Keyword Spotting using CNN+RNN architecture for avoiding overfitting and reducing time

David Altamirano[†], Stefano Meza [‡]

***Abstract*—Nowadays, keyword spotting is a crucial step that must be taking in order to advance to the next era for communication between humans an devices. Many methods for training algorithms exist but the most relevant are related with machine learning, specially with Neural Networks. In this work we tested a generic Convolutional Neural Network (CNN) with a dataset made of 30 labels. It was found that the Short Time Fourier Transform (STFT) feature extractor is the best in terms of efficiency and accuracy but can be still improved. A second model architecture based on Recurrent Neural Networks (RNNs) was tested and for avoiding overfitting, regularization methods were implemented. Finally, it was found that the most efficient and low complexity model is STFT with a l2 regularization of 0.01 and 0.3 dropout value.**

I. Introduction

The rapid advancement of machine learning has fueled significant progress in the field of keyword spotting, which is a critical component of voice-activated systems. In a few words, keywords spotting enable devices such as smartphones, robots, cars to understand and to perform different activities. This facilitates hands-free operations and improves the interaction between human and electronic devices. Nevertheless, when this kind of technology is trained, for instance, training neural networks for recognizing words, sometimes over-fitting may lead to poor performance in real life. As a consequence, in this work, we perform a researching in one of the basic models with a few improvements. Testing different audio features using Pyaudio and Tensorflow libraries and focusing on low complexity, efficiency and mostly important, avoiding over-fitting as much as possible.

This report is structured as follows: in Section II is summarized the state of the art in the keyword spotting field, then Section III the general pipeline of the algorithm is presented. Section IV is focused on explaining the data pre-processing and the treatment for the feature extraction. Efficient methods, avoiding over-fitting techniques and model architectures are explained in Section V. Section VI displays obtained results with their respective metrics. Concluding remarks are provided in Section VII.

II. Related Work

The study of keyword spotting has increased rapidly with the development of mobile devices. In the beginning, mostly of the approaches were HMM-based. Nowadays, the using of artificial neural networks have outperformed them.

Most common are Convolutional neural networks and Recurrent Neural Networks. Sainath and Parada [1] explored CNNs achieving considerable improvements with respect to the previous DNN-based approaches. Specifically, a input signal which has its feature dimension in time and frequency is used for training the neural network. It has convolutional layers for feature extraction, a max-pooling layer which helps to remove variability in the time-frequency space, a dropout layer for regularization, and dense layer for classification.

On the other hand, Mesnil et. al. [2] use a RNN for exploring slot filling in spoken language which refers to refers to the understanding of human speech by machines. One of the employed architecture is Jordan model [3] which includes a form of feedback loop from the output to the hidden layer. The idea is store the output from the previous time step and feed it back to the hidden layer in the next time step, creating temporal dependencies.

III. Processing Pipeline

The main objective of this work is to create a model capable of detecting real time words via microphone with a fast training model and avoiding over-fitting as much as possible. Therefore, for achieving this, it was created an algorithm that can be divided into four different blocks:

1) **Data loading and cleaning**: For creating the speech recognition algorithm it was used a dataset made of 105,829 words belonging to 30 categories. Data was loaded and converted into wavelengths. Then pre-processing took place, trying to reduce the size of the wavelengths and adding noise for improving the predictability of the desired model.
2) **Feature extraction**: In order to train a NN for classifying audio samples, it is necessary to convert them into images, or more specifically, spectrograms. Four methods were used: Short Time Fourier Transform (STFT), Mel-Filterbanks (MFB), Discrete Wavelet Transform (DWT), Mel-frequency cepstral coefficients (MFCCs). The main idea is to obtain the best feature based on: decreasing time consumption and avoiding over-fitting.
3) **Model Compilation**: Extracted spectrograms were divided in training, validation and test set and feed directly

[†]Department of Physics and Astronomy, University of Padova, email: {rossi}@davidalejandro.altamiranocoello@studenti.unipd.it

[‡]Department of Physics and Astronomy, email: .meza-anzules@studenti.unipd.it

Special thanks / acknowledgement go here.

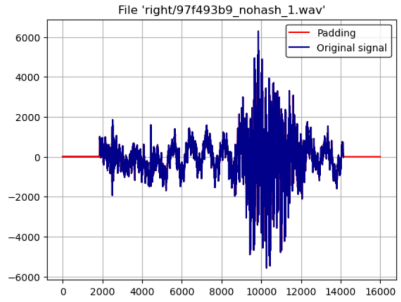

Fig. 1: Padding example

into the model. As first model, it was implemented a CNN for classification task.

4) **Model Optimization**: Best feature was also tested with a CNN+RNN and for assuring the best predictability in any scenario, dropout and regularization was tuned. .

It is important to consider that for speeding up calculations, GPU NVIDIA GeForce RTX 3050 was used just during Model Compilation and CPU was used for the others blocks. (See graph 1 for more details)

IV. Signals and Features

A. Padding

Audio clips have a length of 1 second or less at 16kHz which mean that not all the clips have the same shape. To be able to process them by a neural network is necessary to pad the short clips in order to get all files with the same input shape which is 16000 samples. For performing, the half of the pad is on the left and the other half on the right as can be seen in the Fig. 1. Then, the input data is a vector with shape (1, 16000).

B. Background noise

The dataset has a folder called `_background_noise_` which contains a set of six longer audio clips. Some of them are recorded and other are mathematical simulations of noise. In order to obtain a more realistic approach when the neural network is trained, it can be helpful to mix the background noise with the keyword. We define the function `combine_audio` where we can set the option `apply_noise` as `True` if we want to add the noise and the option `noise_volume_scale` allow us to set the volume of the noise which goes from 0 to 1. Also we can select the type of noise we want with the argument `noise`. The noise choices are `doing_the_dishes`, `dude_miaowing`, `exercise_bike`, `pink_noise`, `running_tap` and `white_noise`. Moreover, as the noise clips are longer than 1 second, we randomly select a portion of them to match the size of the input. An example of the noise addition is shown in Fig. 2.

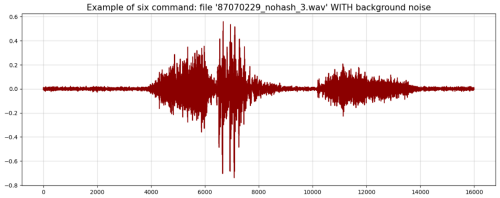

Fig. 2: Example of the noise addition with volume 0.2

C. Feature extraction

The waveforms in the dataset are represented in the time domain. But, to feed and to train the neural network model we need an image, specifically an spectrogram image, which show frequency changes over time in a 2D image. Then, a transformation from the time-domain signals into the time-frequency-domain signals is necessary. There are some mathematical methods and python packages which allow to do it. The simplest and widely used method is the Fourier transform that TensorFlow already has implemented by the function `tf.signal.stft`. Another method is the discrete wavelet transform, `DWT`, provided by `pywt` [4] library. Also `MFCC` and `log Mel-filterbank` allow to extract the audio features and it is implemented in `python_speech_features` [5] library. The application of each method provide a different image shape. The shape of SFTF is (98, 257), for log Mel-filterbanks is (99, 40), for MFCC is (99, 13), and for DWT is (124, 129). The example of the spectrograms is shown in Fig. 3

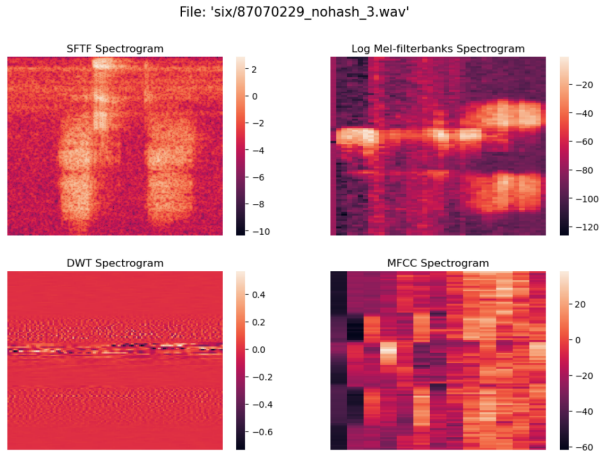

Fig. 3: Example of feature extraction using the 4 methods: short-time Fourier transform (SFTF), log Mel-filterbank (MFB), Mel-frequency cepstral coefficients (MFCC) and Discrete Wavelet Transform (DWT)

D. Partition of the dataset

The partition was done manually and randomly with recommended percentages when a neural network is trained. The 70 % of the dataset which correspond to 74,080 samples is

the training set, 20 % which correspond to 21,166 samples is the validation set and 10 % which correspond to 10,583 samples is the test set.

V. Learning Framework

In order to make the process the most efficient in time and the less over-fitted some considerations were taken:

A. Tensorflow pipeline

All pre-processing and feature extraction were performed using Tensorflow as follows:

- A tensor element was used for storing the .wav files path and their labels.
- Background noise and feature extraction was applied using tf.map. In some cases, tf.py_function had to be implemented due to the non-existence of tf native packages.
- For splitting dataset, tf.take and tf.skip was applied and then tf.batch for batches generation.
- Adittionaly, tf.cache stored dataset in memory for saving time in opening/reading operations per epoch.

B. Layers normalization

Before feeding the spectrograms to the models, layers normalization was performed, reducing significantly the size of the input data. Even though it is shown that accuracy can decrease, it is not a game deciding result.

C. NNs architectures

For both architectures, it was implemented a layer resizing of 32x32 using the layers normalization defined previously. The explained structures are detailed as follow:

1) Generic CNN:

- The first convolutional ReLU layer consisted of 32 filters of square shape (3x3 size, 1x1 stride).
- A second convolutional ReLU layer consisted of 64 filters (3x3 size, 1x1 stride) with max pooling (2x2 size, 2x2 stride). A dropout of 25% was applied.
- A flatten was applied for converting the 2D structure into a flat line (1D).
- A dense ReLU layer of 128 neurons were added and a dropout of 50% was applied.
- A second dense SoftMax layer of 30 neurons, one for each label, were added.

2) RNN+CNN:

- The first convolutional ReLU layer consisted of 10 filters of rectangular shape (5x1 size, 1x1 stride) with padding = same , 0.001 l2 regularized is applied and then a batch normalization.
- A second convolutional ReLU layer consisted of 1 filter of rectangular shape (5x1 size, 1x1 stride) with padding = same, 0.001 l2 regularization is applied and then a batch normalization. A dropout of 30% is applied.
- Lambda layer is applied to squeeze last dimension.
- The first Bidirectional LSTM layer consisted of 64 units with return_sequences=64.
- A second Bidirectional LSTM layer consisted of 64 units.
- A dense ReLU layer of 64 neurons were added, 0.001 l2 regularization is applied and a dropout of 30% was applied.
- A second dense SoftMax layer of 30 neurons, one for each label were added.

D. Dropout and l2 regularization

Deep neural architectures have a natural tendency to overfitting [6]. In order to reduce it and test in in real time, a Gridsearch was applied to the RNN + CNN model with the following values:

$$drop = [0.1, 0.2, 0.3, 0.4]\,, \tag{1}$$

$$l2 = [0.1, 0.01, 0.001, 0.0001]\,, \tag{2}$$

then the best combination, in terms of accuracy and time was selected.

VI. Results

A. Feature comparison

The first test that was performed is shown in Fig. 4. We ran the Generic CNN for 40 epochs using the 4 different features map (STFT, DWT, MFCC, MEL) adding a level of noise of 0.3, and also using STFT without noise as reference. The worst model is the DWT and the best model is the MFCC. MFCC complexity is too high and takes up to 120 min for training. Fastest method is STFT with noise that takes up to 40 min. Based on the trade-off between complexity, time and accuracy, it was decided to use the STFT for the RNN+CNN model.

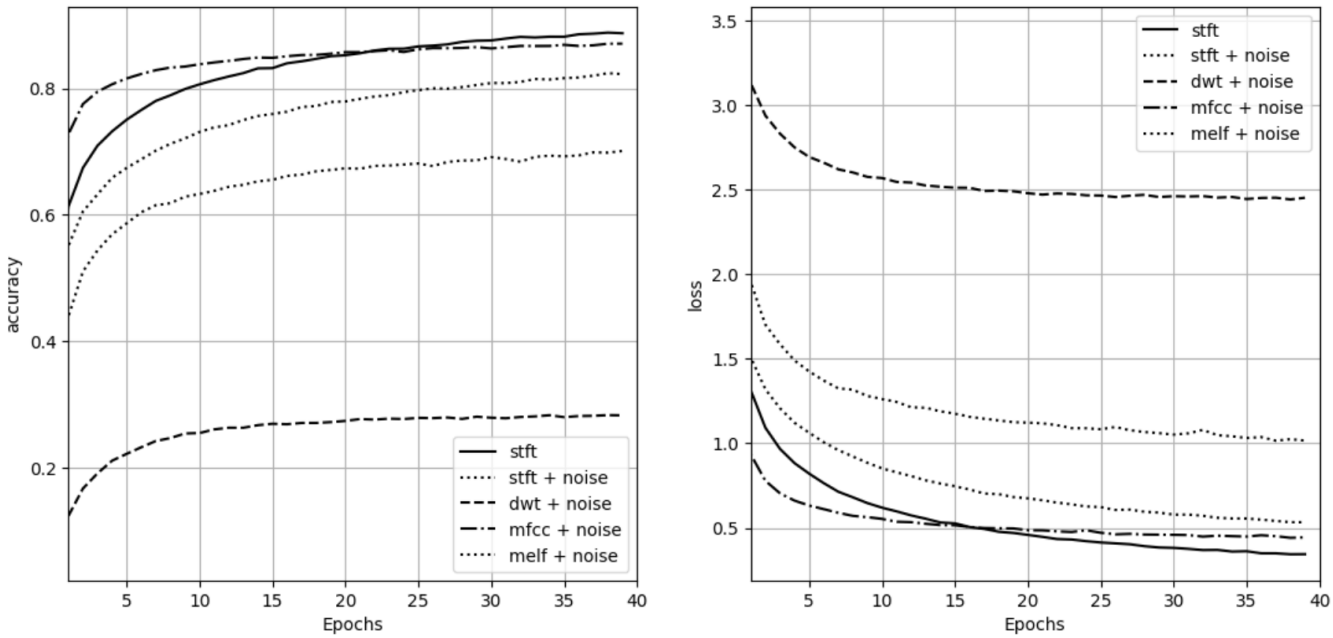

Fig. 4: Comparison between Generic CNN using all the 4 features with noise and STFT without noise for reference

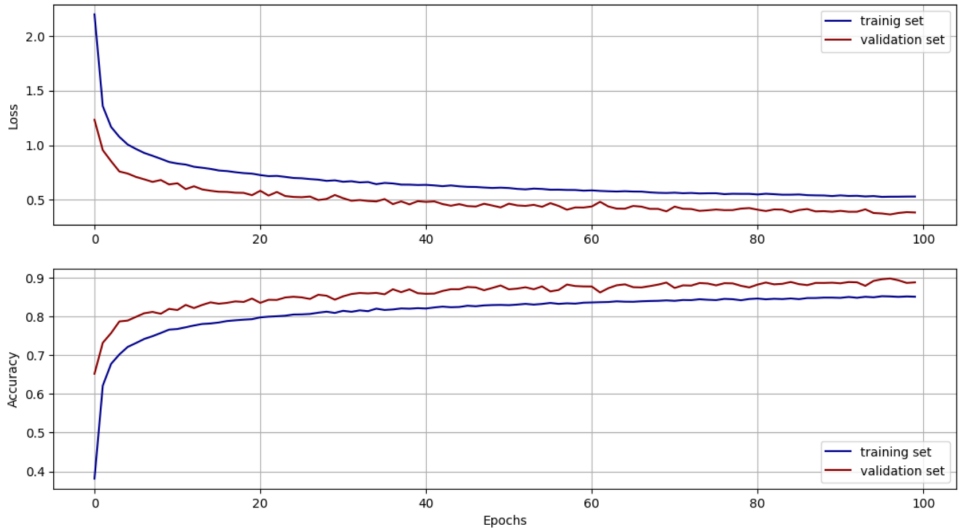

Fig. 5: RNN-CNN model with STFT + noise for 100 epochs with l2=0.001 and dropout=0.3.

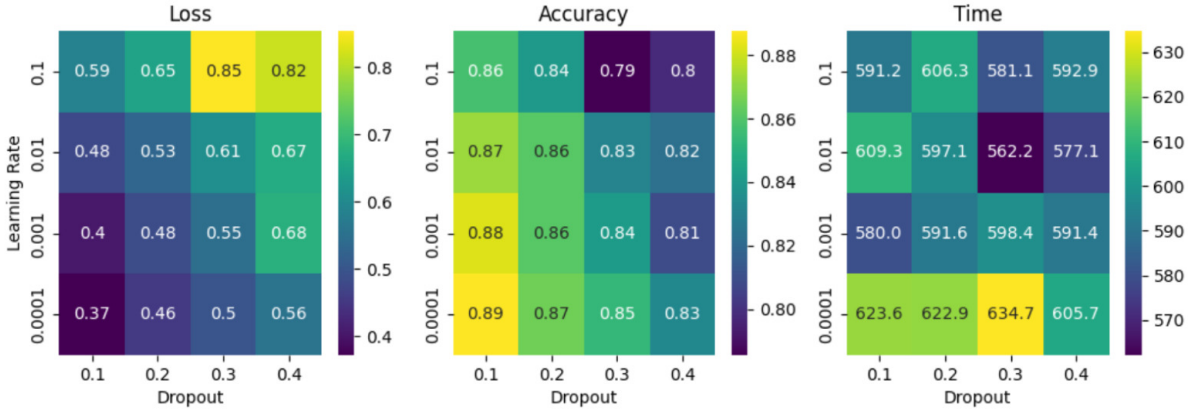

Fig. 6: Heatmap for loss, accuracy and time for different values of dropout and l2 regularization using 25 epochs.

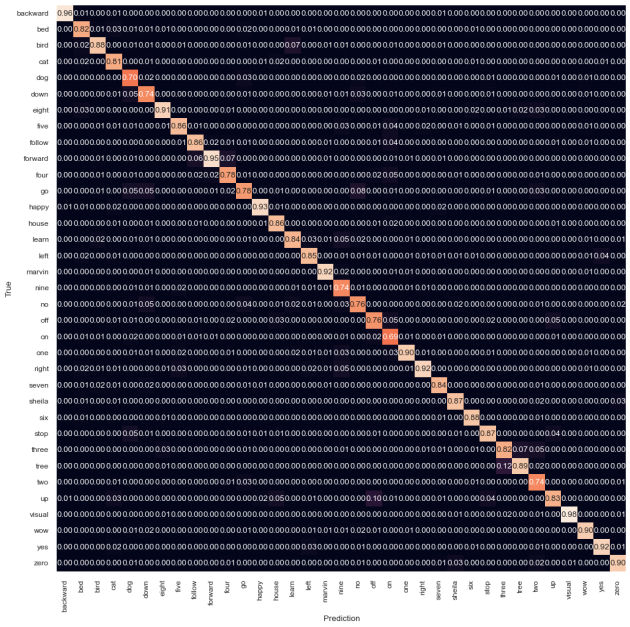

Fig. 7: Confusion matrix using RNN + CNN model trained for 100 epochs using STFT + noise

B. Effect of dropout and learning rate

Then, it was tested the (STFT + noise) using the RNN + CNN with a l2 regularization value of 0.001 and dropout of 0.3. As it can be seen in Fig. 5, accuracy is much more higher which can be good but also may lead to an over-fitting. Therefore we test the model with different values of dropout and l2 regularization and choose the best one. This values were defined in the previous section.

In Fig. 6 it is displayed the heatmap for the combinations of different values of dropout and l2 regularization. In order to obtain the best model we must follow a trade-off between time and accuracy. Therefore the best model for using it as a real-time keyword spotting is: dropout = 0.3 and lr = 0.01.

C. CNN+RNN and best GridSearch values

Finally, the confusion matrix that shows the accuracy of the model used in the test set is shown in Fig. 7

VII. Concluding Remarks

In this project we tackled the keyword spotting task using different methods for feature extraction and exploring a couple of neural network architectures. We started by pushing the performance of a simple CNN and analyzing the impact of regularization and feature extraction techniques. The technique with the best result for loss and accuracy is MFCC and just slightly better than SIFT technique, however if the time is considered, SIFT technique is the best one with huge difference. We then moved towards a more robust model in order to improve the results. The idea was easier, to add a couple of recurrent layers through the method Bidirectional LSTM which gave better results indeed. Finally, to prevent the over-fitting a gridsearch was applied where the best learning rate value was 0.0001 and the dropout value was 0.1, however this combination was the one that took the longest. For this reason and according with our goals, which is prioritize the less time and have acceptable accuracy, we have decided to work with the values, 0.3 for the dropout and 0.01 for the learning rate. This results could be improved for more epochs. For further work, the use of innovative networks will be useful. Improving the architecture considering attention-based architectures or transformers.

References

[1] T. N. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *Proc. Interspeech 2015*, pp. 1478–1482, 2015.

[2] G. Mesnil, Y. Dauphin, K. Yao, Y. Bengio, l. Deng, D. Hakkani-Tur, X. He, L. Heck, G. Tur, D. Yu, and G. Zweig, "Using recurrent neural networks for slot filling in spoken language understanding," *Audio, Speech, and Language Processing, IEEE/ACM Transactions on*, vol. 23, pp. 530–539, 03 2015.

[3] M. I. Jordan, E. Conway, K. Farrelly, J. Grodin, B. Keller, M. Mozer, D. Navon, and S. Parkinson, "Serial order: A parallel distrmuted processing approach," 2009.

[4] G. R. Lee, R. Gommers, F. Waselewski, K. Wohlfahrt, and A. O8217;Leary, "Pywavelets: A python package for wavelet analysis," *Journal of Open Source Software*, vol. 4, no. 36, p. 1237, 2019.

[5] J. Lyons, D. Y.-B. Wang, Gianluca, H. Shteingart, E. Mavrinac, Y. Gaurkar, W. Watcharawisetkul, S. Birch, L. Zhihe, J. HÃ¶lzl, J. Lesinskis, H. AlmÃ©r, C. Lord, and A. Stark, "jameslyons/python_speech_features: release v0.6.1," Jan. 2020.

[6] K. J. Piczak, "Environmental sound classification with convolutional neural networks," in *2015 IEEE 25th international workshop on machine learning for signal processing (MLSP)*, pp. 1–6, IEEE, 2015.