End-to-end Framework for Keyword Spotting

Stefano Ivancich
Department of
Information Engineering
University of Padova, Italy
Email: stefano.ivancich@studenti.unipd.it

Luca Masiero
Department of
Information Engineering
University of Padova, Italy
Email: luca.masiero.8@studenti.unipd.it

Abstract—**In this paper we present an end-to-end approach for the Keyword Spotting task: it consists of a sliding window technique of one second, a Voice Activity Detection module, a feature extraction module, a neural network and a fusion rule.**

We explore the application of a 1D Convolutional Neural Network (1DCNN) that directly learns a representation from the audio signal, a Depthwise Convolution (DSConv) and an ensemble of the best models.

We achieve a maximum accuracy of 96.8% with the largest model on the Google Speech Commands Dataset V2 (for the 10-commands and 21-commands recognition tasks).

We also compare the performances of our networks on different feature extraction methods such as 80 Mel, 40 MFCCs and 40MFCCs + Δs.

Index Terms—**Human voice, Command Recognition, Deep Learning, Keyword Spotting.**

I. Introduction

The goal of Keyword Spotting is to detect a relatively small set of predefined keywords in a stream of user utterances, usually in the context of an intelligent agent on a mobile phone or a consumer "smart home" device.

Since the cloud-based interpretation of speech input requires transferring audio recordings from the user's device, significant privacy implications should be kept in mind. Therefore, on-device keyword spotting has two main objectives: the first one concerns the recognition of common commands (such as "on" and "off", as well as other frequent words such as "yes" and "no") that can be accomplished directly on the user's device; secondly, keyword spotting can be used to detect *command triggers*, such as "hey Siri", that provide explicit hints for interactions directed at the device itself. It is additionally desirable that such models have a small footprint (measured, for example, by the number of model parameters): in this way they can be deployed on low-power and performance-limited devices.

In the last years, neural networks have shown to provide effective solutions to the small-footprint Keyword Spotting Problem. Research typically focuses on a trade-off between achieving a high detection accuracy with, on the other hand, a small footprint. Compact models are usually variants derived from a full model that sacrifices accuracy for a smaller model footprint.

In this work, we focus on Convolutional Neural Networks (CNNs), a class of models that has been successfully applied to small-footprint keyword spotting; in particular, we explore the use of **Depthwise Separable Convolutions** (**DSConv**) and **1DCNN**.

In this paper we will:

- present an end-to-end architecture for the Keyword Spotting task;
- try different audio features (raw data, 80 Mels, 40 MFCC and 40 MFCC with first and second derivatives, represented by the symbol Δ);
- train different types of neural networks (1DCNN on raw data and DSConv CNN on features);
- build an ensemble of the best models;
- compare prediction speed and number of parameters between the models.

The report is structured as follows. In Section II we present the current state of the art in the Speech Recognition field, in Section III we show our approach in order to tackle the problem and in Section IV we explain the pre-processing techniques we used. In Section V we describe the various architectures we considered, in Section VI we report their results and in Section VII we make some extra considerations on future developments and possible improvements.

II. Related work

The first system similar to a modern ASR was built in the mid-1950s by researchers at the Bell Laboratories and was able to recognize numerical digits from speech using formants[1] of the input audio.

For the next thirty years, researchers developed devices capable of recognizing vowels and consonants using different types of features (like phonemes), until the introduction, in the mid-1980s, of the Hidden Markov Models (HMMs). This approach (described in [1]) represented a significant shift from simple pattern recognition methods (based on templates and a spectral distance measure) to a statistical method for speech processing and was possible due to the incredible advancements in the computer computational power during those years.

In recent times, however, the HMMs were challenged by the introduction of Deep Learning and several architectures that work very well with these type of problems: Convolutional Neural Networks take advantage of weight-sharing

[1]These formants are a concentration of the acoustic energy around a particular frequency in the input file wave.

and the convolution operation (which is shift-invariant in the data representation domain), while Recurrent Neural Networks (RNNs) are able to store a huge quantity of information.

Our two neural networks are inspired by [2] and [3]. In [2] the authors apply 1DCNN on raw signals for environment sound classification: we slightly modified the structure to adapt it to our task. In [3] the authors made some improvements on the MobileNet for Mobile Vision Applications using DSConv.

III. Processing Pipeline

The aim of the end-to-end architecture we propose, represented in Fig. 1, is to handle audio signals of variable lengths and learn from the audio signal received.

A. Variable audio length

One of the main challenges of neural networks in audio processing is that the length of the input sample should be fixed, but the sound captured may have various duration. For this reason, it is necessary to adapt the model making it able to deal with audio signals of different lengths. Moreover, a model should be used for continuous prediction of input audio signals. One way to avoid this constraint, imposed by the input layer of the neural network, is to split the audio signal into different frames of fixed length using a sliding window of 1-second width.

For long contiguous audio recordings, instead of increasing the input dimension of the network (and consequently the number of parameters and its complexity), it is preferable to split the audio waveform into shorter frames: in this way, we keep the network compact so it can process audio waveforms of any length.

However, letting the model process each frame is too expensive (from the computational point of view): for this reason we can use a VAD (Voice Activity Detection) module or a Silence Filter to let the model process just the frames containing voice. This component should be very compact, cheap and fast; it could be part of a software or even a little piece of hardware (like some home device that we use nowadays). Only a batch of subsequent windows will be processed. For each of these windows a Feature Extraction module will extract 80 Mels, 40 MFCC, 40MFCC + Δs or perform no extraction at all: this will be the input to our model.

B. Aggregation of audio frames

During the classification we need to aggregate the predictions to come up to a decision, as illustrated in Fig. 1; for this reason, different fusion rules can be used to reach a final decision, such as the *majority vote* or the *sum rule*.

IV. Signals and Features

As stated before, we used the Google Speech Dataset V2. This dataset contains 105,829 audio files in .wav format (duration = one second) divided in thirty classes, plus an additional class containing five different type of noise of variable duration (about ten seconds for each file).

The core words are ”Yes”, ”No”, ”Up”, ”Down”, ”Left”, ”Right”, ”On”, ”Off”, ”Stop”, ”Go”, ”Zero”, ”One”, ”Two”, ”Three”, ”Four”, ”Five”, ”Six”, ”Seven”, ”Eight”, and ”Nine”. To help distinguish unrecognized words, there are also ten auxiliary words that most speakers only say once: ”Bed”, ”Bird”, ”Cat”, ”Dog”, ”Happy”, ”House”, ”Marvin”, ”Sheila”, ”Tree”, and ”Wow”.

We decided to create two different datasets:

- the first one is made of ten classes (”Yes”, ”No”, ”Up”, ”Down”, ”Left”, ”Right”, ”On”, ”Off”, ”Stop”, ”Go”) and each class contains approximately 3,600 files;
- the second is made of twenty-one classes (”Yes”, ”No”, ”Up”, ”Down”, ”Left”, ”Right”, ”On”, ”Off”, ”Stop”, ”Go”, ”Zero”, ”One”, ”Two”, ”Three”, ”Four”, ”Five”, ”Six”, ”Seven”, ”Eight”, ”Nine” and “Unknown”) and each class contains approximately 3,600 files, except for the “Unknown” class that contains all the remaining auxiliary words.

A. Dataset Partitioning

We divided both datasets into train-validation-test sets as suggested by Google Dataset’s README file and we used a split of 30K-3K-3K for the ten classes dataset and of 84K-9K-11K for the twenty-one classes one.

The training set is used to train the network, while the validation set is used to evaluate the network performance during the training and to allow early-stopping to save the model with the lowest validation error, preventing overfitting. The test set is used to evaluate the network’s final score that reaches the best performance.

B. Features extraction

We decided to study four different types of features in order to compare them and tried to see which was the best able to solve our problem:

- **No feature extraction**: we trained a specific model (1DCNN) directly on the raw signal (a 16,000 elements vector);
- **80 Mels spectrogram**: we took the Fourier Transform of (a windowed excerpt of) the raw signal and mapped the powers of the spectrum obtained into the mel scale (using triangular overlapping windows);
- **40 Mel-frequency cepstral[2] coefficients (MFCC)**: the coefficients are obtained after the computation of the Discrete Cosine Transform (DCT) on the logarithm of the Mel spectrogram;
- **40 MFCC + Δs**: matrix containing the MFCCs previously computed also considering their first and second derivatives.

We used the *Librosa Python Library* to compute these features, in particular we used the following functions:

- `librosa.feature.melspectrogram(...)`;
- `librosa.power_to_db(...)`;
- `librosa.feature.mfcc(...)`;
- `librosa.feature.delta(...)`.

[2]The cepstrum is the result of the Fourier Transform applied to the signal spectrum, in decibel.

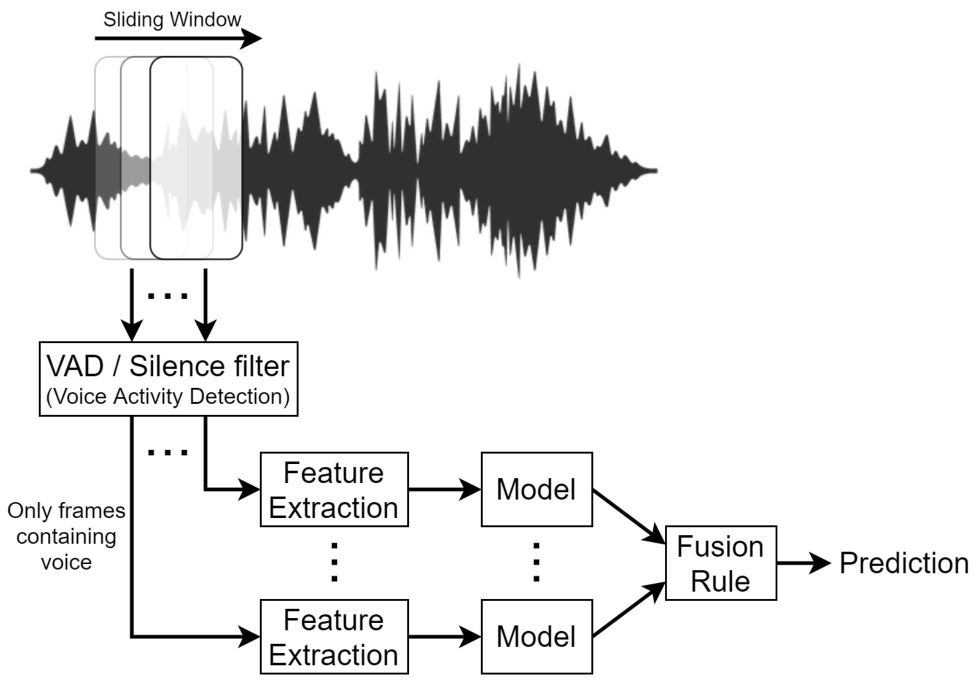

Fig. 1: The end-to-end framework proposed consists of framing the input audio signal into several frames with appropriate overlapping percentage, a Voice Activity Detection module that filters out frames that do not contain voice, a feature extraction module, a neural network and a fusion rule.

V. Learning Framework

A. 1DCNN on raw data

The proposed architecture, shown in Fig. 2, is made of four convolutional layers interlaced with max pooling layers and followed by two fully connected layers and an output layer.

The input consists of an array of 16,000 dimensions, which represents a 1-second audio sample at 16kHz. The proposed 1DCNN has large receptive fields in the first convolutional layers since it is assumed that the first layers should have a more "global view" of the audio signal.

The output of the last pooling layer, for all feature maps, is flattened and represents the input for a fully connected layer. In order to reduce overfitting, batch normalization is applied after the activation function of each convolution layer. After the last pooling layer there are two fully connected layers with 128 and 64 neurons respectively; dropout is applied with a probability of 0.25 for both layers.

Keeping in mind the architecture shown in Fig. 2, it is possible to omit a signal processing module because the network is strong enough to extract relevant low and high-level information from the audio waveform. In the case of multiclass classification, the number of neurons of the output layer is the number of classes. Using the *softmax* as activation function for the output layer, each output neuron indicates the membership degree of the input samples for each class.

B. The DSConv model

The DSConv model applies Depthwise Separable Convolutions. A separable convolution uses less parameters, memory and computations than regular convolutional layers, and in general it even performs better. There are two types of Separable Convolution: Spatial and Depthwise.

Spatial Separable Convolutions divide a kernel into two smaller kernels (e.g. division of a 3×3 kernel into a 3×1 and 1×3 kernel). Instead of considering one convolution with 9 multiplications, we handled two convolutions with 3 multiplications each (6 in total) to achieve the same result. With a lower number of multiplications, the computational complexity significantly decreases, and the network is able to run and train faster.

Depthwise Separable Convolutions use kernels that cannot be split into two smaller kernels that apply two different convolutions: the *depthwise convolution*, which result will be the input for the *pointwise* (1×1) one.

In Fig. 3 we show, on the one hand, the differences between a normal 3D Convolution and a Depthwise one, and on the other the final Depthwise Separable Convolution block we used.

In our work we adopted a Depthwise Separable CNN inspired by [3]. This CNN is made of a 2D-CNN followed by batch normalization and four or five DSConv2D layers. An average pooling, with padding 2×2 and stride 2×2, followed by a fully connected layer, is applied in the end to provide global interaction and reduce the total number of parameters in the final layer. We implemented three different variants: *Large*, *Medium* and *Small*, described in the Tables at pages 5 and 6, where we tried to reduce the number of parameters still maintaining a good accuracy.

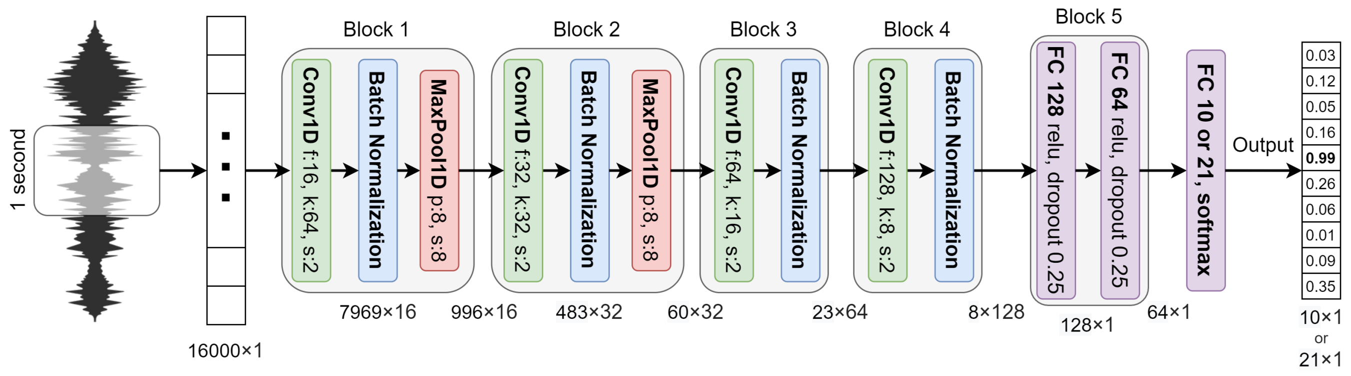

Fig. 2: The architecture of the proposed 1DCNN model.

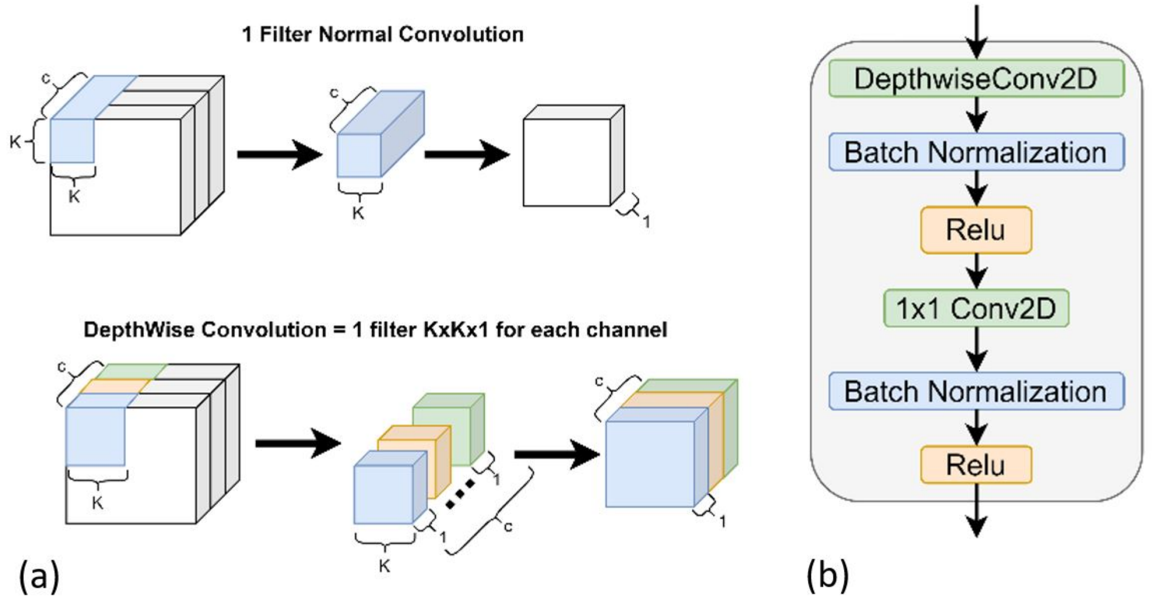

Fig. 3: Differences between normal convolution and depthwise convolution (left). DSConv layer (right).

C. Ensemble

As stated before, we built three kinds of ensembles (*Large*, *Medium* and *Small*) in order to have models that could fit different hardware sizes. The ensembles simply pick the two best models and compute the average of their final prediction, in this way we reached the best accuracy paying a high price: larger networks.

All models were trained with the Google Tensorflow 2.1 Framework with Keras using the standard *Sparse categorical cross entropy* and *Nadam optimizer*. With a batch size of 32, the models were trained for 100 epochs with initial learning rate of 10^{-4}. An *early stopping* procedure with a patience of 10 epochs was applied to avoid overfitting.

VI. Results

We trained each DSConv network for all the features we described in Section IV to see if there was any difference between each set. As stated in Sec. IV, we used two different datasets, one made of ten class and another one of twenty-one.

We found that the number of convolutional layers played a key role in detecting high-level concepts, both for the DSConv model and the 1DCNN one. The number of convolutional layers in our models was determined in an experiment using the audio files of the Google Speech Commands V2 Dataset.

We faced the overfitting problem, recognizing the importance of the *early stopping* procedure during the training, and compared the test scores of the last epochs of the models with the best ones found by the `ModelCheckpoint` function of Keras (using the validation set). It took some time, for us, to choose the best metrics to use to evaluate our models, and in the end we decided *accuracy* was the best choice.

We also measured the prediction speed of each model with and without Feature Extraction.

Furthermore, we noticed that predicting one example or a batch made of ten examples required almost the same amount of time: this happened because TensorFlow makes tensors

Model Size		**Layers**					
		Conv2D	**DSConv2D 1**	**DSConv2D 2**	**DSConv2D 3**	**DSConv2D 4**	**DSConv2D 5**
Small	# Filters	64	64	64	64	64	NA
	Filter size	10×4	3×3	3×3	3×3	3×3	
	Stride	2×2	1×1	1×1	1×1	1×1	
Medium	#Filters	172	172	172	172	172	NA
	Filter size	10×4	3×3	3×3	3×3	3×3	
	Stride	1×2	2×2	1×1	1×1	1×1	
Large	# Filters	276	276	276	276	276	276
	Filter size	10×4	3×3	3×3	3×3	3×3	3×3
	Stride	1×2	2×2	1×1	1×1	1×1	1×1

TABLE I: The configurations of the DSConv model layers considering different model sizes.

	10-commands (30K-3K-3K)	**21-commands (84K-9K-11K)**
Accuracy %	93.0	89.1
# parameters	257,018	257,733
Speed (ms)	28.71	28.25

TABLE II: 1DCNN performances on 10 and 21-commands datasets.

operations; in the final implementation this fact should be taken into account in order to minimize the computational cost.

In Tables II, III, IV, V and VI we summarized the accuracy, the number of parameters and the speed prediction of our models: the 40 MFCCs + Δs one does not perform better, on the contrary this type of model always performed, with a large number of parameters, the worst way.

There is not so much difference between using 80 Mels or 40 MFCCs, calculating MFCCs requires slightly more time since they are calculated after the Mels requiring, however, almost half the parameters.

The fastest network is the 1DCNN on raw data since it has not to calculate any feature extraction, but it is the worst in terms of accuracy (93.0% of accuracy, 257,018 parameters and 28.71 ms of speed prediction for the 10-commands task, while 89.1% of accuracy, 257,733 parameters and 28.25 ms of speed prediction for the 21-commands task).

The smallest model is the DSConvSmall + 40 MFCCs with 127K parameters, 92.9% accuracy and 38.23 ms of speed prediction.

The best accuracy is obtained by the Large Ensemble network (the slower and larger model) that reaches 96.8% of accuracy but with more than 2.5 million parameters.

VII. Concluding Remarks

In this paper, an end-to-end architecture for the Keyword Spotting Task has been proposed. We built two neural networks and an ensemble between the best models.

The 1DCNN consists of five convolutional layers. Instead of using handcrafted static filterbanks (such as the ones used to extract MFCC features), the proposed 1DCNN learns the filters directly from the audio waveform. The other network receives as input features 80Mels, 40MFCCs or 40MFCCs + Δs and consists of four or five DSConv layers. Furthermore, the 1DCNN is faster than the other models, but unfortunately it does not reach the same accuracy.

As we can see from the results, all the features work well but 40MFCCs seems slightly better that the others; the 1DCNN on raw waveform performed the worst way. We then reduced the number of parameters of the DSConv model by lowering the filters number for each convolution. We proposed three version of the DSConv model: *Large*, *Medium* and *Small*.

Our models achieve excellent results but we did not beat the state of the art accuracy of 97.4% reached in [4]. In TABLE VII at page 7 there is a summary of our models compared with the state of the art. We noticed that if we picked more frames per second, as described in the end-to-end architecture, the model would have had many “points of view” of the word, so in the real world it could actually perform better than the test accuracy measured. In various tests we made, the networks (even the 1DCNN, the one with the worst accuracy) captured all the words we were saying thank to the multiple windows – fusion rule technique.

Further improvements could be explored with the SincConv, Attention Mechanism and LSTMs.

The code can be found on the following GitHub page: `https://github.com/ivaste/KeyWordSpotting`.

	10-commands (30K-3K-3K)			**21-commands (84K-9K-11K)**		
	80 Mels	*40 MFCC*	*40 + Δ MFCC*	*80 Mels*	*40 MFCC*	*40 + Δ MFCC*
Accuracy %	**96.0**	95.3	93.9	93.4	**93.7**	92.7
# parameters	874,930	**571,330**	1,178,530	1,375,881	**738,321**	2,013,441
Speed (ms)	**33.39**	30.24	33.42	33.79	**30.87**	33.13
Speed with FE	**41.44**	45.32	44.25	41.62	**39.66**	44.62

TABLE III: DSConv *Large* performances on 10 and 21-commands datasets.

	10-commands (30K-3K-3K)			**21-commands (84K-9K-11K)**		
	80 Mels	***40 MFCC***	*40 + Δ MFCC*	*80 Mels*	*40 MFCC*	*40 + Δ MFCC*
Accuracy %	94.3	**95.0**	94.8	**92.7**	92.2	91.7
# parameters	469,398	**262,998**	675,798	832,673	**399,233**	1,266,113
Speed (ms)	30.75	**29.55**	30.76	**32.25**	30.01	33.03
Speed with FE	38.49	**38.23**	41.82	**39.72**	46.76	47.08

TABLE IV: DSConv *Medium* performances on 10 and 21-commands datasets.

	10-commands (30K-3K-3K)			**21-commands (84K-9K-11K)**		
	80 Mels	***40 MFCC***	*40 + Δ MFCC*	*80 Mels*	*40 MFCC*	*40 + Δ MFCC*
Accuracy %	92.5	**92.9**	92.5	**90.0**	89.2	86.5
# parameters	300,618	**127,818**	473,418	604,757	**241,877**	967,637
Speed (ms)	31.00	**29.27**	30.54	**32.86**	29.48	30.31
Speed with FE	38.31	**38.23**	40.23	**37.97**	48.15	40.99

TABLE V: DSConv *Small* performances on 10 and 21-commands datasets.

	10-commands (30K-3K-3K)			**21-commands (84K-9K-11K)**		
	Large *DSConv L 80 Mels + DSConv L 40 MFCC + DSConv L 40 MFCC Δ*	***Medium*** *DSConv L 40 MFCC + DSConv M 80 Mels + DSConv M 40 MFCC*	***Small*** *DSConv M 40 MFCC + DSConv M 80 Mels + DSConv S 40 MFCC*	***Large*** *DSConv L 80 Mels + DSConv L 40 MFCC + DSConv L 40 MFCC Δ*	***Medium*** *DSConv L 40 MFCC + DSConv M 80 Mels + DSConv L 40 MFCC*	***Small*** *DSConv L 40 MFCC + DSConv M 80 Mels + DSConv L 40 MFCC*
Accuracy %	96.8	96.4	95.6	95.2	95.0	94.2
# parameters	2,624,790	1,303,726	691,463	2,498,019	1,970,227	1,379,431
Speed (ms)	131.01	122.04	106.09	125.9	114.21	109.03

TABLE VI: Ensemble performances on 10 and 21-commands datasets.

	Accuracy %	**# Parameters**	**Speed (ms)**
SincConv [4]	97.4	162,000	40.35
Our Ensemble Large	**96.8**	2,624,790	131.01
Our Ensemble Medium	96.4	1,303,726	122.04
Our DSConvLarge + 80 Mels	96.0	874,930	41.44
Our Ensemble Small	95.6	691,463	106.09
Our DSConvMedium + 40 MFCC	95.0	262,998	38.23
Our 1DCNN on raw data	93.0	257,018	**28.71**
Our DSConvSmall + 40 MFCC	92.9	**127,818**	38.23

TABLE VII: Summary of the 10-commands networks performances.

	Accuracy %	**# Parameters**	**Speed (ms)**
SincConv [4]	97.4	162,000	40.35
Our Ensemble Large	**95.2**	2,498,019	125.9
Our Ensemble Medium	95.0	1,970,227	114.21
Our Ensemble Small	94.2	1,379,431	109.03
Our Ensemble + 40 MFCC	93.7	738,321	39.66
Our DSConvMedium + 80 Mels	92.7	832,673	39.72
Our DSConvSmall + 80 Mels	90.0	604,757	37.97
Our 1DCNN on raw data	89.1	**257,733**	**28.25**

TABLE VIII: Summary of the 21-commands networks performances.

Work Setup

We decided to organize our work in the following way:

- **Literature research**: Stefano Ivancich and Luca Masiero;
- **Dataset preparation**: Stefano Ivancich;
- **1DCNN**: Luca Masiero;
- **DSConv**: Stefano Ivancich;
- **Ensemble**: Stefano Ivancich;
- **Feature Extraction methods**: Luca Masiero;
- **Full Demo**: Stefano Ivancich;
- **LaTeX document**: Luca Masiero and Stafano Ivancich;

What we have learnt as students

We cannot say we faced an easy problem. We learnt how to research and select papers from scientific literature, setup a machine learning project, design an end-to-end architecture for Speech Recognition, debug and improve a machine learning model and write a scientific paper with LaTeX.

We had several problems using our university cluster: even if it is new and powerful, hundreds of students and researchers enqueue jobs everyday, our laptops turned out to perform in a much faster way (the vast majority of the tests run on an ASUS ROG GL703GM *Scar Edition*, with a Intel Core i7-8750H Hexa-Core processor, Nvidia GeForce GTX 1060 and 16GB RAM memory).

References

[1] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," Proceedings of the IEEE, vol. 77, pp. 257–286, Feb 1989.

[2] Sajjad Abdoli, Patrick Cardinal and Alessandro Lameiras Koerich. *End-to-End Environmental Sound Classification using a 1D Convolutional Neural Network*. arXiv:1904.08990v1 [cs.SD] 18 Apr 2019

[3] Yundong Zhang, Naveen Suda, Liangzhen Lai and Vikas Chandra. *Hello Edge: Keyword Spotting on Microcontrollers*. arXiv:1711.07128v3 [cs.SD] 14 Feb 2018.

[4] Simon Mittermaier, Ludwig Kurzinger, Bernd Waschneck and Gerhard Rigoll. *Small-Footprint Keyword Spotting on Raw Audio Data with Sinc-Convolutions*. arXiv:1911.02086v1 [eess.AS] 5 Nov 2019

[5] G Alon. *Key-word spotting—the base technology for speech analytics*. Natural Speech Communications, 2005 - crmxchange.com