CNN architectures for offline Key Words Spotting systems

Alessandro Fella[†]

***Abstract*—In this paper we study a possible implementation of an offline Key Words Spotting (KWS) system, in particular exploring several architectures of the convolutional neural network family we efficiently solve this kind of task obtaining interesting results. This topic is crucial nowadays because in the last decades several speech-related devices are born: they can use an online/offline stream, or just a single key word, in order to trigger some functionalities. In this scenario we consider a simple CNN, a CNN combined with a Quasi-SVM and an autoencoder with a SVM in the end. All of them are trained and tested with a huge dataset: it is composed by thousands of examples in terms audio tracks and it is organized in 35 classes in which we have common English words.**

Beside the initial target we add a layer of complexity to our problem requiring that the total number of parameters remains below an arbitrary threshold (500K). According to this, we can see that the most efficient approach to solve the proposed challenge is based on the "simple" CNN: we are able to reach an accuracy in the classification task above the score of 90%. We also note that the CNN with a Quasi-SVM is an interesting and lighter way to approximate the autoencoder-SVM.

***Index Terms*—Classification, Optimization, Convolutional Neural Networks, Support Vector machine, Key Words Spotting.**

I. Introduction

In the last decades several speech-related devices are born that have slowly entered in the market and in our daily lives. For example Google offers the ability to search by voice [1] on Android phones, similar tasks can be accomplished using personal assistants such as Google Now, Apple's Siri, Microsoft's Cortana and Amazon's Alexa.
All this kind of tools are based on the idea of a device able, in real time or not, to detect and recognize key words: this has a huge impact on the quality of life and it can help us to interact with technologies in a very smart way. In this work we investigate a very small fraction of this vast topic, in particular using different ANN architectures we try to implement an offline key words spotting system.
In this scenario the core problem is the following: is it possible to write an algorithm that makes a device able to recognize what a person has said? Obviously we can add some extra layers of complexity considering full sentences or entire speech, asking for example a good trade off between memory usage and accuracy in the prediction and so on. We explore different approaches to reach a good KWS system about single key words. In brief, assuming a dataset made by 35 categories with thousands of .wav files each, we study three different CNN architectures: a basic one, a CNN with a quasi-SVM based on [2] and the last one where we combine the autoencoder approach to an hard-coded SVM. According to the obtained results, the first class of models seems to be very powerful and very easy in terms of implementation.
This paper is organized as follows. In Section II we describe some important achievements in the field of KWS while the preprocessing part is fully described in Sections III. The proposed signal processing technique is detailed in Section IV while the framework of the models are explained in Section V. The performance evaluations is carried out in Section VI and concluding remarks are provided in Section VII.

[†]Department of Physics, University of Padova, email: {alessandro.fella.1}@studenti.unipd.it

II. Related Work

In a nutshell KWS refers to the task of detecting keywords or phrases in an audio stream where the detection can then trigger or not a specific action of a device. Early popular KWS systems have typically been based on Keyword/Filler Hidden Markov Model (HMM) [3]–[5]. Even if they were proposed about 20 years ago, they remain highly competitive. In this generative approach, an HMM model is trained for each keyword, and a filler model HMM is trained from the non keyword segments of the speech signal (fillers). These systems require Viterbi decoding, which can be computationally demanding according to the topology of the model [6]. Other recent works explore discriminative models based on large-margin formulation [7], [8] or recurrent neural networks [9], [10]. These systems give an improvement over the previous mentioned approaches but they require processing of the entire utterance to find the optimal keyword region or take information from a long time span to predict the entire keyword, this leads inevitably to an increasing detection latency [6].

In recent years, however, neural network-based systems have dominated the area and they have improved the accuracies reached by other systems. Popular architectures include standard feedforward deep neural networks (DNNs) [6], [11], [12] and recurrent neural networks (RNNs) [9], [10], [13].

During the last years, Sainath and Parada worked using CNNs for low-power applications [14]. According to them CNNs are attractive for KWS since they have been shown to outperform DNNs with far fewer parameters. In their work there are two different approaches, one where they limit the number of multiplications of the KWS system while in the other they limit the number of parameters reaching a relative improvement in false reject rate compared to a DNN fitting the constraints. On this way, firstly applied for image recognition purpose, deep residual networks (ResNets) [15] represent a groundbreaking advance in deep learning that has allowed researchers to successfully train deeper networks. In [16], Tang and Lin explored the applications of deep residual learning and dilated convolutions to the keyword spotting task, their best residual network implementation outperforms Google's previous convolutional neural networks in terms of accuracy. In 2020 [17] Sørensen, Epp and May worked on the implementation of a 10-word KWS system based on a DS-CNN classifier on a low-power embedded microprocessor (ARM Cortex M4).The system described in the study is targeted at real-time applications, which can be either always on or only active when triggered by an external system, e.g., a wake-word system. For sure the studies reported in [14] and [17] are important starting point for future application. They take in account the low power aspect of possible KWS systems: low-power embedded microprocessor systems are suitable targets for running real-time KWS without access to cloud computing [18]. However implementing neural networks on microprocessors presents two major challenges in terms of the limited resources of the platform: (1) memory capacity to store weights, activations, input/output, and the network structure itself is limited for microprocessors; (2) computational power on microprocessors is low. The number of computations per network inference is therefore limited by the real-time requirements of the KWS system.

III. Processing Pipeline

An high-level representation about what will follow is reported in Fig.1. The reference dataset [19] for the study has

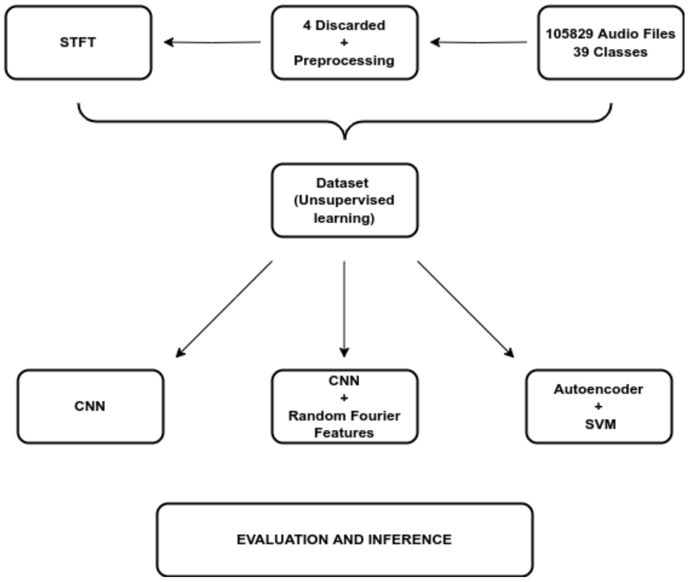
STFT
4 Discarded + Preprocessing
105829 Audio Files 39 Classes
Dataset (Unsupervised learning)
CNN
CNN + Random Fourier Features
Autoencoder + SVM
EVALUATION AND INFERENCE

Fig. 1

been released in August 2017 but the one used is the latest version of April 2018 that contains 105829 audio files. It is organized as follows: each track is basically a one-second-long utterance registered thanks to thousands of different people and there are 39 classes with different numbers (thousands)

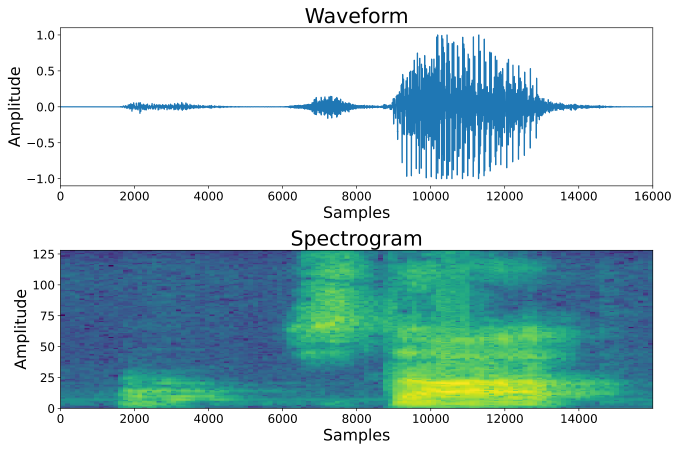

Fig. 2

of examples. In this work the last 4 classes has been discarded since in each of them there were just couple of tracks about environmental sounds while the remaining ones contain basically common words. The very initial part of the project is devoted to extract .wav files from different folders and to organize them in a list to be splitted in training, validation and test set.
Each single file is prepossessed and a couple of functions are created in order to solve two first tasks:

- Extract a normalized waveform associated to a particular audio file
- Store the label starting from the parent directory

According to this a tensor flow dataset is created: it contains the waveform of the different signals and the corresponding labels. This initial dataset is basically represented in the time-domain: performing a short-time Fourier transform, it is converted in the frequency-domain and the full details of the converting procedure is reported in Sec.IV. So we obtain a new dataset with the 2D representation of the different signals (spectrograms) and the associated labels. In Fig.2 we illustrate a simple example of this procedure: in the upper panel there is the normalized waveform extracted from a folder while in the one below there is the result of the STFT. On this way we create 3 different datasets: training, validation and test using a ratio given by 70:20:10. After that we consider three paths to implement the KWS system corresponding to three different set of models and functions. The first is a simple CNN: we use a set of Conv2D+MaxPooling2D and a set of Dense layer with several parameters. In the end the output is basically an array of probabilities for each input: the highest value of such array corresponds to the class to which the input belongs according to the training procedure. With different functions we store the model's weights and the architecture in order to not saturate the memory (RAM and Swap) of the hardware used. Last step is basically the evaluation of the model on the test set checking different metrics to see in detail the performance on the various 35 classes.
The second model, similar to the first, is basically an

approximation of the third one: after the consecutive use of the structure Conv2D+MaxPooling2D to extract features, we use the RandomFourierFeatures [2] exploring different output sizes and taking a kernel initializer set to the Gaussian one. In brief this particular layer is used to "kernelize" linear models by applying a non-linear transformation to the input features and then training a linear model on top of the transformed features. Combining it with the so called hinge loss we obtain a CNN with a quasi-svm for which we perform a set of final steps equal to the ones previously reported.

The last model, the most complex in terms of structure, is built starting from the autoencoder architecture but this requires additional steps to be trained and tested. It basically works with an input of the type [spectrogram, spectrogram] while we have [spectrogram,label]. So we built a set of functions to properly transform the original structure of the dataset and store the labels for the last steps. For what concern the architecture, the enconder part is made by a set of Conv2D+MaxPooling2D with a flatten layer giving at the end in output a 1D representation of the input, its dimension is set before the training procedure by the code size parameter. The decoder is based on Conv2DTranspose layer thanks to which the original input dimension of the "code" is restored. Saving the computed weights, we encode separately the training and the test sets. At this point we merge the code representation we obtained with the labels stored before in order to do the supervised learning using the *SVM* classifier. Building it we are able to perform a classification task in a code-size dimensional space with 35 different classes, we report at the end some evaluation metrics to quantify capabilities of the implemented algorithm.

IV. Signals and Features

The entire project is based on the Speech command dataset [19] that is basically a collection of audio files in .wav format organized in different folders. As just said, they are collected in terms of 1 second tracks thanks to thousands of people. The amplitude values range is between [-32768,32767] while the average number of samples obtained in one second, the so called sample rate, is equal to 16kHz. Starting from here we import the different files in terms of simple waveforms in the time domain normalizing them, in this way the previous range of values is mapped in [-1.0, 1.0] as float32. So we get a set of audio tensors with a structure [samples, channels], where channels is equal to 1 for mono or 2 for stereo. Since the Speech Commands dataset only contains mono recordings, channel dimension is dropped. In Fig.2 we illustrate the example of this procedure.

In order to change the domain of a signal we have to compute the short-time Fourier transform (STFT), in this way we can convert the different waveforms into spectrograms: they basically show frequency changes over time and they can be represented as a 2D image. A basic Fourier transform, computed using for example TensorFlow's functions, converts the signal to its component frequencies but it is characterized by a huge loss of time information, the STFT instead preserves a partial time information since the signal is splitted into windows of time and a Fourier transform on each window is performed. From a practical point of view zero-padding is applied by default, in this way all the waveforms with less the 16000 samples are fixed (basically all input has the same length).

In the conversion of the waveform into spectrograms via STFT, the frame length parameter is set to 22 with a time step of 128. In the end the STFT produces a set of complex numbers so we take the absolute value of them adding an extra channel for the future convolutions in the CNN.

V. Learning Framework

In the following lines we try to summarize the architectures used with their main characteristics.

A. Simple CNN

Since we are basically dealing with images, after the conversion waveform/spectrogram, we built a CNN to classify them. A generic input has the following dimension: [124,129,1] while the label is simply a scalar belonging to the interval [1,35]. The first layer of the model is a Resizing one: it allows us to modify the input dimension in a very easy way including this step directly inside the architecture. After that we have a couple of Conv2D+MaxPooling2D lines to process the image like input thanks to the convolution operation. Here we consider different number of output filters while the activation function and the kernel size are kept fixed, the pool size instead is set to a low value not to lose too much information. In the end there is a Flatten layer followed by different Dense+Dropout layers. Obviously the last line is a Dense layer with a number of neurons equal to the number of classes.

In this study we decided to have the total number of parameters below an arbitrary threshold ($\leq 500K$) for two main reason: lack of computational power and the possibility to implement this kind of algorithm on some devices. According to this we summarize the best configuration found in the following list:

- Input(shape=input shape)
- Resizing(100, 100)
- Conv2D(16, 2, 'relu')+MaxPooling2D(pool size=2)
- Conv2D(32, 2, 'relu')+MaxPooling2D(pool size=2)
- Conv2D(64, 2, 'relu')+MaxPooling2D(pool size=2)
- Conv2D(128, 2, 'relu')+MaxPooling2D(pool size=2)
- Conv2D(256, 2,'relu')+MaxPooling2D(pool size=2)
- Flatten()
- Dropout(0.5)
- Dense(300, 'relu')
- Dropout(0.5)
- Dense(35)

All the computations during the training phase are optimized thanks to the built-in function of tensorflow like *Dataset.cache* and *Dataset.prefetch*, we use intensively also the *.map()* approach to make huge computations saving memory.

The training phase of the architecture, schematized in Fig.3,

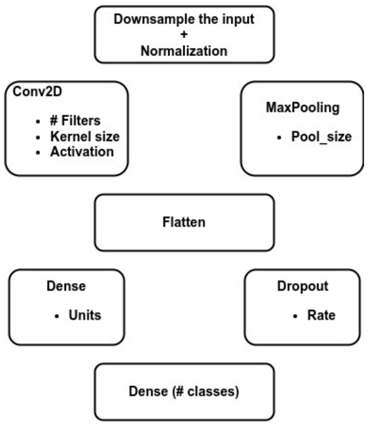

Fig. 3

is based on the Adam optimizer and the sparse categorical loss entropy, the unsupervised learning is performed for an arbitrary number of epochs equal to 50. Since we have a low-power hardware we were not able to work with the full dataset so we consider 1500 examples for labels, in this way we downsize the initial dataset to a global size of 52500 tracks/images.

B. *CNN+Quasi SVM*

Here we built an architecture, schematized in Fig.4, very similar to the previous one: the prepossessing part is the same, we considered again the downsized dataset mentioned in Sec.V-A in order to trade off between number of examples and resources consumption.

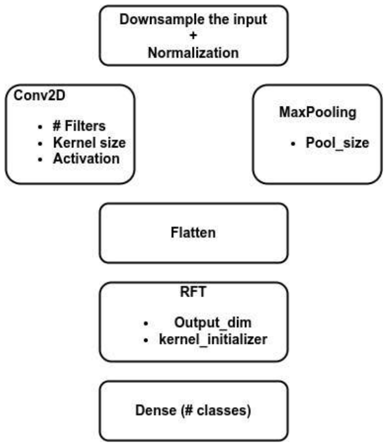

Fig. 4

The first layer of the model is a Resizing one, in this way we work easily with normalized input images of dimension [100,100,1]. After that we consider a set of layers of type Conv2D+MaxPooling2D in order to extract some features from the spectrograms and a flatten layer to transform the elaborated input in 1D vectors .

- Input(shape=input shape)
- Resizing(100, 100)
- Conv2D(32, 2,'relu')+MaxPooling2D(pool size=2)
- Conv2D(32, 2,'relu')+MaxPooling2D(pool size=2)
- Conv2D(64, 2,'relu')+MaxPooling2D(pool size=2)
- Conv2D(64, 2,'relu')+MaxPooling2D(pool size=2)
- Conv2D(128, 2,'relu')+MaxPooling2D(pool size=2)
- Flatten()
- Dropout(0.5)
- RandomFourierFeatures(output dim=800, kernel initializer='gaussian')
- Dropout(0.5)
- Dense(35)

The core of this model is the RandomFourierFeatures layer [2], it can be used to "kernelize" linear model that we obtain with the flattening step by applying a non-linear transformation to the input features and then training a linear model on top of the transformed features. Let's see in detail [20] this particular layer. Consider a learning problem where we have some data (represented by matrices or vectors) and the corresponding labels: $\boldsymbol{x}_n \in \mathcal{X}$ and $\boldsymbol{y}_n \in \mathcal{Y}$. Ignoring the bias, a linear model finds a hyperplane $\boldsymbol{w}$ such that the decision function

$$f^*(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} \tag{1}$$

is optimal for some loss function. Since we are interested in the classification task, a simple linear model breaks down if the data are not linearly separable. For this reason the *kernel method* is used: the input space $\mathcal{X}$ is mapped into a new one $\mathcal{V}$ that has, in general, a higher dimension. This method avoids to work directly in the space $\mathcal{V}$: if $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a positive definite kernel then by Mercer's theorem [21] there exists a feature map $\phi : \mathcal{X} \to \mathcal{V}$ such that

$$k(\boldsymbol{x}, \boldsymbol{y}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle_{\mathcal{V}}. \tag{2}$$

$\langle \cdot, \cdot \rangle_{\mathcal{V}}$ is an inner product in $\mathcal{V}$. Using the kernel trick and a representer theorem [22] kernel methods construct nonlinear models of $\mathcal{X}$ that are linear in $k(\cdot, \cdot)$

$$f^*(\boldsymbol{x}) = \sum_{n=1}^{N} \alpha_n k(\boldsymbol{x}, \boldsymbol{x}_n) = \langle \boldsymbol{w}, \phi(\boldsymbol{x}) \rangle_{\mathcal{V}}. \tag{3}$$

Taken together, Eq.2 and Eq.3 say that provided we have a positive definite kernel function $k(\cdot, \cdot)$, we can avoid operating in the possibly infinite-dimensional space $\mathcal{V}$ and instead only compute over N data points. This works because the optimal decision rule can be expressed as an expansion in terms of the training samples

According to [2] here the point is that the inner product of

Eq.2 can be approximated with a randomized map $\boldsymbol{z} : \mathbb{R}^D \rightarrow \mathbb{R}^R$ where $R \ll N$ i.e.

$$k(\boldsymbol{x}, \boldsymbol{y}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle_{\mathcal{V}} \approx \boldsymbol{z}(\boldsymbol{x})^T \boldsymbol{x}(\boldsymbol{y}). \tag{4}$$

Summarizing if we build a CNN architecture in which after the Flatten layer we use a RandomFourierFeatures layer, combining this with the hinge loss during the learning phase, is possible to implement an approximated version of a CNN with a SVM in the end.

C. Autoencoder+SVM

Here we illustrate the last class of models built. The preprocess is slightly different from the one used before. If we consider $\boldsymbol{x}$, the generic input spectrogram, we used the map approach to have a tensorflow dataset with a structure like [$\boldsymbol{x}$,$\boldsymbol{x}$]. We resize, outside the architecture, each image to have a dimension (100,100,1) and, on the same way, we normalize them.

In the encoder part is possible to transform the input images extracting some features called *codes*, we basically have a way to map the spectrogram in a features array of given dimension that can be changed as input. This can be done with a series of Conv2D+MaxPooling2Dlayers: we consider different number of output filters with the activation function and the kernel size kept fixed. As before the max pool parameter is equal to 2 while the padding is set to ”same”. A Flatten layer followed by a Dense one, with a number of neurons equal to the chosen code size, give us the output of the encoder.

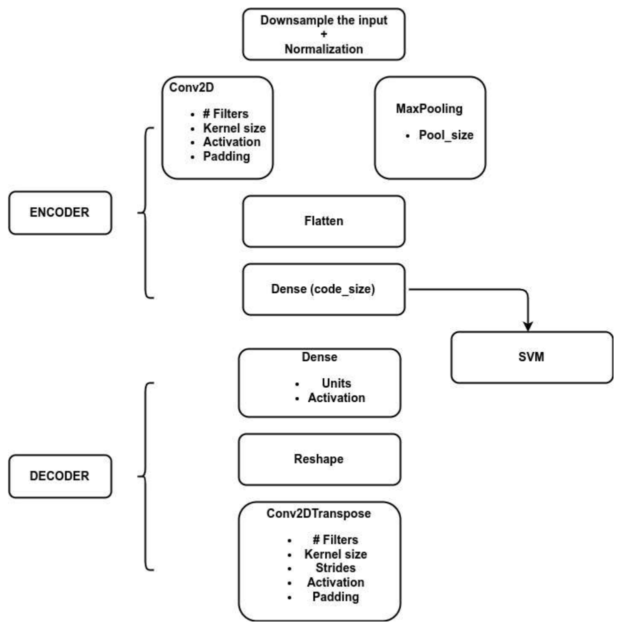

Fig. 5

This is fed to the decoder that starts with a Dense+Reshape, after that we have Conv2DTranspose layers of different size in order to reconstruct the spectrograms with their original (100,100,1) dimensions. Here we have a decreasing number of output filter with the kernel size, the strides and the activation function kept fixed, the padding is set to ”valid”/”same”.

- Input(img shape) $\rightarrow$ Encoder
- Conv2D(32,2,’relu’,’same’)+MaxPooling2D(pool size=2)
- Conv2D(32,2,’relu’,’same’)+MaxPooling2D(pool size=2)
- Conv2D(32,2,’relu’,’same’)+MaxPooling2D(pool size=2)
- Conv2D(64,2,’relu’,’same’)+MaxPooling2D(pool size=2)
- Flatten()
- Dense(code size= 210)
- Input(code size= 210) $\rightarrow$ Decoder
- Dense(6 * 6 * 64, ’relu’) + Reshape(6, 6, 64)
- Conv2DTranspose(128,(3, 3),strd=2,’relu’, pad=’same’)
- Conv2DTranspose(32,(3, 3),strd=2,’relu’, pad=’valid’)
- Conv2DTranspose(32,(3, 3),strd=2,’relu’, pad=’same’)
- Conv2DTranspose(1,(3, 3),strd=2, pad=’same’)

We trained the architecture for about 50 epochs with a lighter dataset respect to the previous mentioned one: it basically contains 750 examples for each of 35 classes.
This choice is made since we faced the problem of trading off between the number of examples for each class, a consistent code size for the model and computational resources. For sure we cannot use a code size too small otherwise the SVM will be not able to reach an higher score in the test phase, on the same way the size of the dataset cannot be too small or too high. In the first case we have poor performance while in the second the system can crush during the learning phase.

For these reasons we considered the configuration written above.

After the training procedure we saved the weights and we loaded just the trained encoder: we encoded the training and the test dataset in order to use them in a SVM. Here since we can work with different hyperparameters we performed a gridsearch, in particular we explored the following alternativeslike:

- Kernel : [rdf, linear, poly]
- C : [10, 100, 1000]
- γ : [1e-2,1e-3, 1e-4,1e-5]

VI. Results

Here we report the main results achieved by our architectures, we will maintain the same structure of the previous section.

A. Simple CNN

The training procedure of the model gave us the following trends where the accuracy saturates to 0.9 while the loss goes down below 0.5. This is basically the best result found taking in account the threshold on the number of parameters: the trained model has 490K of them. In Fig.6 it’s possible to see the accuracy and loss in terms of epochs.

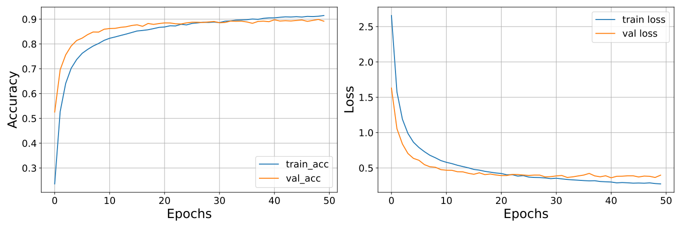
Fig. 6: CNN

The last step is the evaluation on the test set: it contains in mean a number of examples per label of about 150. We obtained a global accuracy equal to 94% and more precise scores are reported in Fig.7,9-10

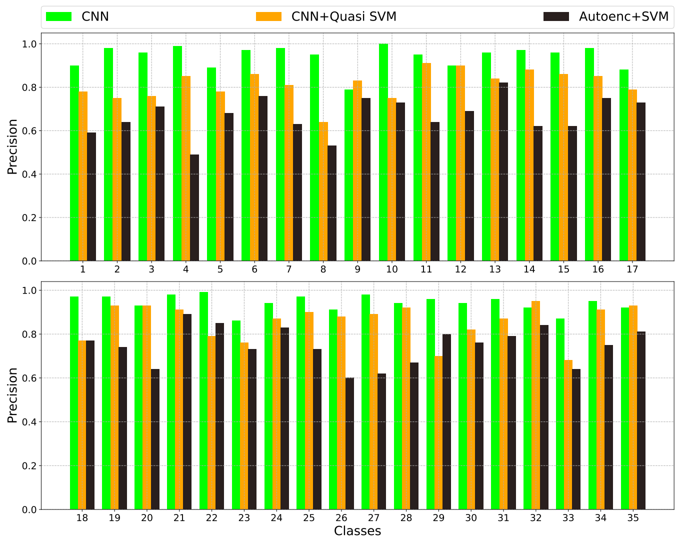
Fig. 7: Precision scores per class/models

B. *CNN+Quasi SVM*

Here the number of parameters are divided in the following way: 90K are trainable while 410K are not. The training of the model gave us an accuracy trend that saturates above the 0.8 value while the loss is close to 0, both of them are reported in Fig.8. As before the testing procedure was performed on a mean number of examples per label of about 150 reaching a global accuracy of 83%, more precise scores are reported in Fig.7,9-10

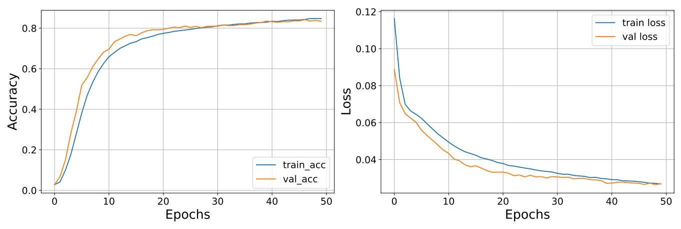
Fig. 8: CNN+Quasi SVM

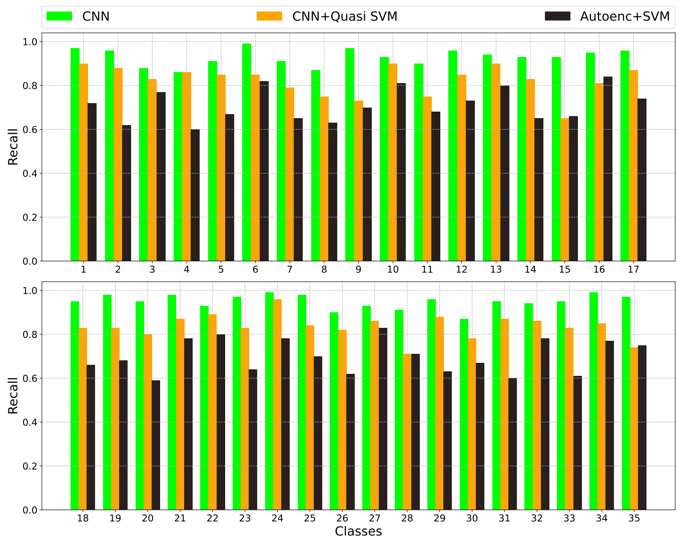
Fig. 9: Recall scores per class/models

C. *Autoencoder+SVM*

In this last part the trainable parameters are about $500K$ for the encoder while $600K$ in the decoder part. After the training procedure we saved the weights and loaded just the encoder to encode the training and the test set. As just said we performed a grid search exploring different sets of values and parameters, we obtained that the best kernel was the *rbf* with $C = 10$ and $\gamma = 0.01$. After we trained the best classifier found and we tested it with a mean number of examples per label of about 75. In the end the score reached for the accuracy metrics is equals to 70% while more precise scores are reported in Fig.7,9-10.

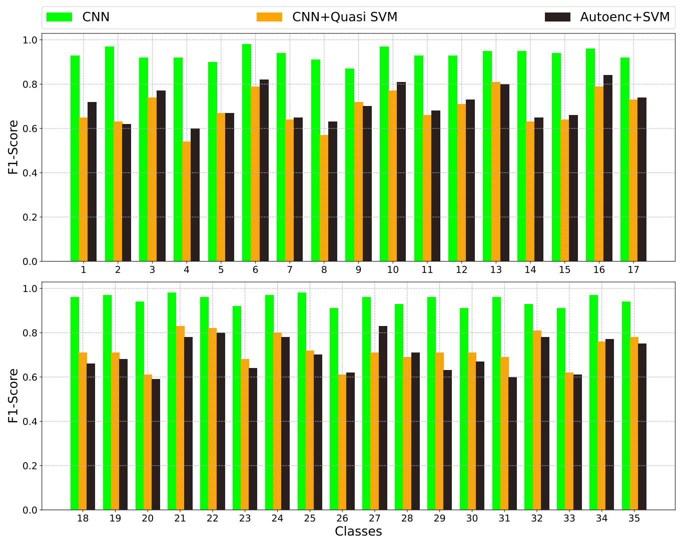
Fig. 10: F1 scores per class/models

VII. Concluding Remarks

In this work we have realized an offline KWS system studying different CNN architectures. We have started from a huge dataset composed by thousands of 1 second .wav files organized in different categories: in order to build a consistent feature space for the classification problem we have decided to work with spectrograms. This was possible thanks to an important preprocessing part in which we have done several operations like: short-time Fourier transform, normalization, resizing and others. All this steps were necessary to train the different architecture we have chosen to build and use for this work. Between the three alternatives the "simple" CNN has achieved the best results per single class in all the metrics. A possible observation we can made is the following: in Fig.7,9-10 we can clearly see that the green lines (simple CNN) are the highest but the dark lines (Autoencoder with SVM) is pretty good despite the smaller number of examples available respect to the other two cases. It could be interesting to see the performance of the Autoencoder with SVM using a bigger test set in order to see if its approximated version, the CNN with Quasi SVM, can be a valid choice for future work. Another interesting point is that we have worked with spectrograms but for sure many other possibilities are available to do the feature extraction phase from a sound track, a very common one is based on Mel-Frequency Cepstral Coefficients (MFCCs). They are a representation of the short-term power spectrum of a sound, based on some transformation in a Mel-scale. It is commonly used in speech recognition tasks as people's voices are usually on a certain range of frequency and different from one to another. So it might be interesting to see how, with similar architecture's configuration, the choice of features affects the final scores.

What I learned and the main problems: Since I choose to do the project alone i had huge chance to learn a lot of things about different topics. I understood how to work with audio file, how to process them in a smart way using the tensorflow built-in functions. I learnt how to make a preprocess pipeline in a light way in terms of RAM and SWAP memory and how to make a CNN from scratch. The main difficulties I encountered are basically two: the absence of brainstorming about problems/ideas to be implemented and the usage and management of the memory of my hardware.

REFERENCES

[1] J. Schalkwyk, D. Beeferman, F. Beaufays, B. Byrne, C. Chelba, M. Cohen, M. Kamvar, and B. Strope, "Your word is my command: Google search by voice: A case study," in *Advances in speech recognition*, ch. 4, pp. 61–90, New York, USA: Springer, September 2010.

[2] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," *Advances in neural information processing systems*, vol. 20, December 2007.

[3] R. C. Rose and D. B. Paul, "A hidden markov model based keyword recognition system," in *International Conference on Acoustics, Speech, and Signal Processing*, (Albuquerque Convention Center, Albuquerque, New Mexico, USA), pp. 129–1321, April 1990.

[4] J. Rohlicek, W. Russell, S. Roukos, and H. Gish, "Continuous hidden markov modeling for speaker-independent word spotting," in *International Conference on Acoustics, Speech, and Signal Processing,*, (Glasgow, UK), pp. 627–630 vol.1, May 1989.

[5] J. Wilpon, L. Miller, and P. Modi, "Improvements and applications for key word recognition using hidden markov modeling techniques," in *[Proceedings] ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing*, (Toronto, ON, Canada), pp. 309–312 vol.1, April 1991.

[6] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (Florence, Italy), pp. 4087–4091, May 2014.

[7] J. Keshet and S. Bengio, *Automatic speech and speaker recognition: Large margin and kernel methods*. John Wiley & Sons, 2009.

[8] S. Tabibian, A. Akbari, and B. Nasersharif, "An evolutionary based discriminative system for keyword spotting," in *2011 International Symposium on Artificial Intelligence and Signal Processing (AISP)*, (Tehran, Iran), pp. 83–88, June 2011.

[9] K. Li, J. Naylor, and M. Rossen, "A whole word recurrent neural network for keyword spotting," in *[Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing*, (San Francisco, CA, USA), pp. 81–84 vol.2, March 1992.

[10] S. Fernández, A. Graves, and J. Schmidhuber, "An application of recurrent neural networks to discriminative keyword spotting," in *Artificial Neural Networks – ICANN 2007*, (Porto, Portugal), pp. 220–229, September 2007.

[11] K. Shen, M. Cai, W.-Q. Zhang, T. Yao, and J. Liu, "Investigation of dnn-based keyword spotting in low resource environments," *International Journal of Future Computer and Communication*, vol. 5, January 2016.

[12] G. Tucker, M. Wu, M. Sun, S. Panchapagesan, G. Fu, and S. Vitaladevuni, "Model Compression Applied to Small-Footprint Keyword Spotting," in *Proc. Interspeech*, (San Francisco, CA, USA), pp. 1878–1882, September 2016.

[13] M. Sun, A. Raju, G. Tucker, S. Panchapagesan, G. Fu, A. Mandal, S. Matsoukas, N. Strom, and S. Vitaladevuni, "Max-pooling loss training of long short-term memory networks for small-footprint keyword spotting," in *2016 IEEE Spoken Language Technology Workshop (SLT)*, (San Diego, CA, USA), pp. 474–480, December 2016.

[14] T. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *Interspeech*, (Dresden, Germany), September 2015.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Las Vegas, NV, USA), pp. 770–778, June 2016.

[16] R. Tang and J. Lin, "Deep residual learning for small-footprint keyword spotting," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (Calgary, AB, Canada), pp. 5484–5488, April 2018.

[17] P. M. Sørensen, B. Epp, and T. May, "A depthwise separable convolutional neural network for keyword spotting on an embedded system," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2020, June 2020.

[18] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," *ArXiv*, vol. abs/1711.07128, November 2017.

[19] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, April 2018.

[20] "detail rff," Dec 2019.

[21] J. Mercer, "Xvi. functions of positive and negative type, and their connection the theory of integral equations," *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character*, vol. 209, January 1909.

[22] G. Kimeldorf and G. Wahba, "Some results on tchebycheffian spline functions," *Journal of mathematical analysis and applications*, vol. 33, January.