Speech Recognition: a Deep study

Yuri Nalesso[†], Ettore Mariotti[‡]

***Abstract*—Speech recognition is a rapidly evolving field. The actual state of the art is represented by the use of artificial neural networks (ANN). In this paper we address the problem of short commands recognition using different features, ANNs architectures and we perform a data augmentation, for a total of 22 experiments. We evaluate also memory/time/accuracy performances and identify suitable trade-off in different scenarios.**

***Index Terms*—CNN, MFCC, Delta, Log Filterbanks, Deep Learning, Speech Recognition, Data Augmentation**

I. Introduction

Speech recognition faces the problem of algorithmically recognize spoken words and correctly classify their meaning. Being able to speak to machines gives a new way of interfacing with them and for this reason many commercial products are leveraging this technology: from cars to mobile phones and smart house assistants, the market is on a quest for accurate and energy efficient models.

Building an accurate Automatic Speech Recognition (ASR) systems isn't a trivial task for many reasons: difference in voice in different individuals, variability of pitch as a consequence of mood, different existing pronunciations for the same word and the presence of additive environmental non-Gaussian noise just to name a few. In order to cope with these problems sophisticated classifiers and highly engineered features were developed trying to mimic the human hearing system.

In this work we face the problem of developing an ASR system using deep learning technologies, which represent the actual state-of-the-art. We purpose different Convolutional Neural Networks (CNN) architectures operating on different audio features (MFCC and Log-Filterbanks, with and without deltas) and we evaluate the different accuracy-complexity trade off on an audio dataset recently published by Google [1]. This dataset is a huge collection of 30 words (classes) spoken by many different people, resulting in a dataset of more than 100000 instances. Even though many of these clips were already recorded in a sub-optimal environment, we artificially corrupted part of the dataset with time shifts and noises in order to set the most realistic stage for our experiments. On this baseline dataset we also performed an artificial data augmentation with time stretch and pitch shift to evaluate the efficacy of this method.

To sum it up we experimented with:

- Different audio features: MFCC, MFCC with deltas and delta-deltas and Log-Filterbanks
- Different CNN architectures: from shallow to deeper versions with many layers, to modified and version of well known networks (Tiny Darknet) [2] to the Inception model.
- Data augmentation

[†]Università di Padova, email: {ettore.mariotti}@studenti.unipd.it
[‡]Università di Padova, email: {yuri.nalesso}@studenti.unipd.it

II. Related Work

The problem of speech recognition began to be investigated in the late 80's with the use of Hidden Markov Models (HMM) with discrete emission probabilities [3]. HMM were used for the powerful property of being somehow resistant to local time-warping [4], an important problem when dealing with speech. Later, the introduction of the Expectation-Maximization algorithm made it possible to use the richness of Gaussian Mixture Models (GMMs) to model the relationship between the HMM states and the audio input [5].

These models were trained with Perceptual Linear Predictive (PLP) [6] or the human-hearing-system-inspired MFCCs (Mel-frequency cepstral coefficients) with their first- and second-order derivative (deltas and delta-deltas respectively) [7], features that could succinctly summarize the large amount of information in raw waveform that is considered irrelevant for classification purpose. MFCCs were particularly successful because they are a set of roughly uncorrelated coefficients that allows the HMM to use a diagonal covariance Gaussian matrix, enabling efficient training.

Despite their success, GMMs suffers from being statistical inefficient at modeling data lying on a non-linear manifold. For this reason researchers moved to Artificial Neural Networks (ANN), highly nonlinear models trained with gradient descent which exhibit a powerful ability to model nonlinear manifolds [8]. The firsts case studies of ANN were in an integration with HMM [9] and with other models such as the Restricted Boltzmann Machines (RBMs) and stacked RBMs (generative models able to represent abstract patterns in data, also called Deep Belief Networks DBN) [10], outperforming the actual state of the art. This motivated further investigation in this direction and with the fortunate advancement of hardware computing power given by GPU, deep learning started being widely explored.

A game-changer in deep learning was the development of Convolutional Neural Networks (CNN). CNN success is due the use weight-sharing and the convolution operation, which is shift-invariant in the data representation domain [11]. This is a powerful feature if one want to achieve speaker-invariance and be robust to time-shifts. In [12], CNN were used for unsupervised feature learning in a DBN-ANN system with high accuracy on phoneme classification. Later CNN were shown to be highly discriminative and able to learn

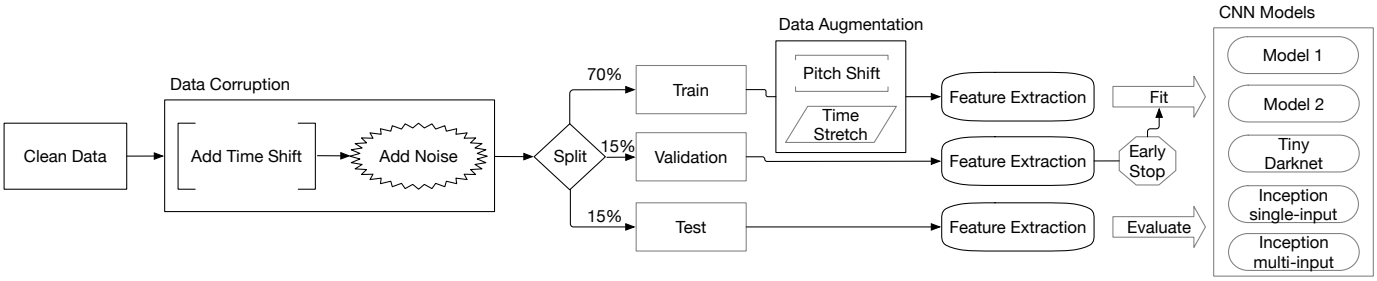

Fig. 1: An exemplification of the processing pipeline of the datase

discriminative and speaker-invariant features in a CNN-HMM model without any DBN [13]. With [14] it was proposed the first solo-ANN (in the form of multilayer perceptron) able to outperform HMM-based methods. A natural and straightforward way to enhance the accuracy of these networks is to make them bigger and deeper, in the sense of adding many layers and enlarge the number of parameters for each layer. This approach have nonetheless some drawbacks such as memory occupancy, difficulties in the training phase and computational load.

A way of overcoming the these problems was addressed in a subsequent work with the use of a CNN in [15]. With the use of parameter sharing and convolution operation on correlated data such as time-frequency audio, the network were successful in learning abstract, discriminating and speaker-invariant feature with a far fewer number of parameters. The advantage were not only in the accuracy, but also in the simplicity of the implementation and in the small memory footprint and computation time, a priority issue for mobile applications.

III. Processing Pipeline

A modern ASR system is made of three parts [14]:

- a *feature extractor* that transform the raw audio signal in a feature space more suitable for classification
- a *classifier* that classify the features instances in the correct classes
- a *posterior handling* that track the temporal evolution of the classifications and give a score to the understood word based on their occurrence.

In this work we focused on the feature extraction and classifying step, leaving the posterior handling (necessary for online applications) as a future work.

The dataset is provided with high quality clips, but since we want to be robust to noise and time shifts typical of a real scenario we time-shifted the raw signal and corrupted it with noise as described in IV-A. After this we split the dataset in train/validation/test following the proportion 70/15/15%; in order to achieve a reproducible experiment we use the hashing algorithm provided by the dataset itself. After the split an augmented dataset is created using the data only of the train set. On these signals are then computed various features: MFCC, MFCC with deltas, Log-Filterbanks. Each feature dataset will be the input of the various proposed neural networks. An exemplification of the pipeline can be found in figure 1.

IV. Signals and Features

The dataset provided by Google is a set of more than 100k audio signals in 30 classes. It also comes with a dataset of 5 different kind of noises:

- dish washing
- dude miaowing
- bike exercise
- running tapisroulant
- white noise
- pink noise

Every instance is less about a 1 second so the shorter words are zero padded and the longer are cut to be exactly 1 second long.

We set up the stage in order to have our classifier being able to distinguish 10 different words and a special category called "silence". We built the silence category by simply adding noise to a zero-signal.

In this setup we are thus discriminating between 11 classes.

A. Noise and time shift

In order to evaluate the classifier in noisy condition typical of a real setting we artificially distorted our dataset. with a probability of 0.1 each audio instance is zero-padded and shifted in the time domain by a random variable Δ uniformly distributed in $(-100, 100)$ milliseconds.

On this shifted signal with a probability of 0.1 a uniformly randomly chosen noise is added with a uniform random weight $\alpha \in (0, 0.4)$.

B. Dataset splitting and Data Augmentation

After the shifting and noise adding process we split the dataset in a deterministic way, using hash names of audio files, in order to reach the reproducibility. Then we created the augmented set, of the same size of train set, modifying each sample randomly with a time stretch and a pitch shift following the results of [16].

C. Features Extraction

We used two different audio features: MFCCs and Log-Filterbanks. We wanted to try the MFCCs as they represent the traditional feature used in speech recognition. MFCCs are a set

of 14 roughly uncorrelated coefficients but since we wanted to exploit the ability of CNN of capturing correlated data, we also experimented with Log-Filterbanks.

The details of feature crafting are the following (see Fig. 2):

1) Frame the signal in overlapping windows of length $25ms$, with a window step of $10ms$
2) For each window compute the estimate of the power spectrum in the interval 0-8000Hz
3) Sum the energy of the power spectrum within some special bands, identified by the so called "Mel-spaced Filterbanks". In this work we used 40 filterbanks.
4) Take the logarithm of the filterbanks energies (This represent the **Log-Filterbanks**)
5) Compute the DCT of the log filterbanks energies (this decorrelate the coefficients)
6) Return 14 DCT coefficients from 2 to 16, discard the rest (which are empirically seen as the most informative). This represent the **MFCCs**.
7) Compute the first order derivative of the MFCCs (**Delta**)
8) Compute the first order derivative of the Delta (also called **delta-deltas**)

We stacked vertically the horizontal coefficients of subsequent windows in a matrix of features of size 99x14 for the MFCCs, of size 99x26 for Log-Filterbanks (we used only 26 filterbanks for them).

This matrix will be the input of our artificial neural network.

V. Learning Framework

We tried 5 different architectures of convolutional neural networks (CNN):

- Model 1
- Model 2
- Tiny Darknet Revised
- Single input Inception
- Multi input Inception

A detailed explanation of the architectures can be found in the tables and figures of the appendix.

We slightly adapted each model for the input of the various features (mostly just the first layer) and we optimized the cross-entropy loss over the 12 class defined in section IV. We used ADAM with an early stopping when the loss computed on the validation set don't decrease of $tol = 10^{-4}$ in 3 epochs, then we proceeded with Stochastic Gradient Descent (SGD) with learning rate $l_r = 0.001$, momentum $\eta = 0.9$, decay $d = 6 * 10^{-5}$ and Nesterov acceleration until convergence defined by the same early stopping of ADAM.

A. Network Architectures Design

Every network have a different design, size and accuracy but all of them rely on the use of convolutional layers with different kind of pooling. The very last layer of each network (with the exception of Model 1) is a dense layer with a softmax activation function. Softmax allows us to interpret the output of the ANNs as the probabilities of the classes for each input instance.

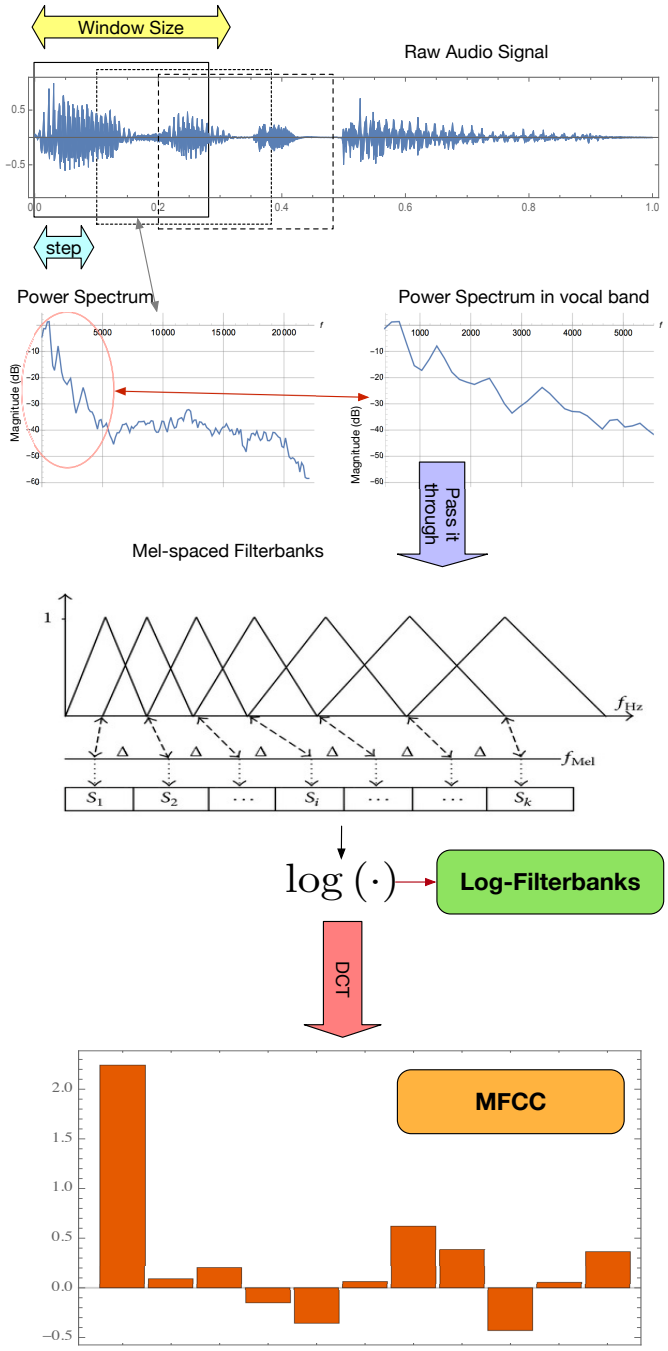

Fig. 2: An exemplification of the feature extraction pipeline

In order to avoid over fitting, regularization strategies have been implemented throught dropout [17], batch normalization layers [18] and global average pooling [19].

The inception model (Figure 8), inspired by [20], is different from the other networks as it is built with the so called inception layer: a layer where filters with different dimensions operate on the same layer and their output is concatenated. We present two variant of the inception: one that operates only on MFCCs (single input) and the other that operates on MFCCs, deltas and delta-deltas at the same time (multi input). The multi input is the model shown in fig 8, the single

input is the same network without the deltas. When operating with MFCCs, since the features can be considered roughly uncorrelated, the first filter of the inception takes all the columns and stride only in the row direction (the convolution is operated solely in the time domain).

On the other hand, we can exploit the time/frequency correlation that is expressed in the representation of audio given by the Log-Filterbanks. This is achieved by using smaller kernels that stride both on rows and columns (the convolution is operated in the time and frequency domain).

Both the inception and the Tiny Darknet Modified present layers of convolution 1x1. This is inspired by [19], as those are equivalent to a multilayer perceptrons that operate on the output of the convolutions and are claimed to enhance generalization.

A detailed view of the topologies is reported in the Tables 1,2,3 and Figure 8.

VI. Results

Each model were trained with different features, the results of these experiment can be seen in figure 3.

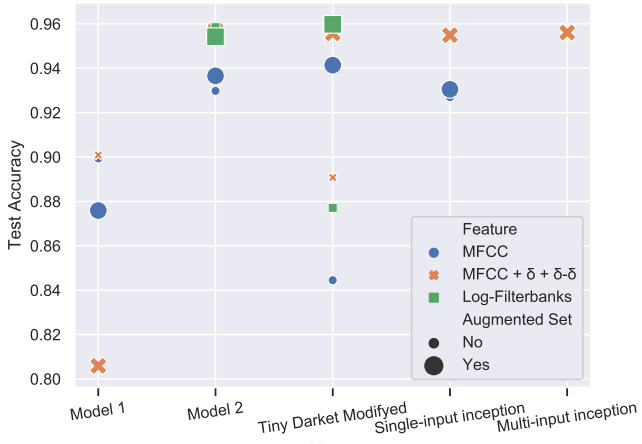

Fig. 3: Compact representation of the performances computed on the test set

Confusion matrices for every model can be found at our GitHub [1]

A. Features

When pursuing this work we wanted to answer the following question: what is the best feature for speech classification using deep learning techniques? The answer is not straightforward, as one can build a perfectly reasonable model that achieve good performances in all the cases we explored. In general as can be seen in fig 7 that the use of deltas and delta-deltas enhanced the performance, but from MFCCs with deltas and Log-Filterbanks we achieve comparable accuracies.

[1]https://github.com/mrektor/HDA_speechCommand

B. Data Augmentation Benefit

As can be better seen in figure 4 the data augmentation provide enhanced performance only in particular circumstances. While there is a general tendency of improvement when using it, models with relatively low depth (Model 1) and a big-size feature (namely the MFCC with deltas and delta-deltas) do not enjoy the same benefit. On the other hand the Tiny Darknet Modifyed, a relatively deep model, do enjoy a great benefit from the augmentation. We hypothesize that this is due to the ability to learn much more abstract representations (i.e. that generalize well) when the network is deep. In this way the model have the opportunity to not fixate on the example we are giving, but to learn the general property of a given pattern in the data.

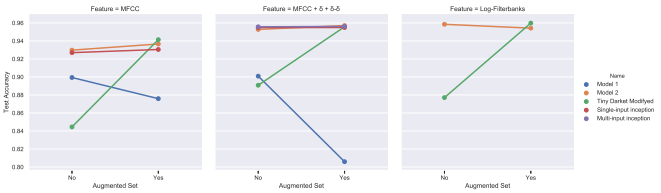

Fig. 4: The impact of the data augmentation for the various models and for the various features (Best seen in digital form)

C. Time-Space-Accuracy tradeoff

As can be seen in figure 6, the single-input inception model is the clear winner when dealing with memory occupancy (see fig 6) and quickness of evaluation (see fig 5), while reaching really satisfying accuracy results. On the other hand the Tiny Darknet have the best classification accuracy (on Log-Filterbanks) but is also the model with the highest number of parameters and a pretty high inference time (probably due to the small size of the kernels, implying many operation on the large Log-Filterbanks data feature).

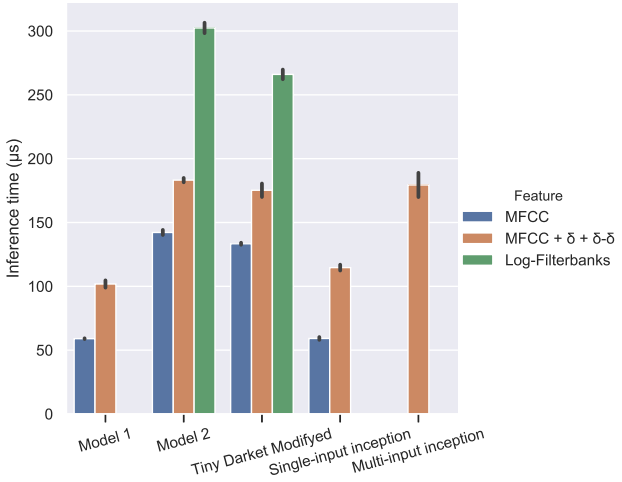

Fig. 5: Inference time of the various models with the different features

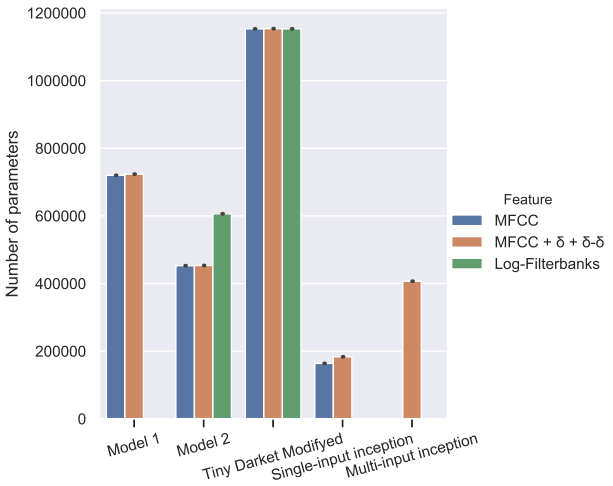

Fig. 6: Size described in terms of number of parameter of the various models

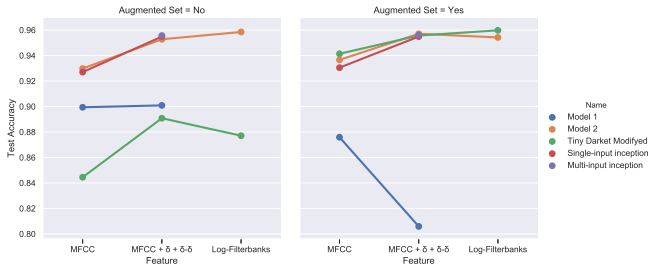

Fig. 7: Performance changing the features

Depending on the required application, the Single-Input Inception model might be preferred to the Tiny Darknet (in a resource constrained environment, like a smartphone). On the other hand if one is interested in high accuracy performances, Tiny Darknet might be preferred.

VII. Concluding Remarks

What we have done is to address the problem of the speech recognition with different feature-type data for different ANN classifiers. Looking at the data, it is possible to understand that the best data are the Log-Filterbanks but are also the slower so a trade-off choice are the MFCCs-delta-delta which performs 0.3% worse but are 40% quicker.

We have acknowledged the benefit of augmenting the dataset with slight variation of the train instances, but also assessed its limitation when trying to use it on a relatively simple network.

The next step to extend the work would be the online recognition of these speech commands, in order to do that there is the need of an unknown class built over thousands of different words. Another item in our to do list would be the hyperparameter optimization to look for the best network's parameters or architectures.

In doing this project we have had a glimpse of the power of deep learning in modeling and providing high quality algorithms for classification purpuses. We explored a wide variety of architectures and of tips and tricks to use for making them work properly.

As it is a quite new research area (and advancing at a considerable growing rate) it wasn't trivial to get a big picture of how and why the deep learning framework work, and still many implementation details are fruit of trial and error. The computational difficulty of training each network was slightly relaxed by the use of a powerful GPU with CUDA processors. We quickly realized that if one wants to explore this corner of the computational universe he must have suitable computing tools for doing so in reasonable time.

References

[1] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *CoRR*, vol. abs/1804.03209, 2018.

[2] J. Redmon, "Darknet: Open source neural networks in c," 2013–2016.

[3] 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.

[4] C. M. Bishop, *Pattern Recognition and Machine Learning*.

[5] P. F. Brown, "The acoustic-modeling problem in automatic speech recognition.," tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1987.

[6] H. Hermansky, "International computer science institute, berkeley, california, usa.,"

[7] S. Furui, "Cepstral analysis technique for automatic speaker verification," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 29, no. 2, pp. 254–272, 1981.

[8] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[9] H. A. Bourlard and N. Morgan, *Connectionist speech recognition: a hybrid approach*, vol. 247. Springer Science & Business Media, 2012.

[10] A.-r. Mohamed, G. Dahl, and G. Hinton, "Deep belief networks for phone recognition," in *Nips workshop on deep learning for speech recognition and related applications*, vol. 1, p. 39, Vancouver, Canada, 2009.

[11] Y. LeCun, Y. Bengio, *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[12] H. Lee, P. Pham, Y. Largman, and A. Y. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks," in *Advances in neural information processing systems*, pp. 1096–1104, 2009.

[13] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, and G. Penn, "Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pp. 4277–4280, IEEE, 2012.

[14] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks.," in *ICASSP*, vol. 14, pp. 4087–4091, Citeseer, 2014.

[15] T. N. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[16] I. Rebai, Y. BenAyed, W. Mahdi, and J.-P. LorrÃ©, "Improving speech recognition using data augmentation and acoustic model fusion," *Procedia Computer Science*, vol. 112, pp. 316 – 322, 2017.

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[18] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[19] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[20] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

APPENDIX

Layer Type	*Details*
Convolutional	64 Kernel of size: (7,4)
Batch Norm	-
Max Pooling	Pool Size: (3,2)
Dropout	Drop Prob: 0.5
Convolutional	128 Kernels of size (4,2)
Batch Norm	-
Convolutional	256 Kernels of size (4,3)
Batch Norm	-
Dropout	Drop Prob: 0.6
Max Pooling	Pool Size: (5,1)
Dense	Softplus Neurons: 100
Dropout	Drop Prob: 0.75
Batch Norm	-
Dense	Sigmoid Neurons: 12
Total parameters:	719'947

TABLE 1: Model 1 architecture. note: all the nonlinear activation functions of the convolutional layers are softplus.

Layer Type	*Details*
Convolutional	16 Kernels of size: (4,2)
Convolutional	32 Kernels of size: (2,2)
Convolutional	64 Kernels of size: (2,2)
Batch Norm	-
Max Pooling	Pool size: (4,2)
Dropout	Drop prob: 0.4
Batch Norm	-
Convolutional	32 Kernels of size: (2,2)
Convolutional	128 Kernels of size: (2,2)
Batch Norm	-
Max Pooling	Pool size: (2,2)
Dropout	Drop prob: 0.4
Batch Norm	-
Convolutional	256 Kernels of size: (2,2)
Maxpool	Pool size: (4,2)
Dropout	Drop prob: 0.3
Dense	Softplus Neurons: 80
Dropout	Drop prob: 0.5
Batch Norm	-
Dense	Softmax Neurons: 12
Total parameters	452'691

TABLE 2: Model 2 architecture

	Layer Type	*Details*
	Convolutional	32 Kernels of size: (3,3)
	Batch Norm	-
	Convolutional	16 Kernels of size: (1,1)
	Batch Norm	-
	Convolutional	64 Kernels of size (3,3)
	Batch Norm	-
	Max Pooling	Pool size: (2,2)
x2	**Convolutional**	16 Kernels of size: (1,1)
	Batch Norm	-
	Convolutional	128 Kernels of size: (3,3)
	Batch Norm	-
	Max Pooling	Pool size: (2,2)
x2	**Convolutional**	32 Kernels of size: (1,1)
	Batch Norm	-
	Convolutional	256 Kernels of size: (3,3)
	Batch Norm	-
	Max Pooling	Pool size: (2,2)
x2	**Convolutional**	64 Kernels of size: (1,1)
	Batch Norm	-
	Convolutional	512 Kernels of size: (2,1)
	Batch Norm	-
	Convolutional	64 Kernels of size: (1,1)
	Barch Norm	-
	Avg Pooling	Pool size: (3,1)
	Dropout	Drop prob: 0.85
	Dense	Softplus Neurons: 80
	Batch Norm	-
	Dense	Softmax Neurons: 12
	Total parameters:	1'153'131

TABLE 3: Modified Tiny Darknet architecture. notes: all the nonlinear activation functions of the convolutional layers are leaky ReLU, the parameters of all the batch normalization are $\epsilon = 10^{-4}$ and momentum = 0.1

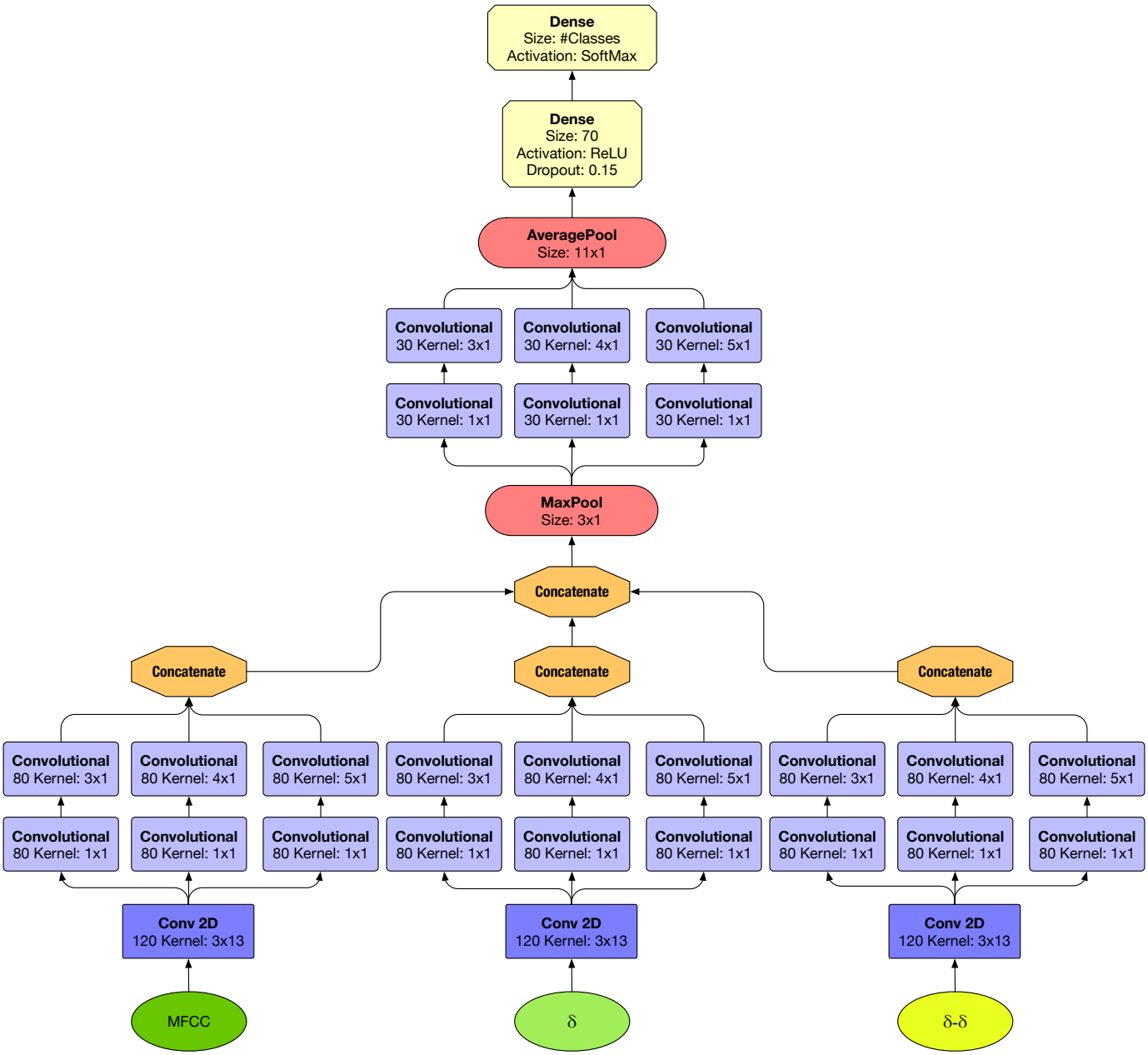

Fig. 8: The multi-input inception model. The single input have the same architecture but without the deltas (until the concatenate)