**END-TO-END KEY WORD SPOTTING**

[**Abstract** 2](#_Toc47284920)

[**1.** **Introduction** 3](#_Toc47284921)

[**2.** **Related work** 4](#_Toc47284922)

[**3.** **Processing pipeline** 5](#_Toc47284923)

[**4.** **Signals and features** 6](#_Toc47284924)

[**5.** **Learning framework** 7](#_Toc47284925)

[**6.** **Results** 9](#_Toc47284926)

[**7.** **Concluding remarks** 12](#_Toc47284927)

**Abstract**

We explore the application of ~~deep residual learning~~ and ~~dilated convolutions~~ to the keyword spotting task.

The proposed model establishes a new state-of-the-art accuracy of ~~94.1%~~ on Google Speech Commands dataset V2 (for the 20-commands recognition task), while still keeping a small footprint of only ~~202K~~ trainable parameters.

In this paper, we present an end-to-end approach for the keyword spotting task. Environmental sound classification based on a 1D Convolution Neural Network (CNN) that learns a representation directly from the audio signal (?). Several convolutional layers are used to capture the signal's fine time structure and learn diverse filters that are relevant to the classification task. The proposed approach can deal with audio signals of any length as it splits the signal into overlapped frames using a sliding window. Different architectures considering several input sizes are evaluated, including the initialization of the first convolutional layer with a Gammatone filterbank that models the human auditory filter response in the cochlea. The performance of the proposed end-to-end approach in classifying environmental sounds was assessed on the UrbanSound8k dataset and the experimental results have shown that it achieves 89% of mean accuracy. Therefore, the propose approach outperforms most of the state-of-the-art approaches that use handcrafted features or 2D representations as input. Furthermore, the proposed approach has a small number of parameters compared to other architectures found in the literature, which reduces the amount of data required for training.

**Index Terms**: human voice, command recognition, deep learning, keyword spotting

1. **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. Such a capability complements full automatic speech recognition, which is typically performed in the cloud. Due to the fact that the cloud-based interpretation of speech input requires transferring audio recordings from the user’s device, there are significant privacy implications. Therefore, on-device keyword spotting has two main uses: to begin with, recognition of common commands such as “on” and “off” (as well as other frequent words such as “yes” and “no”) can be accomplished directly on the user’s device, thereby sidestepping any potential privacy concerns; secondly, keyword spotting can be used to detect “command triggers” such as “hey Siri”, which provides explicit cues for interactions directed at the device. It is additionally desirable that such models have a small footprint (for example, measured in the number of model parameters) so they can be deployed on low power and performance-limited devices.

In recent years, neural networks have been 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 having, 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, often via some form of sparsification (non esiste).

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 recent years. In particular, we explore the use of **depthwise separable convolutions** and 1D CNN.

In this paper we will:

* Try different audio features (raw data, 80 Mels, 40 MFCC and 40 MFCC with deltas and delta-deltas);
* train different type of neural networks (1D CNN on raw data and DSConv CNN on features);
* Compare prediction speed and the number of parameters of 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 used. In Section V we describe the various architectures we used, in Section VI we report their results and in Section VII we make some extra considerations on them and on some possible developments and future improvements.

1. **Related work**

The first system similar to a modern ASR was built in the 1952 by researchers at Bell laboratories and was able to recognize numerical digits from speech using formants (?) of the input audio. These are a concentration of the acoustic energy around a particular frequency in the input file wave. For the next thirty years, various researchers developed devices capable of recognize vowels and consonants using different types of features like phonemes and keep taking incremental steps forward, until the introduction, in the mid 1980s of the Hidden Markov Models (HMMs). This approach represented a significant shift from simple pattern recognition methods, based on templates and a spectral distance measure, to a statistical method for speech processing

*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.*

and was possible due to the incredible advances in the computer computational power during these years.

But in recent times, the HMMs were challenged by the introduction of Deep Learning and several architecture that works well with these type of problems like Convolutional Neural Networks (CNN) due to their use of weight-sharing and the convolution operation, which is shift-invariant in the data representation domain, and Recurrent Neural Networks (RNN) because of their ability to store information.

Our two neural networks are inspired by [7], [8].

<https://www.researchgate.net/publication/332553888_End-to-End_Environmental_Sound_Classification_using_a_1D_Convolutional_Neural_Network>

In [7] the authors apply 1D CNN on raw signal for environment sound classification, we slightly modified the structure to adapt it to our task. In [8] the authors made some improvement on the MobileNet using

As in [8] our first CNN presents a first layer that scans the input allowing the kernel to move along the timedimension (x-axis), while the other dimension of the kernel covers the entire coefficients dimension (y-axis). The second neural network instead has a number of convolutional layers fixed but each of them covers only a smaller portion of the input in both dimensions, such as in [7]. The development of recurrent neural networks and of attentions models increased performance on multiple tasks [9] [10], especially those related to long sequence to sequence models.

These models are extremely powerful ways to understand what parts of the input are being used by the neural network to predict outputs. In the case of acoustic models, Connectionist Temporal Classification loss shows good performance in English and Mandarin speech to text tasks [11]. Results using raw waveform without any Fourier analysis have also been investigated [12].

1. **Processing pipeline**

The aim of the end-to-end architecture we propose, represented in Fig. 1, is to handle audio signals of variable lengths, learning from the received audio signal, a discriminative representation that achieves a good classiﬁcation performance on diﬀerent speech commands.

**Variable Audio Length**

One of the challenges of using neural networks in audio processing is that the length of the input sample must be ﬁxed but the sound captured may have various duration. Therefore, it is necessary to adapt the model to be used with audio signals of diﬀerent lengths. Moreover, a model must 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 several 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 increasing 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.

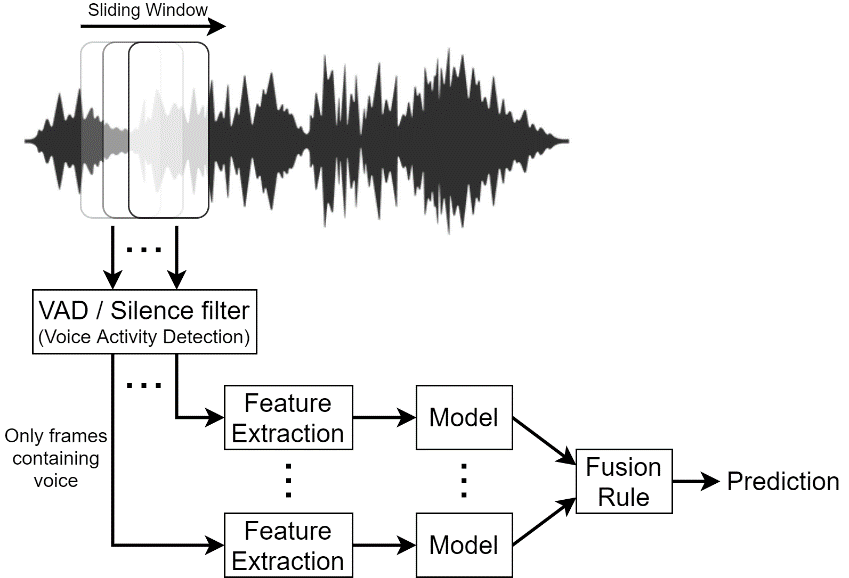
This fact, however, leads to … is too computational expensive to let the model process each frame. So, we can use a VAD (Voice Activity Detection) or Silence filter to let the model process just the frames containing voice. This component must be very compact, low computational expensive and very fast, like just ignoring anything under 40db, it can be a piece of 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+deltas or no extraction at all: this will be the input of our model.

**Aggregation of Audio Frames**

During the classiﬁcation we need to aggregate the predictions to come up to a decision, as illustrated in Figure 1. For such an aim, diﬀerent fusion rules can be used to reach a ﬁnal decision, such as the majority vote or the sum rule.

When there are K classes, we generate K values and them for an audio input, we choose the class with the maximum yi value.



1. **Signals and features**

As stated before in Section I, we used the Google Speech Dataset V2 containing 105,829 audio files in .wav format divided in 30 classes of duration of about 1 seconds, plus an additional class containing 5 different type of noises of variable duration (about 10 seconds 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, which most speakers only said once.

These include "Bed", "Bird", "Cat", "Dog", "Happy", "House", "Marvin", "Sheila", "Tree", and "Wow".

The original audio files were collected in uncontrolled locations by people around the world. Was requested that they do the recording in a closed room for privacy reasons, but didn't stipulate any quality requirements. This was by design, since Google wanted examples of the sort of speech data that they 're likely to encounter in consumer and robotics applications, where we don't have much control over the recording equipment or environment. The data was captured in a variety of formats, for example Ogg Vorbis encoding for the web app, and then converted to a 16-bit little-endian PCM-encoded WAVE file at a 16000 sample rate. The audio was then trimmed to a one second length to align most utterances, using the [extract\_loudest\_section](<https://github.com/petewarden/extract_loudest_section>) tool. The audio files were then screened for silence or incorrect words, and arranged into folders by label.

We decided then to create 2 different datasets:

1) composed by 10 classes = {yes, no, up, down, left, right, on, off, stop, go} where each class contains approximately 3,600 files.

2) composed by 21 classes = {yes, no, up, down, left, right, on, off, stop, go, zero, one, two, three, four, five, six, seven, eight, nine, unknown} where each class contains approximately 3,600 files, except for the *unknown* class that contains all the remaining auxiliary words.

**Dataset Partitioning**

We divided both datasets into train-validation-test sets as suggested by Google into the dataset’s README file, so we used a split of 30k-3k-3k for the 10 classes dataset and 84k-9k-11k for the 21 classes. The training set is used to train the network, while the validation one to compute the network performance during the training and to allow early-stopping to save the model with the lowest validation error, avoiding in this way the overfitting. The test dataset finally, is used to compute the network score with the best performing network.

**Features extraction**

We decided to study 4 different types of features in order to compare them and try to see which is the best performing for resolving our problem:

* **No feature extraction:** we trained a specific model (1D CNN) directly on the raw signal, that is a 16,000 elements vector.
* **80 Mel spectrogram:** Take the Fourier transform of (a windowed excerpt of) the raw signal and map the powers of the spectrum obtained onto the mel scale, using triangular overlapping windows.
* **40 Mel-frequency cepstral coefficients (MFCC):** coefficients that are obtained after the computation of the Discrete Cosine Transform (DCT) on the logarithm of the Mel spectrogram.
* **40 MFCC + Delta + Delta-Delta:** matrix containing the MFCCs previously computed, their first derivative (Delta) and their second derivative (Delta-Delta).

We used the python library to compute those features, in particular the functions: librosa.feature.melspectrogram(…), librosa.power\_to\_db(…) and librosa.feature.mfcc(…), librosa.feature.delta(…).

1. **Learning framework**

**1D CNN on raw data**

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

The input is an array of 16,000 dimensions, which represents 1-second of audio sampled at 16kHz.

The proposed 1D CNN 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.

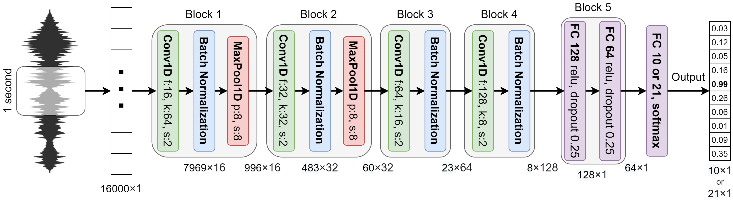
The output of the last pooling layer for all feature maps is flattened and used as input to a fully connected layer.

In order to reduce the over-fitting, 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 on which a drop-out is applied with a probability of 0.25 for both layers.

By the use of the architecture shown in Figure 2, it is possible to omit a signal processing module because the network is powerful enough to extract relevant low-level 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 softmax as the activation function for the output layer, each output neuron indicates the membership degree of the input samples for each class.



**DSConv Model**

This model uses Depthwise Separable Convolutions.

A separable convolution uses less parameters, less memory and less computations than regular convolutional layers, and in general they even perform better.

There are two types of Separable convolution: Spatial and depthwise.

Spatial Separable Convolutions divides a kernel into two, smaller kernels. E.g. divide a 3x3 kernel into a 3x1 and 1x3 kernel. So, instead of doing one convolution with 9 multiplications, we do two convolutions with 3 multiplications each (6 in total) to achieve the same effect. With less multiplications, computational complexity goes down, and the network is able to run and train faster.

The problem with Spatial Separable Convolutions is that not all kernels can be “separated” into two, smaller kernels. So during training, since of all the possible kernels the network could adopt, it can only end up using one of the tiny portion that can be separated into two smaller kernels.

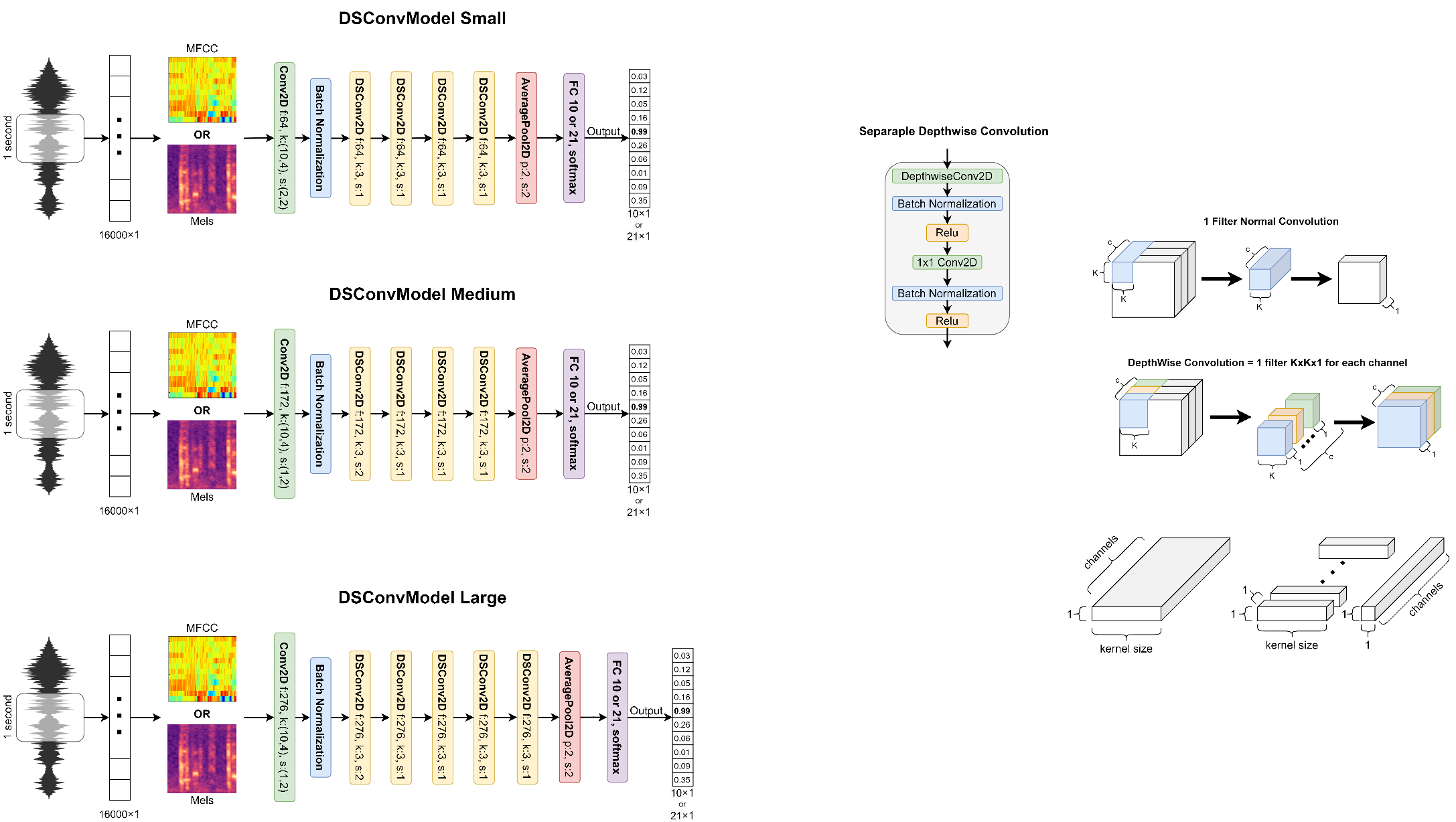
Depthwise Separable Convolutions uses kernels that cannot be “factored” into two smaller kernels.

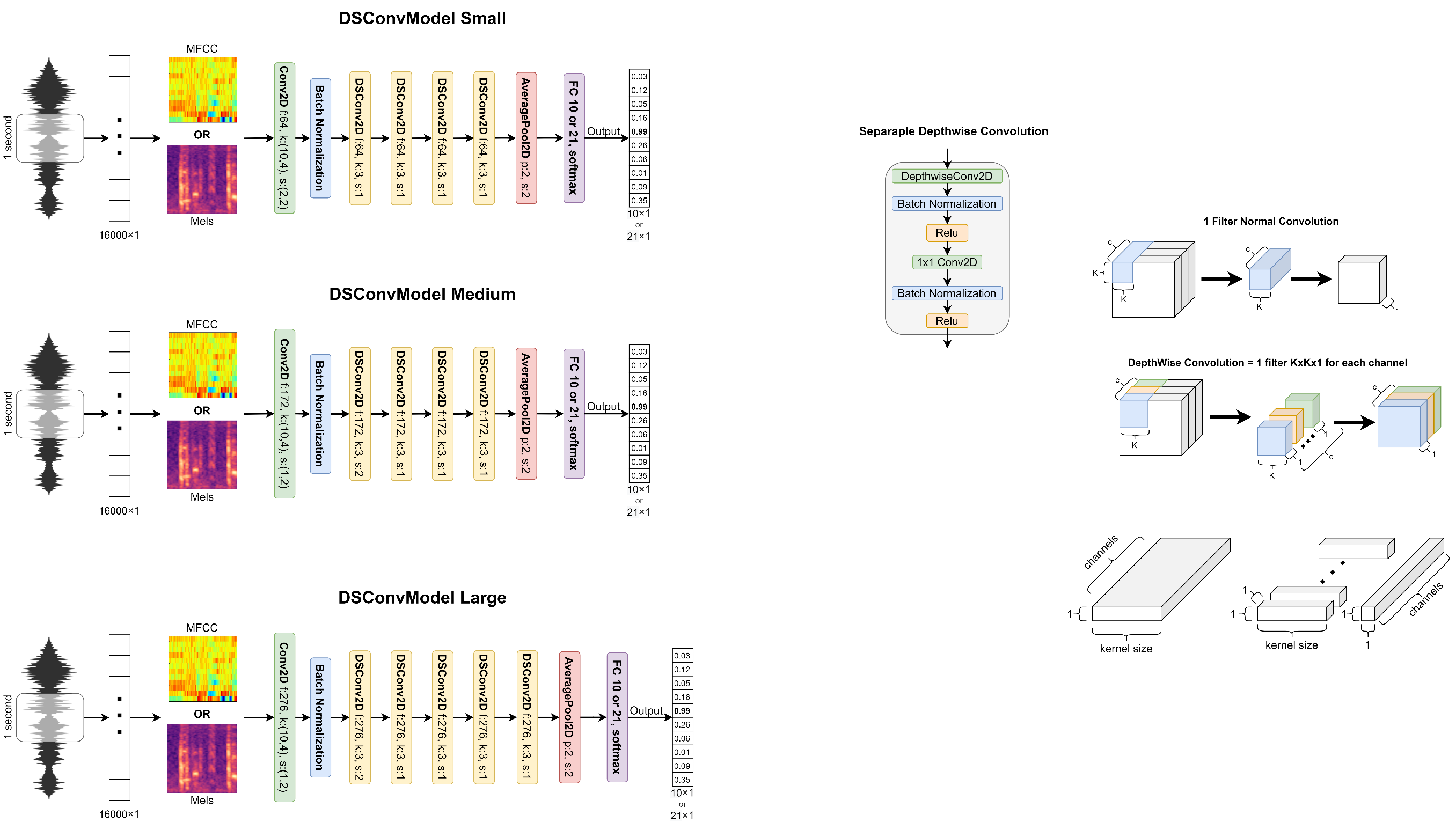
It deals not just with the spatial dimensions, but also with the depth dimension (number of channels).

Splits a kernel into 2 separate kernels that do two convolutions:

* the depthwise convolution, which result will be the input for:
* the pointwise (1x1) convolution.

In figure X you can see the difference between a normal 3D Convolution and a Depthwise, and you can see the final Depthwise Separable Convolutions block that we used.



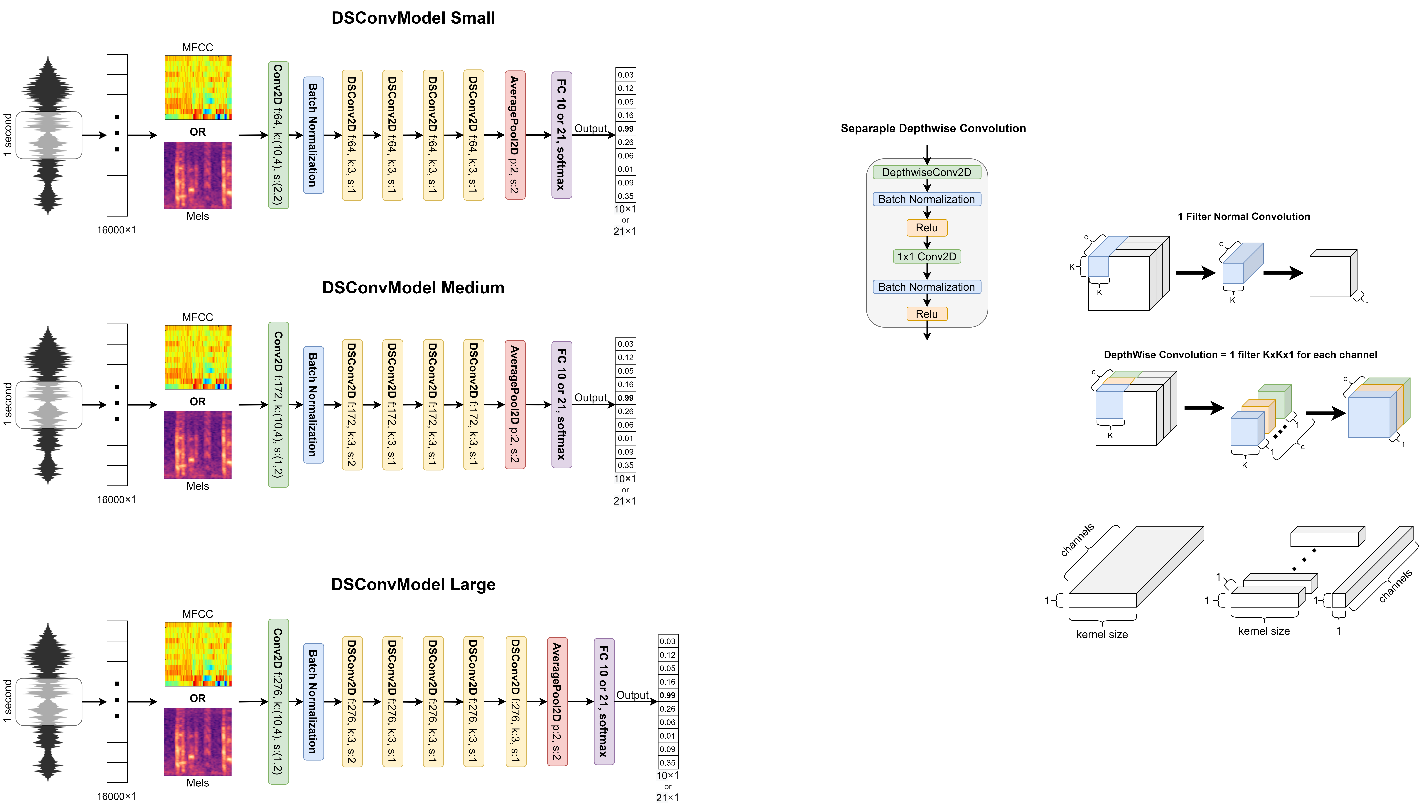


In this work, we adopt a depthwise separable CNN.

Inspired by the implementation of Hello Edge: Keyword Spotting on Microcontrollers.

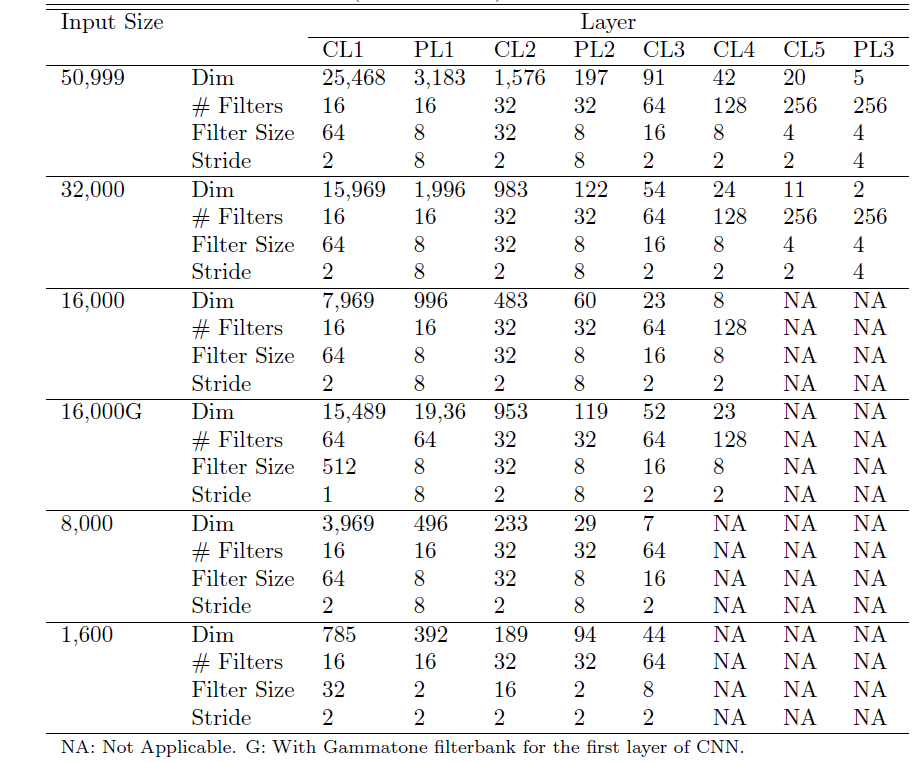
Which is composed of a 2D CNN followed by batch normalization and 4 or 5 DSConv2D layers. An average pooling, with padding 2x2 and stride 2x2, followed by a fully-connected layer is used at the end to provide global interaction and reduce the total number of parameters in the final layer.

We implemented 3 variants: Large, medium and small, described in Table X, in which we tried to reduce the number of parameters while maintaining a decent accuracy.



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Model Size | | Layers | | | | | |
| Conv2D | DSConv2D 1 | DSConv2D 2 | DSConv2D 3 | DSConv2D 4 | DSConv2D 5 |
| Small | #Filters  Filter size  Stride | 64  10x4  2x2 | 64  3x3  1x1 | 64  3x3  1x1 | 64  3x3  1x1 | 64  3x3  1x1 | NA |
| Medium | #Filters  Filter size  Stride | 172  10x4  1x2 | 172  3x3  2x2 | 172  3x3  1x1 | 172  3x3  1x1 | 172  3x3  1x1 | NA |
| Large | #Filters  Filter size  Stride | 276  10x4  1x2 | 276  3x3  2x2 | 276  3x3  1x1 | 276  3x3  1x1 | 276  3x3  1x1 | 276  3x3  1x1 |

TIPO QUESTA



**Ensemble**

The ensemble simply took the 2 best models and mean their final prediction. This allowed us to reach the best accuracy.

All models are trained in Google Tensorflow 2.1 framework with Keras [31] using the standard Sparse categorical crossentropy and Nadam optimizer. With a batch size of 32, the models are trained for 100 epochs with initial learning rate of 10^-4. An early stopping with a patience of 10 epochs is applied to avoid overfitting.

1. **Results**

We trained each DSConv networks for all the features we described in Section IV to see if there are any differences between each set. As stated before, we used two different datasets, one composed by 10 class and another one composed by 21 class.

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

We addressed the overfitting problem, recognizing the importance of the early-stopping procedure during the training comparing the test scores on the last epochs of the models with the best ones found by the ModelChekpoint function of Keras using the validation set.

We though very much which should be the best metrics to evaluate our models, at the end we choose accuracy. Given the fact that in a problem where there is a large class imbalance, a model can predict the value of the majority class for all predictions and achieve a high classification accuracy. So, further performance measures are needed such as F1 score and Brier score, but since in our dataset classes are balanced, we can still use Accuracy.

We also measured the prediction speed of each model with and without Feature Extraction. We should not consider Multiplication as metrics because is an indirect alternative for the direct metric such as latency.

Furthermore, we notice that predicting 1 example or a batch of 10 requires the same amount of time, this because TensorFlow make tensors operations. So in the final implementation this fact should be taken into account in order to minimize the computational cost.

In tables X,Y,Z,W,P you can see the accuracy and the speed prediction of our models.

The most performant one

As you can see using 40MFCCs + deltas don’t perform better, on the contrary the models have worse accuracy, a larger number of parameters and they are slower.

There is no much difference on using 80 Mels or 40 MFCCs, calculating MFCC requires slightly more time since they are calculated after the Mels but MFCC requires almost the half of parameters.

The fastest network is the 1D CNN on RAW data since it has not to calculate any feature extraction, but is the worst in terms of accuracy.

The best accuracy is obtained by the ensemble network, but it is obviously the slower and the bigger model.

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

1DCNN

|  |  |  |
| --- | --- | --- |
|  | **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 |

DSConv Large

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **10 commands**  (30k - 3k - 3k) | | | **21 commands**  (84k - 9k - 11k) | | |
| 80 Mels | 40 MFCC | 40 +  MFCC | 80 Mels | 40 MFCC | 40 +  MFCC |
| **Accuracy %** | **96,0** | 95,3 | 93,9 | 93,4 |  |  |
| **# parameters** | 874,930 | 571,330 | 1,178,530 | 1,375,881 |  |  |
| **Speed (ms)**  **Speed with FE** | 33,39  41,44 | 30,24  45,32 |  | 33,79  41,62 |  |  |

DSConv Medium

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **10 commands**  (30k - 3k - 3k) | | | **21 commands**  (84k - 9k - 11k) | | |
| 80 Mels | **40 MFCC** | 40 +  MFCC | 80 Mels | 40 MFCC | 40 +  MFCC |
| **Accuracy %** | 94,3 | **95,0** | 94,8 | 92,7 | 92,2 |  |
| **# parameters** | 469,398 | **262,998** | 675,798 | 832,673 | 399,233 |  |
| **Speed (ms)**  **Speed with FE** | 30,75  38,49 | **29,55**  **38,23** |  | 32,25  39,72 | 30,01  46,76 |  |

DSConv Small

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **10 commands**  (30k - 3k - 3k) | | | **21 commands**  (84k - 9k - 11k) | | |
| 80 Mels | **40 MFCC** | 40 +  MFCC | 80 Mels | 40 MFCC | 40 +  MFCC |
| **Accuracy** | 92,5 | **92,9** |  | 90,0 | 89,2 |  |
| **# parameters** | 300,618 | **127,818** |  | 604,757 | 241,877 |  |
| **Speed (ms)**  **Speed with FE** | 31,00  38,31 | **29,27**  **38,23** |  | 32,86  37,97 | 29,48  48,15 |  |

Ensemble:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **10 commands**  (30k - 3k - 3k) | | | **21 commands**  (84k - 9k - 11k) | | |
| DSConv Large  80 Mels  +  DSConv Large  40 MFCC |  |  | DSConv Large  80 Mels  +  DSConv Large  40 MFCC |  |  |
| **Accuracy** | 96.6 |  |  |  |  |  |
| **# parameters** | 1,446,260 |  |  |  |  |  |
| **Speed (ms)** | 89,28 |  |  |  |  |  |

BEST CONFUSION MATRIX…..

1. **Concluding remarks**

In this paper, an end-to-end architecture for the key word spotting task has been proposed. We built 2 neural network and an ensemble between the best models.

The 1D CNN network consists of five convolutional Layers. Instead of using handcrafted static filterbanks such as those used to extract MFCC features, the proposed 1D CNN learns the filters directly from the audio waveform.

The other network receives as input features 80Mels, 40MFCCs or 40MFCCs+deltas and consists of 4 or 5 DSConv layers that…

Furthermore, the proposed end-to-end 1D architecture has fewer parameters than all the other CNN, unfortunately doesn’t reach the same performances.

As we can see from the results, all the features work well but 40MFCCs seems slightly better that the other and the 1DCNN on RAW waveform was the worst.

We further reduce the number of parameters of the DSConv model by reducing the number of the filters for each convolution. Proposing 3 version of the DSConv model: Large, medium and small.

Our models achieve excellent results. Unfortunately we didn’t beat the state of the art accuracy of 97.4% reached by SMALL-FOOTPRINT KEYWORD SPOTTING ON RAW AUDIO DATA WITH SINC-CONVOLUTIONS. You can find a summary of our model compared with the state of the art in TABLE X.

We noticed that if we pick more frames per second as described in the end-to-end architecture the model has many “point of view” of the word, so in real world it actually performs better than the test accuracy measured. In various tests we have made, also the 1D CNN, that is the one with the worse accuracy, captured all the words we were saying thanks to the multiple windows – fusion rule technique.

Further improvement could be exploring the SincConv,….

You can find the code on github [X]

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Accuracy %** | **# Parameters** | **Speed (ms)** |
| SincConv | 97.4 | 162K |  |
| Our Ensemble | **96.6** | 1,446,260 | 89,28 |
| Our DSConvLarge + 80Mels | 96.0 | 874,930 | 41,44 |
| Our DSConvMedium + 40MFCC | 95.0 | 262,998 | 38,23 |
| 1D CNN on Raw data | 93,0 | 257,018 | **28,71** |
| Our DSConvSmall + 40MFCC | 92,9 | **127,818** | 38,23 |
|  |  |  |  |