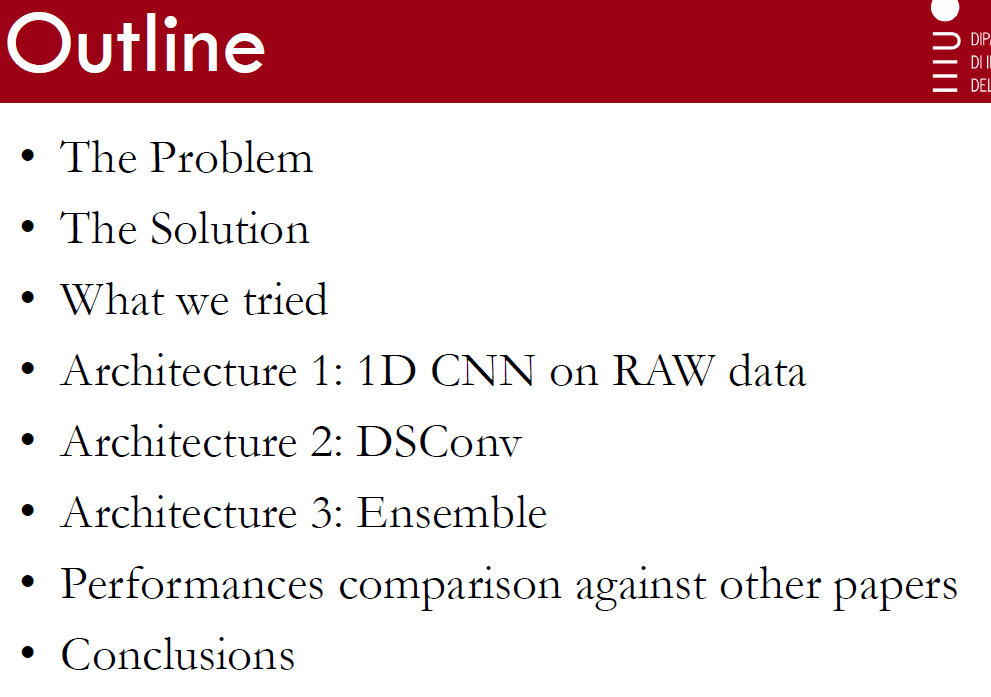
**PRESENTATION SCRIPT**

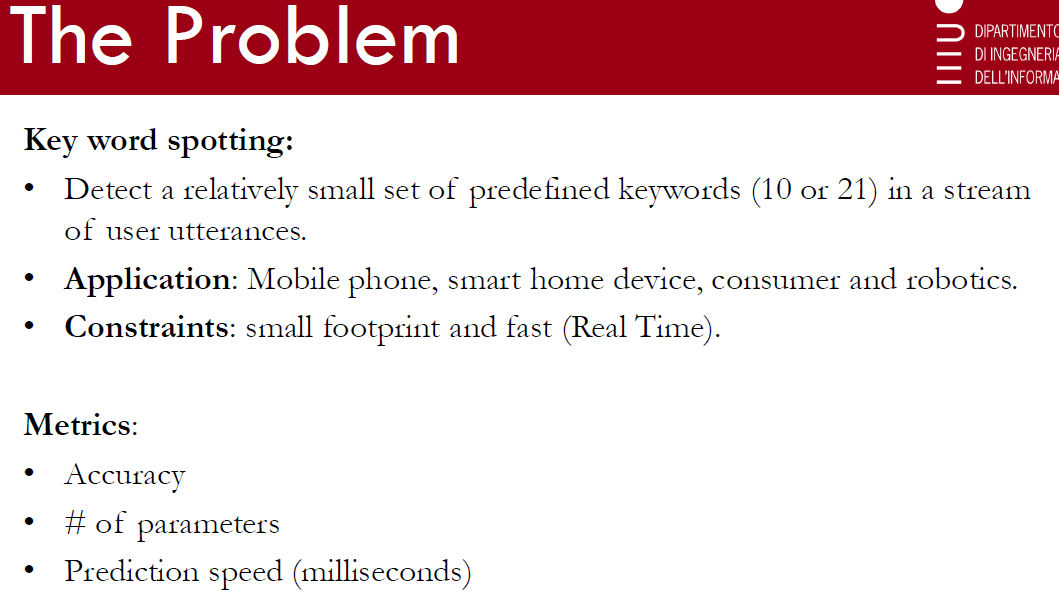
Good morning, we are Stefano Ivancich and Luca Masiero.

**Outline**



Today, we are going to talk about the Keyword Spotting Problem, we will present a possible solution for it, discuss the three architectures we made and we will compare them with other papers.

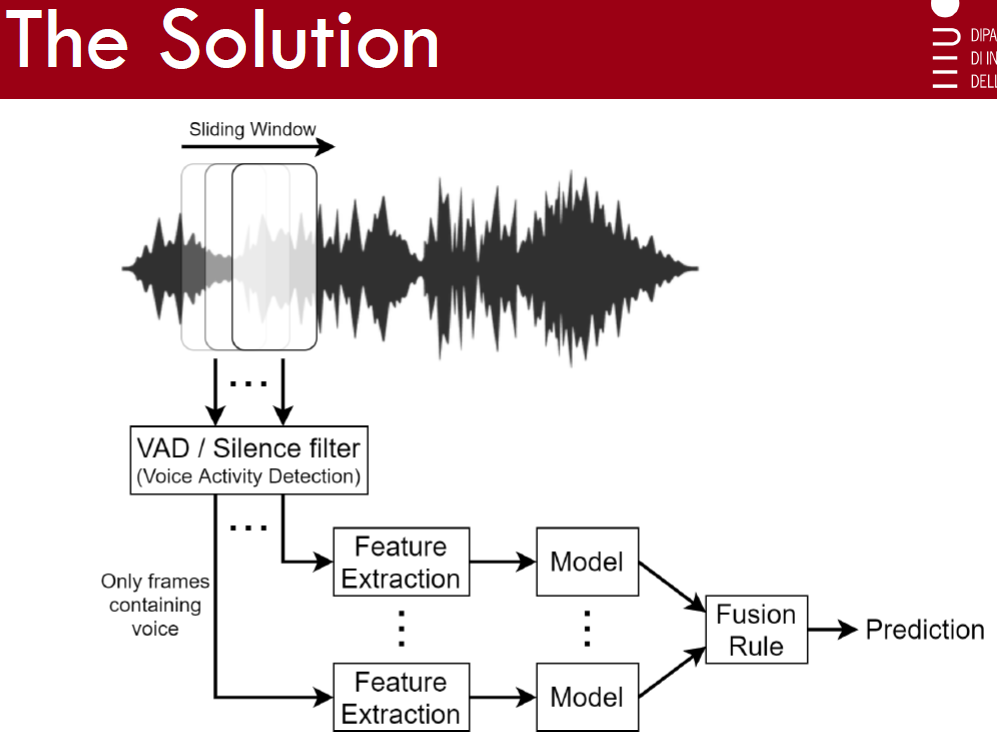
**The problem**



The goal of Keyword Spotting is to detect a relatively small set of predefined keywords in a stream of user utterances, usually on a mobile phone, a consumer “smart home” device or a robotics application.

The constrains are that such models should have a small footprint and a fast prediction speed so they can be deployed on low-power and low-performance-limited devices.

**The Solution**



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.

The aim of the end-to-end architecture we propose, represented in this figure, is to handle audio signals of variable lengths.

One way to avoid the 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.

However, letting the model continuously process each frame is too computationally expensive. For this reason, we can use a VAD (a Voice Activity Detection) module or a Silence Filter to let the model process only the frames that contain voice.

This component must be very compact, cheap and very fast, it could be piece of software or even a little piece of hardware.

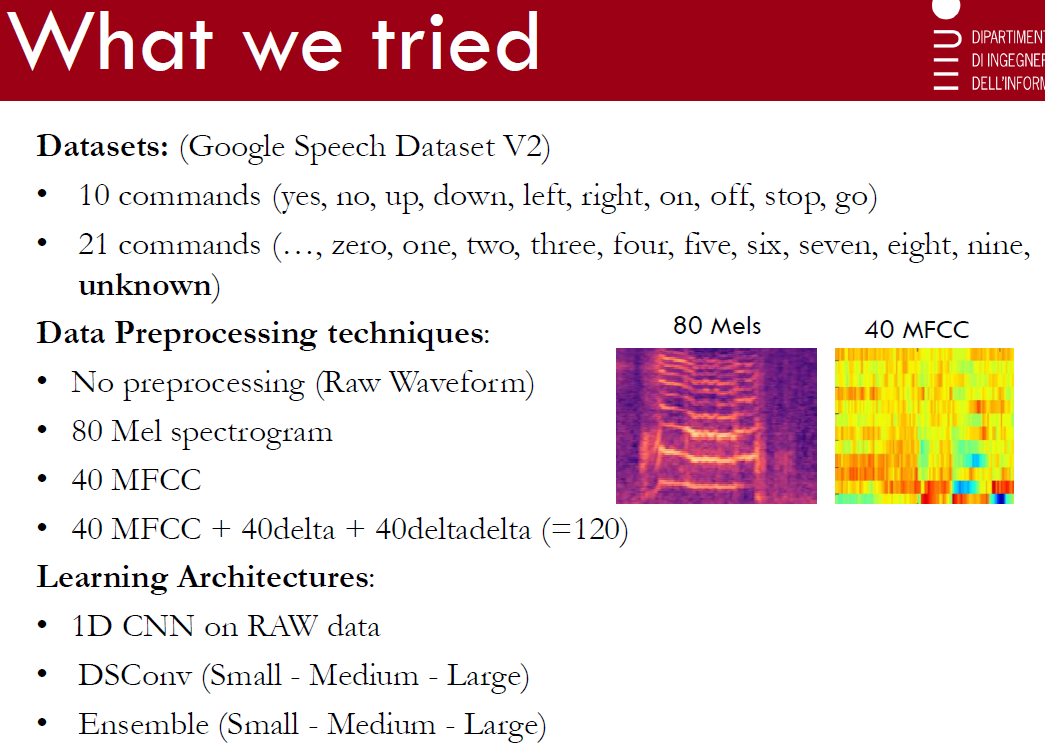
Only a batch of subsequent windows will be processed. For each of these windows a Feature Extraction module will extract 80 Mels, 40MFCC, and this will be the input of the model.

During the classification we need to aggregate the various predictions to come up to a single decision. For this reason, different fusion rules can be used to reach a final decision, such as the majority vote or the sum rule.

**Live DEMO**

…………….

**What we tried**



We used the **Google Speech Dataset V2**. This dataset contains 100K audio files of 1 second divided in thirty classes.

We decided to create two different datasets: the first one is made of 10 classes, the second of 21 classes.

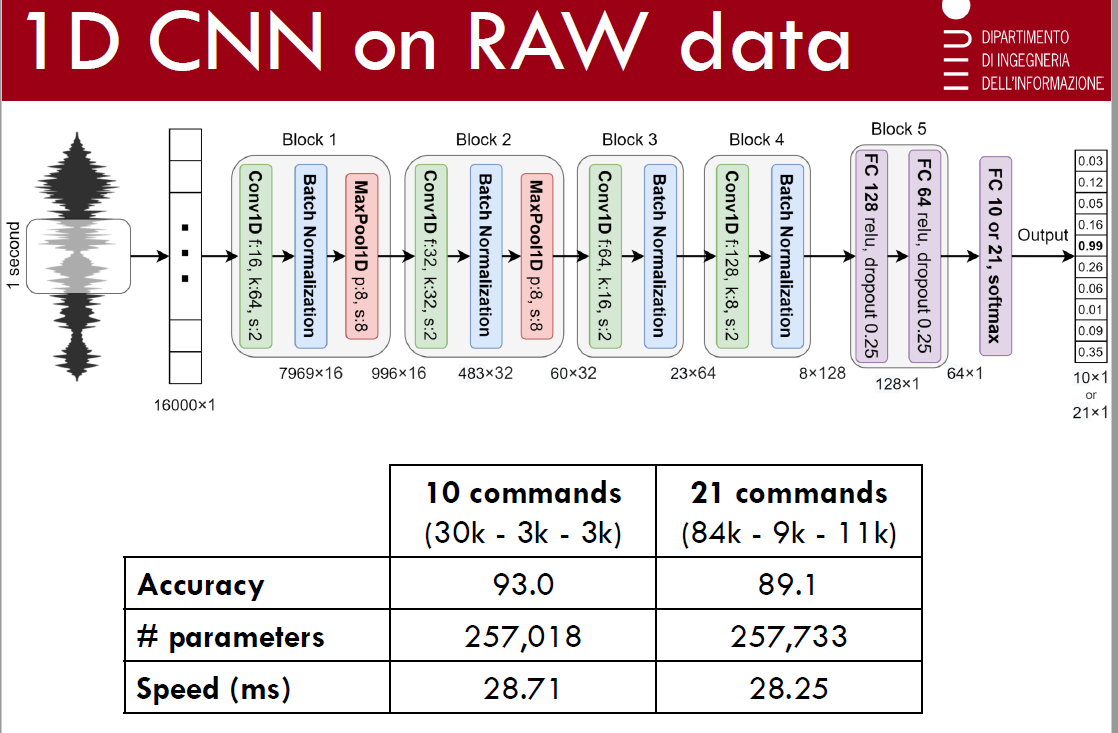
We decided to study four different types of **features** in order to compare them and try to see which was the best one able to solve our problem:

* No feature extraction: we trained a specific model (1DCNN) directly on the raw signal;
* 80 Mels spectrogram;
* 40 Mel-frequency cepstral coefficients (MFCC);
* 40 MFCC with their first and second derivatives.

We tried different types of neural networks: 1DCNN on raw data and DSConv CNN on features and we built an ensemble of the best models.

Furthermore we implemented different model sizes in order to fit different devices.

**1D CNN on RAW data**



The 1DCNN architecture 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 window sampled at 16kHz.

In order to reduce overfitting, batch normalization is applied after the activation function of each convolution layer and dropout is applied with a probability of 0.25 on the fully connected layers.

The performances of this model are the worst of all models: for the 10-commands task it reaches a 93% of accuracy and 89.1 for the 21-command one, with 250K parameters, and a prediction speed of 28 ms, it is the fastest compared to all the other models.

**LUCA**

**Separable CONV**

**Separable convolution** uses less parameters, less memory and less computations than regular convolutional layers, and it even performs better.

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

**Spatial separable convolution** divides a kernel into two, smaller kernels. The **problem** is that not all kernels can be mathematically “separated”.

**Depthwise Separable convolution** uses kernels that cannot be “factored” into two smaller kernels. It deals not just with the spatial dimensions, but also with the depth dimension (the number of channels).

This type of convolution splits a kernel into two separate kernels that apply two more convolutions:

the depthwise and the pointwise one.

**Figure - Separable CONV**

In the first picture we have a normal convolution with 1 filter.

The depthwise convolution corresponds to the application of 1 filter for each channel.

Then, here we have the Depth Separable convolution layer that we implemented. It is made of DSConv2D and a pointwise 2Dconvolution, with batch normalization and a Relu activation fuction.

**DSConv Model - Large**

The Depth separable model takes as input the Mel spectrum or MFCC or MFCC+Δs.

It is made of a normal 2D CNN followed by batch normalization and 5 DSConv2D layers. An average pooling layer followed by a fully-connected layer, is used 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*, where we tried to reduce the number of parameters still maintaining a good accuracy.

For the 10cmd task:

* the best accuracy of 96% is reached using 80mel;
* The smallest model is the 40MFCC with 570K parameters;
* The fastest 80mel with 41ms;

For the 21cmd task: the best accuracy 93.7%, the smallest and fastest model using 40MFCC

**DSConv Model - Medium**

The network is the same as before but with a DSConv layer less and, for each convolution we use less filters. For example in the large one we used 276 filters while in the medium 172.

For the 10cmd task: the best accuracy of 95%, the smallest and fastest model using 40MFCC

For the 21cmd task:

* the best accuracy of 92.7% is reached using 80mel.
* The smallest model is the 40mfcc with 400K parameters
* The fastest 80mel with 39ms

**DSConv Model - Small**

As before we reduced the number of filters of each convolution to 64.

For the 10cmd task: the best accuracy of 92.9%, the smallest and fastest model using 40MFCC

For the 21cmd task:

* the best accuracy of 92.7% is reached using 80mel.
* The smallest model is the 40mfcc with 241K parameters
* The fastest 80mel with 37ms

**Ensemble 10-cmd e 21-cmd**

As stated before, we built three kinds of ensembles: *Large*, *Medium* and *Small*, in order to have models that can fit different hardware sizes. The ensembles simply pick the 3 best models and compute the average of their final prediction, in this way we reached the best accuracy paying a high price: larger networks.

RESULTS DISCUSSIONNNNN………………………….

**Performances 10-cmd**

The state of the art of 97.4% of accuracy with 127K parameters is reached by the paper SincConv.

For the 10cmd task:

* the best accuracy of 96.8% is reached by the Ensemble large model but with 2.6M parameters

A good alternative could be the Ensemble Medium that reach 96.4% of accuracy with 1.3M parameters.

* The smallest model is the DSConvSmall + 40MFCC that reach 92.9% of accuracy with 127K parameters
* The fastest model is 1D CNN on raw data that reach 93% of accuracy with 250K parameters and 28ms of speed prediction.

**Performances 21-cmd**

For the 21cmd task:

* the best accuracy of 95.2% is reached by the Ensemble large model but with 2.5M parameters

A good alternative could be the Ensemble Small that reach 94.2% of accuracy with 1.3M parameters.

* The smallest and fastest model is the 1D CNN on raw data that reach 89.1% of accuracy with 250K parameters and 28ms of speed prediction.

**Conclusions**

**Conclusions**

* The tests showed that our model was **very good at classify keywords**.
* Unfortunately we didn’t beat the state-of-the-art models.
* We found that the number of convolutional layers played a key role in detecting high-level concepts
* There is not so much difference between using 80 Mels or 40 MFCCs.
* 40MFCC +deltas actually perform worse than just 40MFCC
* We presented different model sizes in order to fit different devices.

**Future Work:**

* Try different hyper-parameters during training
* Change the **structure** of the **network** using:
  + SincConv
  + GDSConv
  + Tenporal convolutions
* Build a Feature extractor with convolutional autoencoder