End-to-end Keywords Spotting
A minimal deep learning approach

Alfredo Petrella[†]

Abstract—**Abstract here**

Index Terms—**Keyword Spotting, Small Footprint Keyword Spotting, Convolutional Neural Networks, Recurrent Neural Networks.**

I. Introduction

End-to-end small-footprint keyword spotting (KWS) is a crucial task in Artificial Intelligence, being very useful for several applications such as virtual assistance, smart domotic control or wearable intelligent devices. In this project Google Speech Commands Dataset V2 has been used to test the performances of lightweight classifiers, providing a comparison between different preprocessing techniques for recognising predefined speech commands in a stream of user utterances.

In the last years, deep neural networks have shown to provide effective solutions to the small-footprint Keyword Spotting Problem, but the trade-off between accuracy and footprint remains key. Adapting a model on devices with limited-performance from a memory, energetic and computational point of view, in fact, enables us to avoid transferring user data to the cloud, both reducing the latency of the service and preserving their privacy.

In this work, Convolutional Neural Networks (CNNs), previously and successfully applied to the small-footprint Keyword Spotting task, have been tested on different types of input features in order to minimize the number of parameters and the computational complexity of the models. Moreover, a residual-fashion network has been implemented in order to provide a fair benchmark to the simpler models.

In particular, the work consists of:

- a high level description of the whole end-to-end architecture;
- an illustration of the different tested preprocessing pipelines (window size and stride, 24 Mels over a reduced frequency spectrum, 40 Mels, MFCC);
- the training of different neural networks (minimal CNN models and residual-like CNN);
- an analysis of the memory-efficiency trade-off for the studied models.

The report is structured as follows: in Section II the state of the art in the KWS field is presented, in Section III the approach chosen in this work is described and in Section IV the preprocessing pipeline is reported. In Section V, then, the considered architectures are detailed, while Section VI contains the obtained results. Finally, Section VII deals with some extra considerations about future developments and improvements.

Remark I.1.

Paper contribution: First and foremost decide on what precisely is the contribution of your paper over the state of the art. If you think you have several contributions, *focus on the most important one*. It may be that you can add one or two contributions as side topics, but in general you should focus on the most important one so as *to keep your paper focused*. As a side note: If you think you have several contributions for a single paper, you should probably invest in researching state of the art and related work more thoroughly. When you know your contribution, think of a good title.

Remark I.2. Title: Find a short and precise title for you paper exactly matching the content. It's worth investing time into this matter, as *the title will be that part of the paper by which it will be referenced* (in case it gets published).

II. Related Work

Some hints:

- **Approach:** For each you should comment on the paper's contribution, on the good and important findings of such paper and also, **1)** on why these findings are not enough and **2)** how these findings are improved upon/extended by the work that you present here. At the end of the section, you may recap the main paper contributions (maybe one or two, the most important ones) and how these extend/improve upon previous work.
- **References:** please follow this *religiously*. It will help you a lot. Use the Latex `Bibtex` tool to manage the bibliography. A `Bibtex` example file, named `biblio.bib` is provided with this template.

- **Citing conference/workshop papers:** I recommend to always include the following information into the corresponding `bibitem` entry:
  1) author names,
  2) paper title,
  3) conference / workshop name,
  4) conference / workshop address,
  5) month,
  6) year.

  Examples of this are: [1] [2].

[†]Data Science student, Department of Mathematics, University of Padova, badge n. 1206627, email: {alfredo.petrella}@studenti.unipd.it

- **Citing journal papers:** I recommend to always include the following information into the corresponding `bibitem` entry:
  1) author names,
  2) paper title,
  3) full journal name,
  4) volume (if available),
  5) number (if available),
  6) month,
  7) pages (if available),
  8) year.

  Examples of this are: [3] [4] [5].

- **Citing books:** I recommend to always include the following information into the corresponding `bibitem` entry:
  1) author names,
  2) book title,
  3) editor,
  4) edition,
  5) year.

Remark II.1. Note that some of the above fields may not be shown when you compile the Latex file, but this depends on the bibliography settings (dictated by the specific Latex style that you load at the beginning of the document). You may decide to include additional pieces of information in a given bibliographic entry, but please, **be consistent** across all the entries, i.e., use the same fields for the same publication type. Note that some of the fields may not be available (e.g., the paper *volume*, *number* or the *pages*).

III. Processing Pipeline

In order to be of some use, the first requirement that an end-to-end architecture for KWS needs to satisfy is to be enabled dealing with audio signals of variable lengths, and doing it in real time: a solution to this problems could be the one represented in Figure 1. In particular, the convolutional models that will be described in the next sections perform better when trained with a consistent fixed shape for all the input data, and this must be achieved starting from a captured recording of arbitrary size. The audio signal is then processed and segmented into audio chunks and, to avoid slowing down the pipeline processing every single chunk, only the ones containing a sample louder than a given threshold are returned from a Voice Activity Detector. This is a crucial point to speed up the processing and save energy and memory. At this point, from the selected chunks, log-Mels or MFCC are extracted and used as input for the predicting model.

IV. Signals and Features

A. Datasets description

The Google Speech Commands Dataset V2 contains 105,829 samples belonging to 35 classes, plus five longer tracks containing different noise types of variable duration. All the samples are in `.wav` format, with a sampling rate of 16000, while their duration is not standard: besides the five types of noise (about 10 minutes each), most of the samples are one-second long, but some of them are shorter. For this reason, the latter have been padded with silence to meet others duration. An attempt of cropping the longest ones to a smaller size was also performed with worse results, due to the fact that many utterances are not centered in the sample.

The classes that were also present in the first version of the dataset are *go, stop, on, off, up, down, yes, no, left, right*, to which one should add the digits from 0 to 9 (slightly less frequent) and the least frequent *dog, house, bed, cat, bird, visual, backward, sheila, happy, marvin, wow, learn, follow, tree, forward.*

In this work the models have been trained, validated and tested according to the corresponding split suggested in the original dataset paper ???, using an hash function applied to the speaker IDs in order to avoid their utterances to appear both in more than one set, distorting the computed metrics.

Moreover, three different datasets have been considered, based on the frequency of the classes they contain: the original 10-classes dataset, a 20-classes dataset including the second most frequent labels and a 21-classes dataset, different from the previous one for including the *other* class to classify all the input sequences belonging to the 15 remaining classes.

In the three cases, the datasets roughly contain respectively:

- 10-classes: 25k, 8k, 8k samples;
- 20-classes: 46k, 15k, 15k samples;
- 21-classes: 64k, 21k, 21k samples.

In all the cases, an intensive grid search for hyperparameters tuning over the training and the validation set have been executed, and the best resulting models have then been evaluated over the actual test set.

B. Feature extraction

Three different approaches have been used to extract features from the raw input data in order to minimize the number of parameters of the resulting model:

- 24 log-Mels spectrogram ignoring all frequencies above 1760 Hz, as an upper bound to the highest notes an opera soprano is usually required to sing, before windowing (0.05 s length, 0.025 s stride performed better than other previously utilized intervals), applying the FFT, mapping to a Mel scale and filtering with overlapping triangular filters;
- 40 log-Mels spectrogram without masking the highest frequencies, same intervals than above;
- 20 MFCC obtained from the previously computed 40 log-Mels coefficients.

  To speed up the implementation, *Librosa* python library have been used.

V. Learning Framework

A. 2D-CNN

The first architecture is shown in Fig. ???. It is, in its general form, composed of only 4 two dimensional convolutional

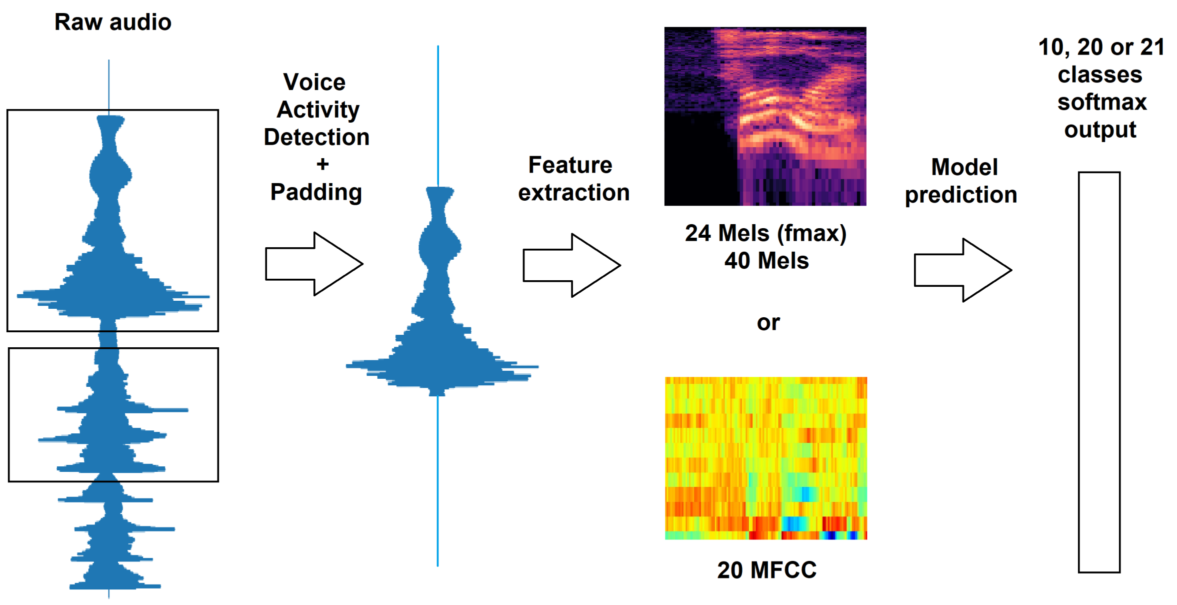

Fig. 1: Proposed end-to-end framework

layers, each followed by a batch normalization layer, to speed up the training and improve generalization, and a ReLU activation function. Moreover, after the first and the last two blocks, an average pooling layer is applied to reduce the shape of the model and, thus, its memory footprint. After the first activation, a small dropout is also applied in some case to regularize the training and generalize against noise.

As for the input shape, it depends on the adopted feature extraction technique. The first is the time dimension, while the second one is related to the frequency, and possible input shapes are `(38, 40)` for the 40 log-Mels case (note that $16000 = 16000 \times (0.025 \times 38 + 0.05)$), `(38, 24)` for the 40 log-Mels case and `(20, 40)` for the MMFC.

After the convolutional blocks, a flattening layer allows the single following dense layer, with `nClasses` neurons followed by a softmax activation, provides the output probability distribution of the input sample belonging to each class.

As a general rule, we can say that the kernel shape is decreasing along the architecture, also due to the subsequent average pooling layers, and also the number of filters of the last convolutional layer shouldn't be too big in order to limit the total number of parameters, strongly influenced by the number of connections in the final fully connected layer. The parameters of the best performing models for each dataset configuration and feature extraction technique are presented in Tab.???

B. 2D-residual-fashion CRNN

The second architecture is shown in Fig. ???. Like the ResNet50 architecture, it is composed by a first block including a two dimensional convolutional layer followed by a batch normalization layer, to speed up the training and improve generalization, and a ReLU activation function. The ResNet50 max pooling layer has been sobstituted with an average pooling layer which has shown better performance in the previous model.

After the first stage, the analogous of a ResNet50 *identity block* is inserted in the network, and it is thus consisting of two convolutional layers, each followed by a batch normalization and a ReLU activation layers, and than one more convolutional-batch normalization layer; before the final activation, a shortcut from the output layer of the first stage is added, with kernel size of 1 and the same number of filters of the last layer to match its dimension, and then the ReLU activation is applyied.

The exact same stage is replicated a second time, and then, after a needed reshape, a GRU layer is inserted to catch the eventually remaining temporal correlation among the features.

This network has only been evaluated with the 40 log-Mels feature extraction, so the input shape is `(38, 40)` as explained before.

At the end, as before, the final dense layer, with `nClasses` neurons followed by a softmax activation, provides the output probability distribution of the input sample belonging to each class.

The parameters of the best performing models are presented in Tab.???

VI. Results

In this section, you should provide the numerical results. You are free to decide the structure of this section. As a general "rule of thumb", use plots to describe your results, showing, e.g., precision, recall and F-measure as a function of the system (learning) parameters. You can also show the precision matrix.

Remark VI.1. Present the material in a progressive and logical manner, starting with simple things and adding details and explaining more complex findings as you go. Also, do not try to explain/show multiple concepts within the same sentence. Try to **address one concept at a time**, explain it properly, and only then move on to the next one.

Remark VI.2. The best results are obtained by generating the graphs using a vector type file, commonly, either `encapsulated postscript (eps)` or `pdf` formats. To plot your figures, use the Latex `\includegraphics` command. Lately, I tend to use pdf more.

Remark VI.3. If your model has hyper-parameters, show selected results for several values of these. Usually, tables are a good approach to concisely visualize the performance as hyper-parameters change. It is also good to show the results for different flavors of the learning architecture, i.e., how architectural choices affect the overall performance. An example is the use of CNN only or CNN+RNN, or using inception for CNNs, dropout for better generalization or attention models. So you may obtain different models that solve the same problem, e.g., CNN, CNN+RNN, CNN+inception, etc.

VII. Concluding Remarks

Conclusion here

In many papers, here you find a summary of what done. It is basically an abstract where instead of using the present tense you use the past participle, as you refer to something that you have already developed in the previous sections. While I did it myself in the past, I now find it rather useless.

References

[1] M. Zargham, A. Ribeiro, A. Ozdaglar, and A. Jadbabaie, "Accelerated dual descent for network optimization," in *American Control Conference*, (San Francisco, CA, US), June 2011.

[2] C. M. Sadler and M. Martonosi, "Data compression algorithms for energy-constrained devices in delay tolerant networks," in *ACM SenSys*, (Boulder, CO, US), Oct. 2006.

[3] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423, July 1948.

[4] S. Boyd, N. Parikh, E. Chu, and B. P. J. Eckstein, "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers," *Foundations and Trends in Machine Learning*, vol. 3, pp. 1–122, Jan. 2011.

[5] D. Zordan, B. Martinez, I. Vilajosana, and M. Rossi, "On the Performance of Lossy Compression Schemes for Energy Constrained Sensor Networking," *ACM Transactions on Sensor Networks*, vol. 11, pp. 15:1–15:34, Aug. 2014.