `ResAttRNN` and the

`TransfClassifier`. The `ResAttRNN` is a mixture between CNNs, RNNs with attention, and deep residual networks, while the `TransfClassifier` is an adaptation of the transfomer's encoder for classification. After performing the MFCCs computation as a shared feature extraction module, these models were paired against an existing approach, the `AttRNN`.

This experiment produced three models -`ResAttRNN5`, `ResAttRNN2`, and `ResAttRNN3`- that rival state-of-the-art architectures. These networks could potentially be implemented in practical contexts where low memory footprint and computational power are essential. Notably, `ResAttRNN5` surpasses `AttRNN` in accuracy despite its lighter weight, while the latter two exhibit comparable performance with a 20% reduction in parameters. Although prediction times are similar across these models, the `AttRNN` maintains superiority in training times.

These results suggest that implementing deep residual learning techniques before a RNN with attention could indeed improve the hidden representation of the data, even when the number of parameters is reduced. However, this approach might introduce an extra cost in terms of training time.

Looking ahead, since the starting point for the networks in this work was always the extraction of MFCCs from short audio clips, a possible direction for future research could involve testing the effectiveness of the proposed models with different types of input. This includes exploring various feature extraction techniques for audio and applying these models to tasks beyond keyword spotting. Additionally, exploring different transformer adaptations that are better tailored for audio and speech recognition could further improve the results I obtained with this architecture. This could involve incorporating convolutional layers to capture local relationships or starting from the raw audio signal as input.

In addition to the theoretical considerations discussed earlier, this work provided me insights into the practical challenges of implementing an end-to-end machine learning project. Particularly, given the size of the datasets and the complexity of the algorithms, ensuring efficiency in the data processing pipeline -including preprocessing steps- is crucial and demands careful attention and time for adequate implementation. Furthermore, the limited computational resources at hand restricted thorough experimentation with different network configurations. Therefore, exploring new approaches, as mentioned above, would demand significant efforts and computational capabilities, highlighting the importance of devising rigorous and well-thought-out solutions before implementation.

References

[1] T. N. Sainath and C. Parada, "Convolutional Neural Networks for Small-footprint Keyword Spotting," in *Interspeech*, (Dresden, Germany), Sept. 2015.

[2] D. Coimbra de Andrade, S. Leo, M. Loesener Da Silva Viana, and C. Bernkopf, "A neural attention model for speech command recognition," *ArXiv e-prints*, Aug. 2018.

[3] R. Tang and J. Lin, "Deep residual learning for small-footprint keyword spotting," 2018.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[5] H. Song, D. Rajan, J. J. Thiagarajan, and A. Spanias, "Attend and diagnose: Clinical time series analysis using attention models," 2017.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[7] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," 2016.

[8] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2016.

[9] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," 2015.

[10] P. Warden, "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition," *ArXiv e-prints*, Apr. 2018.

[11] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[12] "TensorFlow Datasets, a collection of ready-to-use datasets." https://www.tensorflow.org/datasets.

[13] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sequences," *IEEE Transactions on Acoustic, Speech and Signal Processing*, vol. 28, no. 4, 1980.

[14] J. Lyons, D. Y.-B. Wang, Gianluca, H. Shteingart, E. Mavrinac, Y. Gaurkar, W. Watcharawisetkul, S. Birch, L. Zhihe, J. HÃ¶lzl, J. Lesinskis, H. AlmÃ©r, C. Lord, and A. Stark, "jameslyons/python_speech_features: release v0.6.1," Jan. 2020.

[15] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, *Efficient BackProp*, pp. 9–48. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[16] N. V. Prasad and S. Umesh, "Improved cepstral mean and variance normalization using bayesian framework," in *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, pp. 156–161, 2013.

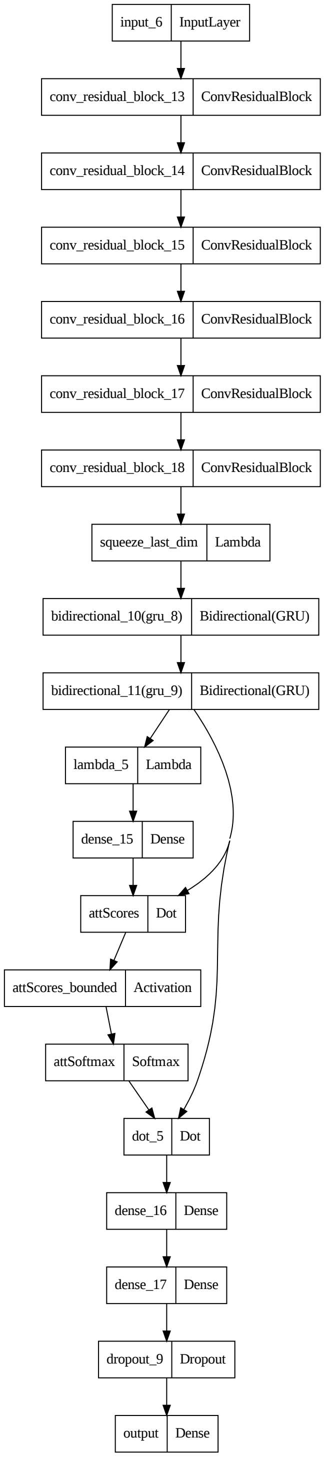
[17] A. Torfi, "Speechpy-a library for speech processing and recognition," *arXiv preprint arXiv:1803.01094*, 2018.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," 2016.

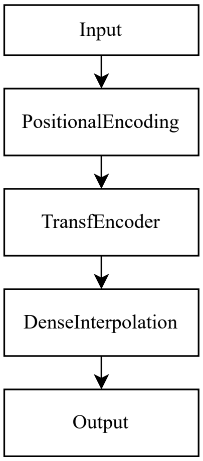
[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Appendix

A. `ResAttRNN5` architecture


input_6 | InputLayer
conv_residual_block_13 | ConvResidualBlock
conv_residual_block_14 | ConvResidualBlock
conv_residual_block_15 | ConvResidualBlock
conv_residual_block_16 | ConvResidualBlock
conv_residual_block_17 | ConvResidualBlock
conv_residual_block_18 | ConvResidualBlock
squeeze_last_dim | Lambda
bidirectional_10(gru_8) | Bidirectional(GRU)
bidirectional_11(gru_9) | Bidirectional(GRU)
lambda_5 | Lambda
dense_15 | Dense
attScores | Dot
attScores_bounded | Activation
attSoftmax | Softmax
dot_5 | Dot
dense_16 | Dense
dense_17 | Dense
dropout_9 | Dropout
output | Dense

B. `TransfClassifier` general architecture


Input
PositionalEncoding
TransfEncoder
DenseInterpolation
Output