Analysis of keyword spotting techniques

Simone Ceccato[†], Federico Venturini[‡]

***Abstract*—Speech recognition is an actual field in the Deep Learning community that periodically sees in the last years improvements on newer neural networks architectures, especially on recurrent neural networks. The state of art of such networks is based on the Attention mechanism or on the Connectionist Temporal Classification (CTC). This paper focuses itself on the first one, with a look also on the initialization of the hyperparameters of the networks and on the best coefficients to represent the audio files, such as the MFCC and the Log Filterbanks. A dataset augmentation with the injection of noise is done on the Google Speech dataset with the creation of 4 different datasets and the use of 4 different neural networks that use convolutions, LSTM, Bidirectional RNNs and the attention mechanism.**

***Index Terms*—Attention, LSTM, Speech recognition, Hyperparameters search, Keyword Spotting, Convolutional Neural Network, MFCC, Filterbanks.**

I. Introduction

Speech recognition is the ability for a electronic device to recognize and translate spoken language into text and is one of the most studied field in the machine learning world due to the fact that it can be useful in a lot of practical applications that ranges from cars to mobile phones and smart house assistants, the market is on a quest for accurate and energy efficient models.

Building an Automatic Speech Recognition (ASR) system that is suitable for a real-world case is a difficult task due to the difficulty to create a robust model to the various variations in natural human speech as changes in the voice pitch, different dialects and pronunciations or the presence of environmental noise.

In this project we are tackling the problem of specific keyword recognition into utterances, a special branch in continuous speech recognition, using deep learning and that has appeared to be used a lot into data from telephone speech [1], air travel information [2], broadcast news task [3], but also in non-business problems, in order to help people with disabilities to execute tasks that otherwise they would be unable to complete [4].

We present a comparison between different types of input features, created using the original signal and that will be presented in *Section IV*, and between different neural networks (that will be explained in *Section V*), trying to extract a performant model capable of achieve a real-world usability. The dataset is composed by 65,000 one-second long registrations of 30 short words and is created by Google using an open-source web applications where any user can contribute to enrich it with new keyword's variations. [5]. In order to increase the dataset size and to uniform each class size, we performed a data augmentation introducing into the utterances a sample of a background noise included in the dataset. This has forced the neural networks to become more robust with respect to noise.

To summarize in this paper we will:

- increase the dataset size adding a class *silence* and new samples with some background noise.
- try different audio features (Log-Filterbanks, MFCC, MFCC with deltas and delta-deltas).
- build different neural networks and perform an hyperparameter initialization in order to grab the best parameters to build the network.
- train different type of neural networks (CNN, CNN-LSTM,Attention-based CNN-LSTM).

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 preprocessing and dataset creation task. In Section V we describe the various architectures used, in Section VI we report their results and in Section VII some extra considerations on them and on some possible developments and future improvements.

II. 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 (HMM). 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 [6] 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 architechture 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.

[†]Universitá degli Studi di Padova, email: {simone.ceccato.1}@dei.unipd.it
[‡]Universitá degli Studi di Padova, email: {federico.venturini}@dei.unipd.it

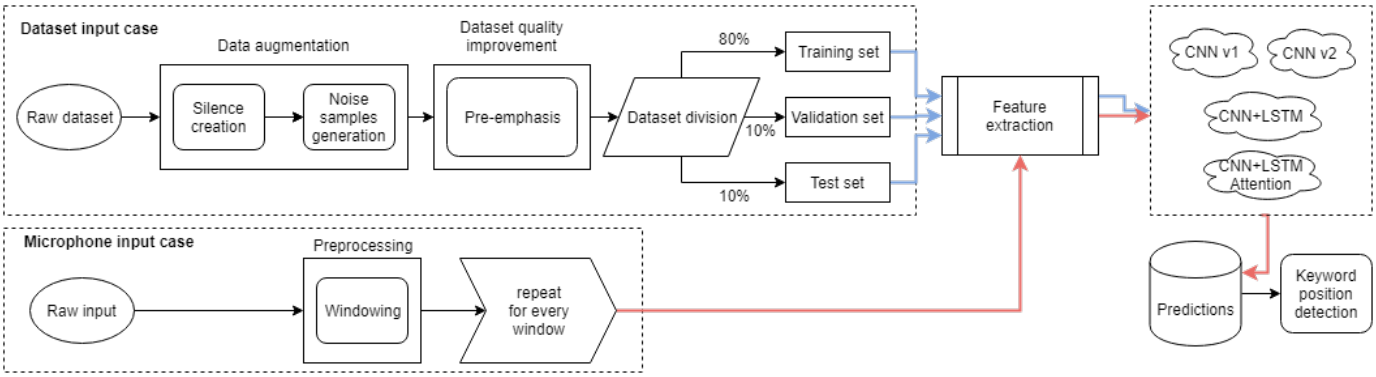


Fig. 1: Processing pipeline of the general application

The first 2 neural networks of this paper are inspired by [7] and [8]. As in [8] our first CNN presents a first layer that scans the input allowing the kernel to move along the time-dimension (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 Englinsh and Mandarin speech to text tasks [11]. Results using raw waveform without any Fourier analysis have also been investigated [12].

III. Processing Pipeline

As we can see from Fig. 1, we built our solution in various compartments that depends on the type of input we receive. In the first case (i.e. **Dataset input case**), we want to train our network using a raw dataset of signals like the Google Speech Dataset; as we can see we have a first part where we perform a data augmentation task in order to uniform the dataset size and to make it robust to noise. After that we generate a new class called *silence* that represent the space between different words and perform a data quality improvement task on all the samples we have. Then we split the dataset into train/validation/test following the proportion 80/10/10% and compute, on all the sets, various features vectors that will be provided to our neural networks in order to be trained and tested.

In the second case (i.e. **Microphone input case**) we receive as input a raw input signal that is not labeled and we want to detect inside it, all the keyword occurences. In order to do this, we create a set of frames of equal duration that will be feeded, one by one, inside the neural network in order to predict the correspondent class. This will give us the possibility to detect the pronunciation of specific keyword, but also to estimate the word position inside the original file and it's duration.

IV. Signals and Features

As stated before in Section I, we used the Google Speech Dataset containing over 100,000 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 first thing we did, is to pad every signal in order to have uniform duration audio of 1 second by adding the needed zeros in front and at the end of the real signal to have a perfectly centered result.

Then we decided to add another class called *silence* that is represented by samples of duration of 1 second: these samples are created taking randomly pieces of audio from the 5 background noises described before and modifying their pitch in order to obtain stronger or weaker signals in intensity with a uniform random weight $\alpha \in (0, 0.3)$.

We decided then to create 2 different datasets:

1) composed by 5 classes

$$classes = \{down, left, right, silence, up\}$$

where each class contains 5,000 files.

2) composed by all the 31 classes where each class contains 3,000 files (for computational reasons).

A. Data augmentation

The following step is to augment the size of each class in order to obtain a uniform distribution of data inside each class: we calculated the number of new samples needed for each class and we generate a new version of a uniformly random chosen file, contaminating it with noise generated following the technique explained before during the creation of the *silence* class. In order to increase more the size and diversity of the dataset we could perform a data augmentation pitching and time-shifting the signals.

B. Dataset quality improvement

In order to amplify the high frequencies we apply a pre-emphasis filter to every signal that can be useful in many

ways like to balance the frequency spectrum since high frequencies usually have smaller magnitudes compared to lower frequencies, to avoid numerical problems during the Fast Fourier Transform (FFT) and to improve the Signal-to-Noise Ratio (SNR). We can represent this type of filter as:

$$y(t) = x(t) - \alpha x(t-1)$$

where α is the filter coefficient and it's usually in the range $(0.95, 0.97)$ (in our case we chose $preemph = 0.97$).

C. Dataset division

After these preprocessing steps, we divided the dataset into train-validation-test sets with the following ratio 80-10-10. The training set is used to train the network, while the validation one to compute the network performance during the training and use it 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.

Before the training we perform a pre-training in order to compute the best hyperparameters for the network (this will be explained later in Section V). During the pre-training, we use a reduced version of the dataset, containing only 30% of the total samples for each set. This pre-training procedure is very time consuming so we applied it only on the first 2 CNN.

D. 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:

- **Log-Filterbanks:** the logarithm of the filterbanks (i.e. the power spectrum in some frequency bands) energies.
- **Mel-frequency cepstral coefficients (MFCC):** coefficients that are obtained after the computation of the Discrete Cosine Transform (DCT) on the Log-Filterbanks.
- **MFCC + Delta + Delta-Delta:** matrix containing the MFCCs previously computed, their first derivative (Delta) and their second derivative (Delta-Delta).

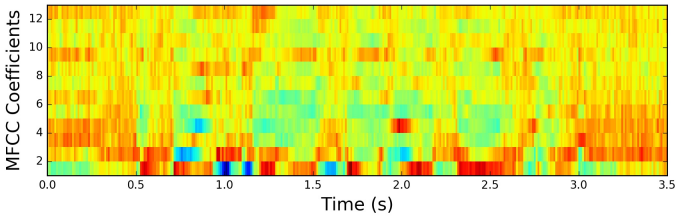
All theses features have in common these steps:

1) we apply a windowing technique to the signal with $window\_length = 0.02\ s$ and $window\_step = 0.01\ s$; given that our signals are 1 seconds long we obtain 99 windows.
2) for every window, we compute the estimate of the power spectrum using the FFT in the interval $(0, F_s/2)$ where F_s is the frequency sampling of the signal (i.e. $F_s = 16000\ samples/s$ in all our samples).
3) we perform a summation of the energies contained in the frequency bands defined by the number of filterbanks we want to use (i.e. $nfilt = 40$ in our case). These summation represent the filterbanks energies we have defined before.
4) we take the logarithm of these summation in order to obtain the **Log-Filterbanks** (in our case the ouput shape is *(99,40)*).

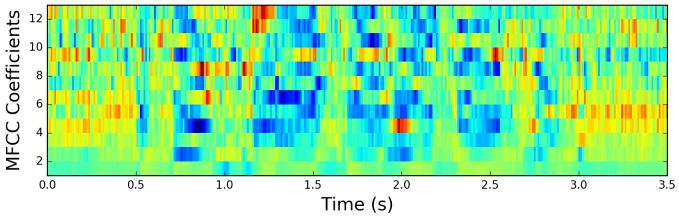
At this step, we have obtained the first feature we wanted; In order to obtain the other three we have to:

- given the Log-Filterbanks, we compute their DCT in order to decorrelate the coefficients and we obtain $ncep$ coefficients (in our case we set $ncep = 12$). These are our **MFCCs** and their output shape is *(99,12)*.
- five the MFCCs, we compute **Delta** feature vector performing the first derivative of the MFCCs and the **Delta-Delta** feature vector doing the first derivative of Delta vector. Both of them have output shape equal to (99,12). In order to create our third network input, we append to the MFCCs both the Delta and Delta-Delta vectors obtaining a matrix of shape (99,36).

After all these steps, in order to improve the SNR, we perform a **Mean Normalization** where we subtract the mean of each coefficient from all the frames (we can see an example in Figure 2).



(a) Raw MFCC coefficients



(b) Mean-normalized MFCC coefficients

Fig. 2: Result of the Mean Normalization on a set of MFCC coefficients

V. Learning Framework

As said before, we implemented different techniques in order to improve the training and the overall accuracy of the networks. Due to the computation complexity, we have implemented these procedures only in the first two CNNs.

A. Optimization techniques

1) Automatic best hyperparameters search: Machine learning algorithms typically have configuration parameters, that influence their output and ultimately predictive accuracy. Hyperparameter tuning can drastically improve the performance of a model. The easiest way is to follow a random search [13] or a grid search on the hyperparameters multi-dimensional space (Figure 3).

However, both methods scales poorly with the number of hyperparameters. There's also the possibility to follow an adaptive approach, using information from previously trained

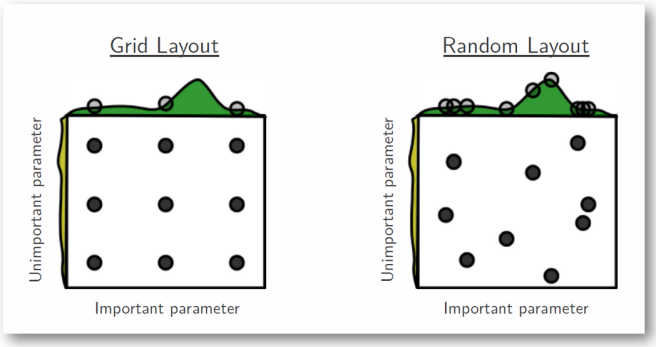


Fig. 3: Grid and random search

configurations to inform which hyperparameters to train next. Now there exist some smart algorithms, such as the ASHA [14] algorithm that are able to use the cluster computing power of many GPUs to speed up the initialization of the hyperparameters. In this work we tried to construct a sort of Iterative Adaptive Hyperparameter Optimization Search (IAHOS). The user selects which are the minimum and the maximum value for each hyperparameter that must be tuned and how many attempts (for each round of the algorithm) and how many round that he wants. Then the algorithm starts computing a small number of epochs (in our cases only 1 most of the time) for each possible combination of the hyperparameters limits values defined. Then it tunes each of them trying some newer values (number defined by the user: the attempts number) near the best ones observed until now. The algorithm in this way computes a grid non-uniform search that can be set also to a random search on the best region found until now. It's also an iterative approach because the duration of the procedure is determined by the user. At the end of this pre-training, the best hyperparameters values are saved for successive real training and a function creates a plot of the initial hyperparameters initialization and the last situation studied by the algorithm (Figure 4 and 5). In this way it's possible to see which where the performances on the limits inserted by the user and which are the best ones discovered at the end. The result is an hyperparameters table in which you see how all the features of the neural network affects the performances in terms of training and validation accuracy (some of them are more important than others) and which are their best values. We used the improvements of the validation accuracy but it's also possible to use the training accuracy.

These are then used to train the network for an higher number of epochs with respect to the one used during pre-training. The pre-training used on this paper is made only on the features that define the architecture of the neural network, but there exist other hyperparameters that have their hyperparameters too, such as the optimizer, the loss and the metric used to compute the gradient and improve the performance of the neural network. We didn't apply IAHOS also on these hyperparameters only for a matter of time,

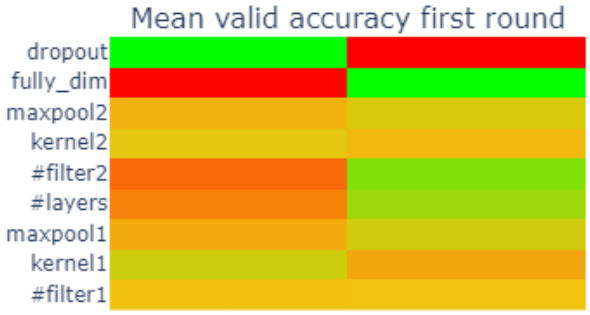


Fig. 4: IAHOS first round

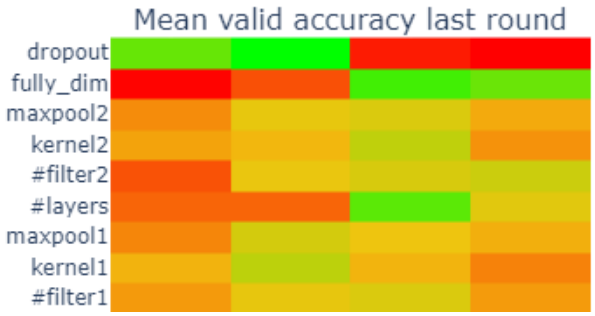


Fig. 5: IAHOS second round

but the code is constructed to allow it. However, on all the networks we tried different optimizers in their default version defined in Keras: *sgd, adam, rmsprop, adagrad, adadelta, adamax, nadam*. Moreover we tried a new optimizer, *RAdam*, that should be able to learn well independently from its learning rate. [15]

.

2) Early stopping and Dropout as regularizers: Regularizers techniques help avoiding overfitting, such as l1 and l2 regularizations. On all this work we didn't use any form of regularizations on the neural networks weights, but instead we applied early stopping and dropout techniques. Early stopping uses the validation set during the training procedure to save the model with the lowest validation error, avoiding in this way a possible over-fitting: instead of using the model obtained in the last epoch of the training, we use the one saved on disk by early-stopping. This technique has the effect of restricting the optimization procedure to a relatively small volume of hyperparameter space in the neighborhood of the initial parameter values. Early stopping has the advantage over weight decay in that it automatically determines the correct amount of regularization while weight decay requires many training experiments with different values of its hyper-parameters [16]. Also Dropout can be seen as a bagging technique (techniques thought for reducing generalization error by combining several methods) [16].

B. Models

Now we present the models we have implemented:

1) CNN v1

2) CNN v2
3) CNN + LSTM
4) CNN + LSTM with Attention

As we can see, all the model have in common a convolutional part that is useful to extract the important features of the inputs we give to them.

1) CNN v1 and CNN v2: The first two networks used in this work are convolutional neural networks with hyperparameters chosen adaptively every time from the outputs of the pre-training (IAHOS). So the architecture can vary independently from the dataset used. CNN1 starts with a Conv2D layer, whose number of filters and kernel dimension along the time dimension are chosen properly during pre-training. The dimension of the kernel along the coefficients dimension is the complete height of the image given in input to the convolutional neural network. So this network tries to follow the architecture described in [8]. Then there's a Dropout layer (used only during training) before a MaxPooling2D layer, whose pool size is chosen with IAHOS too. Then there's a for cycle that allows IAHOS to choose the best number of additional triplet Conv2D-Dropout-MaxPooling layers. After them a projection on a Flatten layer followed by 2 fully connected layers: the first one (with number of neurons chosen by IAHOS) bigger than the last one that must have as many neurons as the classes to identify. The activation used on all the layers and on all the networks (except the softmax on the last one) is the "relu" function, since it avoids vanishing gradient problems (possible with sigmoid and tanh activation function) and makes fast the computation of the gradient from its linearity. CNN2 instead tries to follow work done in [7]. So it has only 2 consecutive Conv2D layers whose number of filter and the dimension of the kernel along the coefficient dimension is always tuned by IAHOS. For the limited hardware resources due to the exponential complexity of the search on the multi-dimensional hyperparameters space, the kernel size along the time dimension is the same for both the layers (for both the CNN pre-training makes a round of 1 epoch on all the possible hyperparametrs values. Having 9 parameters for each CNN to tune, each round of IAHOS has to study 512 possible combinations of them). Each of them is followed by a Dropout layers used only during training and a MaxPooling2D layer. Then, as in the CNN1, there's the projection of the previous layers on a Flatten one followed by 2 fully connected layers as in CNN1. The dropout after the Conv2D layers of the 2 neural networks derives from the result obtained in [17] that makes the MaxPooling a Stochastic Pooling. There also some possible techniques to test in the future such as DropConnect that applies a masks to the weights of neurons instead on their activations.

2) CNN + LSTM: This network implements a convolutional part in order to extract features from the input data combined with an LSTM network; The model will read subsequences of the main time series as blocks, extract features from each block and then allow the LSTM to interpret the features extracted. The network is composed by:

- a convolutional part containing two Conv2D layers, the first one with 10 filters and the second one with 1 filter, both using a kernel size equal to (5, 1) and a ReLU activation function that are alternated with 2 BatchNormalization layers useful to normalize the activations of the previous layer at each batch.
- a Lambda keras layer that removes the last column of the tensor exiting from the previous part.
- a LSTM part containing two Bidirectional LSTM layers with 64 neurons. The Bidirectional wrapper allows the network to get information from both the backward and the forward the current input in order to enhance the performance.
- a Fully-Connected part useful to classify the output of the last LSTM layer.

3) CNN + LSTM with Attention: The last network we have implemented is composed by:

- a convolutional part containing two Conv2D layers, the first one with 10 filters and the second one with 1 filter, both using a kernel size equal to (5, 1) and a ReLU activation function that are alternated with 2 BatchNormalization layers useful to normalize the activations of the previous layer at each batch.
- a Lambda keras layer that removes the last column of the tensor exiting from the previous part.
- a LSTM part containing two Bidirectional LSTM layers with 64 neurons. The Bidirectional wrapper allows the network to get information from both the backward and the forward the current input in order to enhance the performance.
- a Lambda keras layer that extract the vector in the middle of the last LSTM output (this because we assume that the word will be centered in the audio file).
- an Attention layer that receive the previous vector, project it using a dense layer and use as query vector to identify what part of the audio is the most relevant performing a dot product with the output of the last LSTM in order to obtain a weighted average of the LSTM output.
- a Fully-Connected part useful to classify the weighted average.

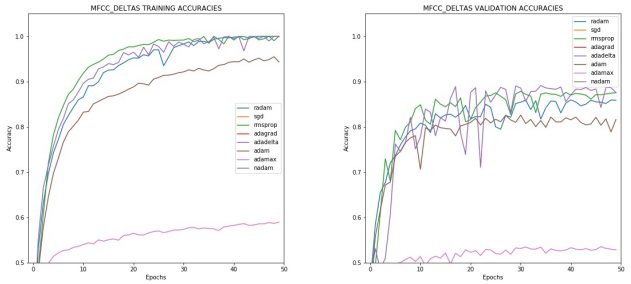
The proposed architecture uses convolutions to extract short-term dependencies, RNNs and Attention to extract long-term dependencies.

VI. Results

We trained each network for all the features we have described in Section IV in order to see if there are any differences between each set. As stated before, we used two different dataset, one composed by 5 class and another one composed by 31 class. We also performed the hyperparameters automatic initialization for almost every network to find the best parameters and to choose the right optimizer in order to maximize the models accuracy on the test data as we can see in TABLE 1.

Feature	CNN v1		CNN v2		CNN-LSTM		Attention CNN-LSTM	
	Dataset 1	***Dataset 2***	***Dataset 1***	***Dataset 2***	***Dataset 1***	***Dataset 2***	***Dataset 1***	***Dataset 2***
Log-Filterbanks	adadelta (91.0%)	adagrad (78.72%)	adadelta (92.7%)	adagrad (80.06%)	nadam (94.3%)	radam* (83.78%)	nadam (95.6%)	radam* (87.81%)
MFCC	adamax (90.1%)	adagrad (77.04%)	adadelta (89.0%)	adamax (74.80%)	nadam (92.8%)	radam* (83.54%)	adadelta (95.0%)	radam* (89.00%)
MFCC Deltas	radam (90.4%)	adagrad (77.43%)	adadelta (88.2%)	radam (69.13%)	rmsprop (92.4%)	radam* (82.95%)	adadelta (95.2%)	radam* (88.78%)

TABLE 1: Best optimizer for each network, dataset and feature with relative accuracy over a test set. * due to computational issues we have trained these networks with dataset 2 only using the RAdam optimizer.



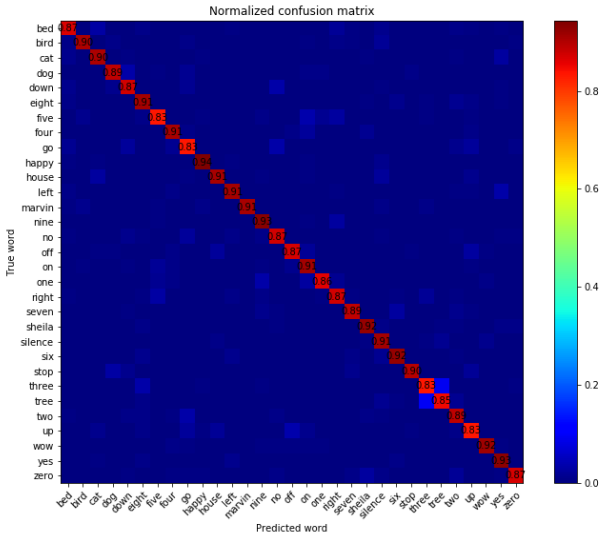
(a) Training accuracies plot (b) Validation accuracies plot

Fig. 6: Low performance optimizers during training and validation process

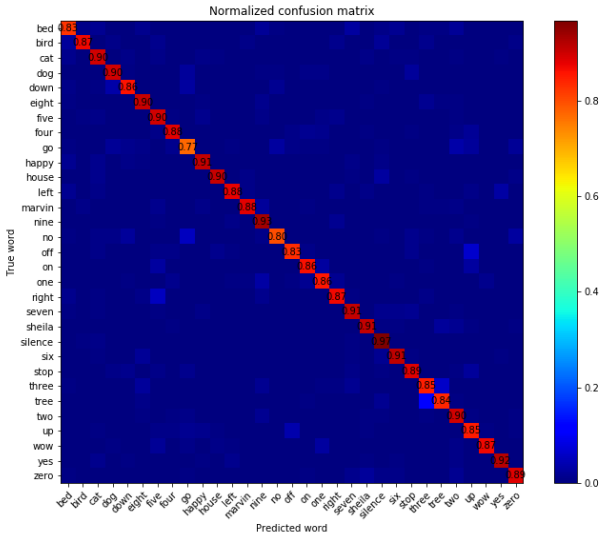
Apart some rare exceptions where the optimizer were not able to allow at the network to learn (see Figure 6), all the optimizers were capable of achieving an high accuracy with all the datasets and all the networks. The full list of the accuracies for each optimizer can be found in the GitHub page of this project[1]. Unfortunately we were not able to complete the hyperparameters initialization with the 31 class dataset for the two LSTM based networks, so for these trainings we have used the RAdam optimizer because we have found that it is the most consistent in the set.

Regarding the feature differentiation, we have seen some differences in how each network react to a different input but we cannot say that the Log-Filterbanks or the MFCC-Deltas outclass the MFCC, even if they have an higher dimensionality and should bring more information, because the difference in the final accuracy is minimum or null like in the LSTM-based networks case. To be more specific, Log-Filterbanks surclassed the other two features when dealing with the first two CNN network but in the LSTM based networks, we did not notice any relevant difference in final accuracy terms. On a deeper analysis of the first case (i.e. when dealing with CNN v1 and v2), we noticed that all the features perform in a very similar manner when the number of classes is low and the gap is very small; instead, with the increasing of the number of classes, the accuracy gap between Log-Filterbanks and the others widens. This phenomenon does not seem to happen in the LSTM-based network: in fact the most performing network in terms of accuracy is a CNN + LSTM with Attention trained using the 31 classes MFCCs dataset, but the gap from the same network trained on the Log-Filterbanks is very close (as we can see in Figure 7). This can be noted also looking at the various accuracies in Table 1.

[1]Github repo: https://github.com/Fede5991/HumanDataAnalytics.



(a) MFCC



(b) Log-Filterbanks

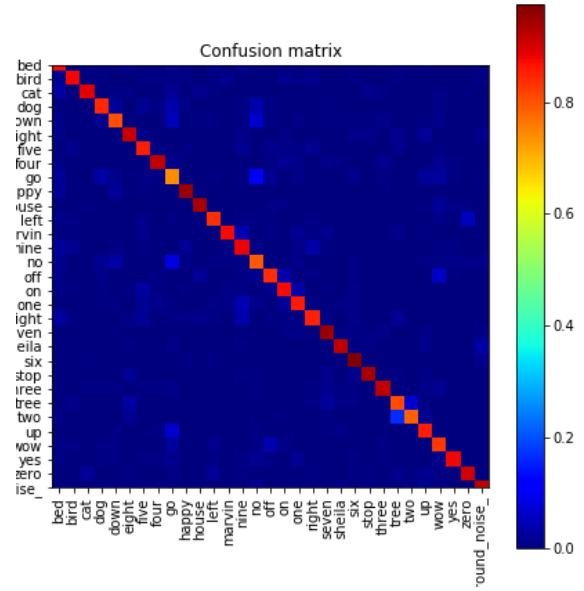
Fig. 7: Confusion matrix on the same test set of two different input neural networks.

Looking the results in TABLE 2, the different networks perform in different manner during the classification task and differ in terms of inference time. As we can see from the table, the two networks based on the LSTM have an higher inference time compared to the first two. This is due to the fact that LSTM networks are heavier than the traditional convolutional

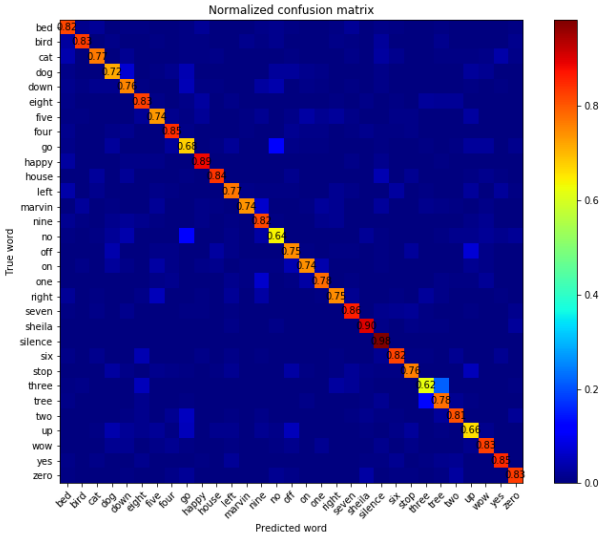
Network	Inference time (μs)
CNN v1	484
CNN v2	602
CNN + LSTM	5121
CNN + LSTM Attention	5503

TABLE 2: Mean inference time for each network

networks and they will take more time in order to process the input data.



(a) Confusion matrix CNN1 raw dataset



(b) Best performing CNN1 confusion matrix with noise

Fig. 8: Differences in the repeating modules

As we can see in Figure 8, we performed a training of the first network with the filterbanks coming from a raw dataset that has not been augmented with the addition of new noisy samples. If we compare its confusion matrix with the best performing network trained on a 31 class noisy dataset, we could see that the network perform in a similar manner with the merit of being more robust to background noise.

If we look these results in a more general way, we could say that each feature or network has its pros and cons: if we are searching for a model capable of working in a limited resource environment or where the real-time response is a key factor, then the CNN v1 and v2 networks together with the use of the MFCCs might be preferred. Otherwise if we are looking for the max accuracy without any regard about the computational complexity of the system, then the Log Filterbanks and the CNN-LSTM with Attention network would be a good choice.

VII. Concluding Remarks

In this paper we addressed the problem of detect which are the best feature vectors to use in order to perform KWS and we tested these input data with different models we have built. As we can see from the results, all the features works well but the Log-Filterbanks outclass the others. this come with a cost consisting on the bigger dimension of this feature respect to the MFCCs. Surely if we have to develop a real-time classifier, our choice would be for the latter ones.

We can also see that the addition of noise inside the dataset affected the final accuracy, but this also created a system that is more robust to noise and more suitable for a real-world application.

In this paper we also verified the importance of the attention mechanism on speech recognition tasks, given the fact that, in terms of accuracy, the latest network outperform the others.

Another possible implementation of this mechanism could be useful on a convolutional denoising autoencoder able to reconstruct the original signal and that it uses the encoder part also for classification. 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 constructed a powerful pre-training mechanism whose potential benefits are beyond our hardware possibilties, opening some great results on clusters of GPUs. Some possible developments in this sense are based on a full pre-training on all the hyperparameters used by our code, parallelizing all the neural networks features and others params concerning the computations of the loss functions, the gradient computations and the batch size. Some possible tests of IAHOS can be done also on the last 2 neural networks studied in this paper.

We would have liked to analyze the behaviour of the Spectral Subband Centroids because they are somewhat more robust compared to conventional MFCC features as well as being partially complementary [18], but we have got some issues in using them inside our neural networks that forced us to remove them from our analysis.

Thanks to this project we have tried several networks and *"played"* a lot with Keras; this experience had shown us the potential of deep learning, in particular in the speech recognition world.

A. Division of the work

The workload was divided in such a way:

- **Dataset study:** Simone Ceccato and Federico Venturini
- **Features extraction, data augmentation:** Simone Ceccato
- **Automatic hyperparameters initialization:** Federico Venturini
- **CNN v1 and CNN v2:** Federico Venturini
- **LSTM networks:** Simone Ceccato

REFERENCES

[1] J. G. Wilpon, L. R. Rabiner, C. . Lee, and E. R. Goldman, "Automatic recognition of keywords in unconstrained speech using hidden markov models," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, pp. 1870–1878, Nov 1990.

[2] M. Weintraub, "Lvcsr log-likelihood ratio scoring for keyword spotting," in *1995 International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, pp. 297–300 vol.1, May 1995.

[3] S. Renals, D. Abberley, D. Kirby, and T. Robinson, "Indexing and retrieval of broadcast news," *Speech Commun.*, vol. 32, pp. 5–20, Sept. 2000.

[4] B. Ben Mosbah, "Speech recognition for disabilities people," in *2006 2nd International Conference on Information Communication Technologies*, vol. 1, pp. 864–869, April 2006.

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

[6] 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.

[7] T. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," 2015.

[8] C.-H. Li, S.-L. Wu, C.-L. Liu, and H.-y. Lee, "Spoken squad: A study of mitigating the impact of speech recognition errors on listening comprehension," *arXiv preprint arXiv:1804.00320*, 2018.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[10] D. C. de Andrade, S. Leo, M. L. D. S. Viana, and C. Bernkopf, "A neural attention model for speech command recognition," *arXiv preprint arXiv:1808.08929*, 2018.

[11] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International conference on machine learning*, pp. 173–182, 2016.

[12] P. Jansson, "Single-word speech recognition with convolutional neural networks on raw waveforms," 2018.

[13] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[14] L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, M. Hardt, B. Recht, and A. Talwalkar, "Massively parallel hyperparameter tuning," *arXiv preprint arXiv:1810.05934*, 2018.

[15] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," *arXiv preprint arXiv:1908.03265*, 2019.

[16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.

[17] H. Wu and X. Gu, "Towards dropout training for convolutional neural networks," *Neural Networks*, vol. 71, pp. 1–10, 2015.

[18] N. Poh, C. Sanderson, and S. Bengio, *Spectral Subband Centroids as Complementary Features for Speaker Authentication*, vol. 3072, pp. 1–38. 01 1970.