**Presentation Stefano**

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 in 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.

So, the **metrics** we chose for the project are the accuracy of the model, it’s number of parameters and it’s prediction speed.

**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 the Mel spectrogram or the Mel Cepstral coefficients, 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**

We can have different audio input:

We can record a clip from the microphone

or reding it from a numpy array or a wav file

Now I will record a 10 seconds audio.

YES NO UP STOP

Then we can see the waveform of the clip.

We load the model.

With the sliding window technique, we process the clip.

We found that it isn't actually needed a huge amount of windows per second, just with 2 we can have good result.

Now we will use 5 windows per second.

We have a total of 46 windows.

We can have some mistakes, but they will be fixed with the fusion rule.

The average prediction speed per window is ...

But we noticed that predicting 1 single window or a batch of 10 requires the same amount of time.

This because TensorFlow works with tensors.

So this fact should be taken into account for the final real implementation, to reach a better prediction speed.

Than with the Fusion Rule, we can see that all the errors are now fixed.

And finally we can plot the waveform again.

To see that all keywords are correctly predicted.

**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 able to solve the problem:

* No feature extraction: we trained a specific model (1DCNN) directly on the raw signal;
* Mel spectrogram with 80 components
* 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 4 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,
* it uses 250K parameters
* and has a prediction speed of 28 ms, that is the fastest compared to all the other models.

Now I let to Luca present the other models and final results.