Depthwise Convolutional Neural Network and Temporal Convolutional Network: a comparative analysis for Keyword Spotting task.

Anna Dorna[†], Simone Trevisan[‡]

***Abstract*—Nowadays, the growing interest in speech interaction with a large number of mobile devices requires the design of an efficient Keyword Spotting (KWS) based system. These algorithms should be able to run locally with limited resources and achieve high accuracy performance to ensure a good user experience. In this work we explore and compare two novel architecture: a Depthwise Separable Convolutional Neural Network (DS-CNN) and a Temporal Convolutional Network (TCN). DS-CNN is designed as an efficient alternative to standard CNN in terms of number of parameters and operations. TCN instead, exploits causal convolutions and dilation factor to achieve parallelism, flexible receptive field and stable gradient. Also, scalability of these architectures is explored to understand how their accuracy change with respect to the size. To train the networks, the Speech Commands Dataset by Google is used: it includes 105k utterances of 30 short words. Audio files are processed and Mel-Frequency Cepstral Coefficient (MFCC) are extracted to feed the models. The implementations proposed achieve an accuracy of 0.89 for DS-CNN and 0.92 for TCN with a number of parameters below 20k. Increasing the number of parameters, the performance does not grow significantly.**

***Index Terms*—Keyword spotting, Mel-frequency cepstral coefficients, neural networks, depthwise separable convolution, temporal convolution.**

I. Introduction

During the last decades, deep learning algorithms have surpassed human accuracies in a variety of cognitive tasks and have improved performances in a wide range of applications, including speech recognition. Since speech has become a prevailing interface to enable human-computer interaction, keyword spotting (KWS), the task of detecting keywords, has become an important component in this context.

This technique has grown in importance due to its efficiency in devices with limited functionality. In contrast with this solution, automatic speech recognition (ASR) systems use large vocabularies that could require access to cloud computation for the demanding processing tasks with possible concerns derived from the necessity of continuous internet access. In recent years, neural networks have dominated the area of KWS systems. Popular architectures include standard feedforward deep neural networks (DNNs), recurrent neural networks (RNNs) and inspired by advancements in techniques used in computer vision, also convolutional neural networks (CNNs). [1]

In this paper a challenge dealing with the limited resources in terms of memory capacity to store weights is faced. Most of the previous mentioned solutions for KWS in fact failed to provide a complete answer. Two different architectures are then here adopted. On one hand, the depthwise separable convolutional neural networks (DS-CNNs) are proposed as an efficient alternative to the standard CNNs. On the other hand, temporal convolutional networks (TCNs) are designed since recently in [2] it was shown that networks combining convolution with causality performs better than both RNNs and Attention based models. [3]

The main contributions of this work are as follows:

- to propose the implementation of two relatively new networks to solve the KWS task;
- to focus on how the network scales accordingly to the number of weights;
- to investigate the trade-off between accuracy and size for the neural networks adopted;
- to compare the results against other existing approaches.

This report is structured as follows. In Section II some related work about neural networks' implementation for KWS is presented. Section III deals with the processing pipeline: an high level description of the approach is reported and the processing blocks are presented. The proposed signal processing techniques and feature extraction are discussed in Section IV, while the learning strategy from a technical point of view is detailed in Section V. Results are carried out in Section VI. Concluding remarks are provided in Section VII.

II. Related Work

Recently, with the success of deep learning in a variety of cognitive tasks, neural network based approaches have become popular for KWS. Several neural network architectures have been implemented. However, although there is an extensive literature in KWS, not all of the proposed methods are suitable for the applications in resource constrained environments. Only some papers faced this issue, as in [4], where the authors limit the number of computations and parameters in order to show the improvements of CNNs over DNNs.

This is the reason why this paper wants to focus on the changes related to the number of parameters while finding the best performant networks. In order to reach this aim, this

[†]Department of Information Engineering, University of Padova, email: {anna.dorna}@studenti.unipd.it

[‡]Department of Information Engineering, University of Padova, email: {simone.trevisan.3}@studenti.unipd.it

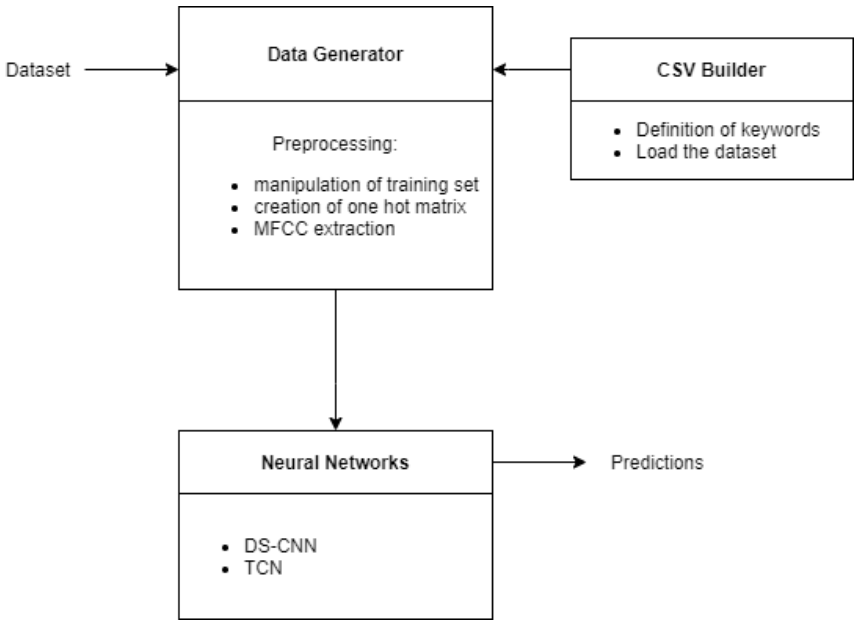

Fig. 1: Processing pipeline

report extends previous efforts to implement a KWS system based on a DS-CNN and a TCN, respectively. Regarding the first neural network, in [5], inspired by the success of resource-efficient MobileNet in computer vision authors further explore the depthwise separable convolutional neural network. In their analysis, they performed resource-constrained neural network architecture exploration and presented comprehensive comparison of different network architectures within a set of compute and memory constraints of typical microcontrollers. They found that DS-CNN achieves an accuracy that is $\sim$10% higher than the DNN model. On the other hand, regarding the TCN, in [6] a temporal convolutional network is designed to classify keywords. Then the performances of the TCN based classifier is compared with other related work in the literature. Also for this second architecture, the accuracies achieved are pretty high if compared with other standard architectures. In this paper, authors referrred also to Multi-Frame Shifted Time Similarity (MFSTS) and their performance is compared against the widely known Mel-Frequency Cepstral Coefficients (MFCC) that are computed in the frequency-domain. However, they found that the achieved classification accuracies are higher using MFCC rather than using MFSTS. Since the reached performances are promising for these two innovative neural network architectures, this paper builds the rest of the analysis by starting with the implementation of these two nets.

III. Processing Pipeline

In this section a high level description of the approach is reported. In the Figure 1 the processing blocks of this project are visible. A Speech Commands dataset for small-footprint keyword spotting is used as input and brings some useful results after some steps. The main blocks developed are the following:

- **CSV builder block.**
  In this block a file .json with each keyword and the corresponding label is loaded. Then, the paths for the different audio files from the dataset are saved in three different .csv files. Trainset.csv, validation.csv and test.csv are created with these three columns: 'file_name', 'label', 'class_name' (see csv_builder.py notebook).
- **Data generator block.**
  In this block, a generator is defined in order to load the batch of audio files and preprocess them in a memory efficient manner (see datagenerator_v2.py notebook).
- **Preprocessing block.**
  In this block three functions are defined. The first function regards some preprocessing techniques, the second deals with the creation of one hot matrix and the third consists of the Mel-Frequency Cepstral Coefficients (MFCC) extraction (see utilitypreproc.py notebook).
- **Neural network model block.**
  In this block, the iteratively produced batches of the generator block are used as input for the neural networks. The implementation of depthwise separable convolutional neural networks and of temporal convolutional networks are part of this process. The training and the evaluation of the implemented architectures are then performed (see dscnnx.py and tcn.py notebooks).

The trained models with different hyperparameters are saved and numerical results are obtained from the different architectures. Plots, tables and confusion matrices to describe the results of the classification are then realized.

Parameters	Explanation	Values used in the project
signal	the audio signal from which to compute the features	16000*1 array
samplerate	the samplerate of the signal	16000
winlen	the length of the analysis windows in seconds	0.02 or 0.04
winstep	the step between successive windows in seconds	0.01 or 0.02
numcep	the number of cepstrum to return	10 or 13
nfilt	the number of filters in the filterbanks	13 or 26 or 40
nfft	the FFT size	the smallest power of 2 bigger than the number of seg
lowfreq	lowest band edge of mel filters	300
highfreq	highest band edge of mel filters	samplerate/2
winfunc	the analysis window to apply to each frame	hamming

TABLE 1: Parameters of mfcc function from python speech features library

IV. Signals and Features

The Speech Commands dataset [7] used in this project consists of 105,000 one-second long utterances of 30 short words pronounced by thousands of different people contributed by members of the public through the AIY website. Different speakers' contribution to the dataset ensures a high speaker diversity. In this paper the following 10 words are used as keywords: 'Down', 'Go', 'Left', 'No', 'Off', 'On', 'Right', 'Stop', 'Up' and 'Yes'. A label from '0' to '9' is assigned to each class name. The remaining 20 words of the dataset are used as fillers and the label '10' is assigned to them.
The dataset is formatted as a collection of WAV files sampled at 16KHz.
The first step applied to the input consists in signals preprocessing for the training samples. The following are the techniques applied to the audio:

- **Shifting time** that just shifts audio to left/right with a random time and set to silence for heading/tailing depending on the direction of the shift.
- **Noise injection** that simply adds some random value into data by using numpy.
- **Changing pitch** that was constructed as a wrapper of `librosa` function and changes pitch randomly.

Since we noticed that a few samples have not exactly the same length, `resample` function from library `librosa` is applied in order to assure the same samplerate. Then we used the `mfcc` function from the library `python_speech_features` in order to compute Mel-frequency cepstral coefficients (MFCC) features from the audio signal. The parameters inserted into the function are summarized in the Table 1. Therefore, each input speech signal of length L (one second in our case) given as input is framed into overlapping frames of length l with a stride s, giving a total of

$$T = \frac{L - l}{s} + 1, \tag{1}$$

frames. From each frame, F speech features are extracted, generating a total of TxF features for the entire input speech signal of length L. We used Mel-frequency cepstral coefficients (MFCC) because they are the commonly used human-engineered speech features in deep learning based speech-recognition.
The dataset has been split in train, validation and test sets, with relative proportions 80:10:10. Each sample belongs to a specific set accordingly to the provided lists for the validation and test samples. These lists were constructed in order to ensure to have different speakers in each set and ensure in this way more heterogeneity.

V. Learning Framework

In this section we go into more technical details about our learning strategy. The extracted speech feature matrix is fed into a classifier module that has one output class for each of the keywords it should detect. It furthermore has an output class for unknown speech signals that are those audio signals with '10' as label.
In this paper we implemented two innovative and less common neural networks: a depthwise separable convolutional neural network (DS-CNN) and a temporal convolutional neural network (TCN).

A. DS-CNN

Recently, depthwise separable convolution has been proposed as an efficient alternative to the standard 3-D convolution operation and has been used to achieve compact network architectures in the area of computer vision. While standard convolution performs the channelwise and spatial-wise computation in one step, depthwise separable convolution splits the computation into two steps:

- depthwise convolution that applies a single convolutional filter per each input channel;
- pointwise convolution is used to create a linear combination of the output of the depthwise convolution.

The comparison of standard convolution and depthwise separable convolution is shown in the Figure 2. Therefore, DS-CNN first convolves each channel in the input feature map (TxF) with a separate 2-D filter and then uses pointwise convolutions (i.e. 1x1xNum_channels) to combine the outputs in the depth dimension. By decomposing the standard 3-D convolutions into 2-D convolutions followed by 1-D convolutions, depthwise separable convolutions are more efficient both in number of parameters and operations, which makes

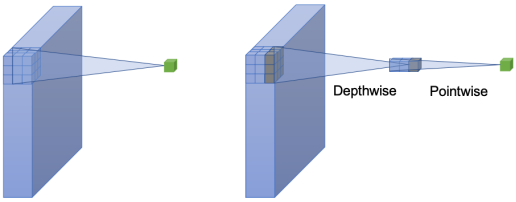

Fig. 2: Standard convolution and depthwise separable convolution

deeper and wider architecture possible even in the resource-constrained microcontroller devices.

a) Architecture:

The architecture of the DS-CNN implemented is shown in the Figure 3. At first, the input feature map is passed onto the first convolutional standard layer, followed by batch normalization and ReLU activations. Batch normalization has been employed to accelerate training and to stabilize the learning process. This has the effect of dramatically reducing the number of training epochs, 35 in our case, required to train deep networks. Then, 4 depthwise separable convolutions follow. Each of them consists of a depthwise convolution and pointwise convolution and followed by a batch-normalization layer with ReLU activation. Each convolutional layer of the network applies a number of filters to detect local time-frequency patterns across input channels. At the end, an average pooling layer reduces the number of activations by applying an averaging window to the entire feature map of each input channel. Finally, after a dropout for reducing overfitting, a fully connected (FC) layer with softmax activations generates the probabilities for each output class.

b) Training:

All networks were trained with TensorFlow machine learning framework using an Adam optimizer to minimize the categorical cross-entropy. The networks were trained in 35 epochs with a batch size of 100. The learning rate of 0.005 was used.

B. TCN

We know that Conv1D can be used for time-series, however standard convolution considers ”future” values in the computation and this fact is unconvenient for several applications, e.g. sequential sampling. This problem is therefore solved by providing a causal formulation to the convolution in which the present value only depends on past and present input values. Causality is easily obtained by padding asymmetrically. Therefore, for a convolutional kernel of size K, a padding of K-1 can be added in the ”past direction”. In this way, the output at position t can be dependent on input values up to K-1 steps in the past, the so called receptive field. To avoid a large number of weights to deal with long range dependencies, the convolutional filter can be dilated by a dilation factor d. Therefore, the output at position t can be dependent on input values up to d(K-1) steps in the past. A final strategy applied in order to avoid extreme sparsity, consists in stack convolutions with different dilation factor. Several convolutional layers are

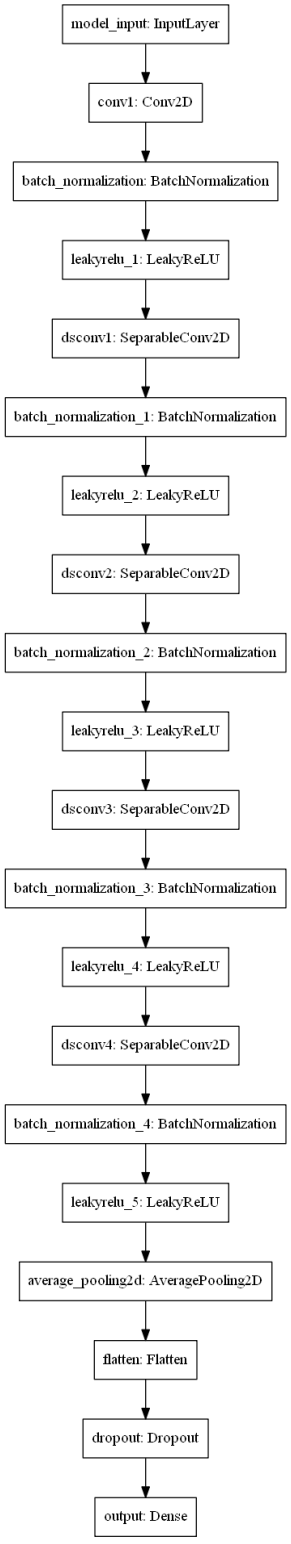

Fig. 3: DS-CNN model

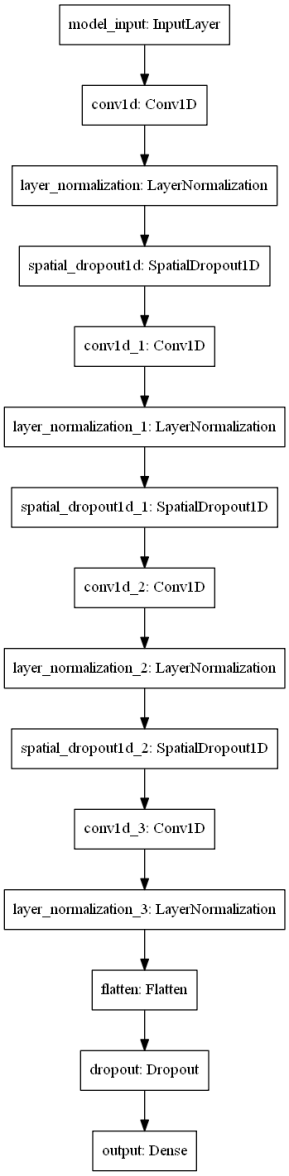

Fig. 4: TCN model

stacked to form dilated causal convolutional networks aka temporal convolutional neural networks (TCN). Considering a dilation factor equals to the 2^l where l is the depth of the layer, the receptive field R of a TCN with l layers and convolutional kernels of size K is calculated as

$$R = 2^l(K-1) \tag{2}$$

The choice to implement a TCN network against a recurrent neural network (RNN) is mainly attributed to some practical and theoretical reasons [8]:

1) **Parallelism.** Unlike RNNs, TCN does need to wait for previous predictions to finish for the current prediction to be declared. Therefore, parallelism can be utilized in TCN to accelerate the inference.
2) **Flexible receptive field size.** TCN provides many ways to capture such variations by controlling the receptive field (RF). For instance, using larger 1D filters, stacking more TCN layers or using bigger dilation factor are all factors that can be tuned for setting a proper receptive field.
3) **Stable gradients.** TCN has a backpropagation path that is different from RNN. Accordingly, TCN avoids problems like vanishing gradients which RNN is famous for.

a) Architecture:

The architecture of the TCN is shown in the Figure 4. The network is designed by training a 1D fully-convolutional network (FCN) but using causal convolutions. The FCN is performed with zero padding to maintain the length of subsequent layers. We stacked multiple TCN layers with kernel of size K. Stride is set to one and dilation rate to $d(j) = d^{j-1}$, where j is the index of the layer. ReLU activation function is applied at each step. Then, the activations of the previous layer for each given example are normalized in a batch independently, rather than across a batch (as for Batch Normalization). A spatial 1D Dropout is applied. This version drops entire 1D feature maps instead of individual elements. It promotes independence between feature maps and should be used if adjacent frames within feature maps are strongly correlated, as happens in early convolution layers.

The dilated convolution in the last layer is then flattened in order to go into the last fully connected layer. Then, a regular dropout is applied in order to randomly sets input units to 0 with a probability of 0.05. The output activation function has been set to Softmax.

b) Training:

All networks were trained with TensorFlow machine learning framework using an Adam optimizer to minimize the cross-entropy loss. The networks were trained in 35 epochs with a batch size of 100. The learning rate of 0.0005 was used.

VI. Results

After having implemented these two networks, the performance has been evaluated in terms of numerical results. In particular, since the KWS is a task especially implemented in devices with limited resources, the focus was on the impact of different networks sizes on the performance. Therefore, we investigated how different numbers of weights would impact the performance in terms of accuracy. In order to obtain scaled versions of the same architecture, we changed some hyperparameters both in DS-CNNs and TCNs. The table 2 summarizes the hyperparameters, the number of weights and the accuracies for each network.

As the first thing, we can underline that also the networks with a relatively small number of parameters are capable to reach good accuracies around 0.850 for DS-CNNs and 0.878 for TCNs. In general, an increasing number of parameters results in performance improves as expected. However, it could be interesting to observe that for DS-CNNs from 17.1K parameters to 69.9K, and for TCNs from 19.9K to 28.8K the accuracy improvement is negligible if compared with the

Type	TxF	Filters	Size	Acc
DS-CNN-1	49x10	16	5.8K	0.850
DS-CNN-2	49x13	16	7.5K	0.865
DS-CNN-3	49x13	32	17.1K	0.887
DS-CNN-4	99x13	64	69.9K	0.888
DS-CNN-5	99x13	170	257.7K	0.899
TCN-1	49x10	16/16/8/8	7.5K	0.878
TCN-2	49x13	32/32/16/16	19.9K	0.919
TCN-3	99x13	32/32/16/16	28.8K	0.921
TCN-4	99x13	64/64/32/32/16/16	79.5K	0.939
TCN-5	99x13	128/128/64/64/32/32	270.1K	0.946

TABLE 2: Hyperparameters, size and accuracy

Type	Size	Accuracy
DS-CNN-3	17.1K	0.887
TCN-2	19.9K	0.919
DNN	288K	0.843
CNN-1	556K	0.907
CNN-2	149K	0.846
LSTM	26K	0.888
CRNN	298K	0.878

TABLE 3: Networks comparison

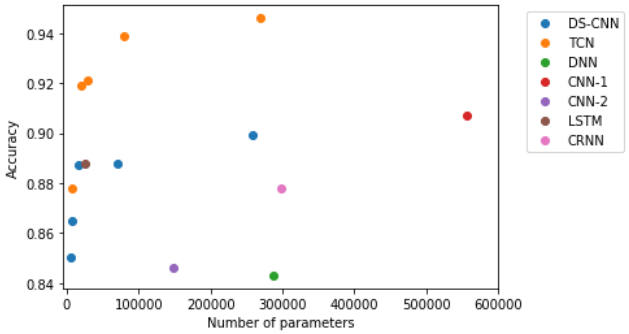

Fig. 5: Network comparison: size vs accuracy.

increased number of parameters. Thus, designing an oversized network is useless since both of our architectures seems to saturate the accuracy up to a certain number of parameters (logarithmic trend).

At this point, we searched in the literature some other different implemented networks with corresponding numbers of parameters and accuracies. We selected a DNN from [9], two CNNs from [4], a LSTM from [10] and a CRNN from [11]. In the Table 3 the comparison between the different networks is reported. Looking at the values, we have a confirmation about the better performance reached by our two architectures with smaller size. This consideration is evident also looking at the Figure 5 where each color for the points corresponds to a network and its coordinates are given by the number of parameters and the accuracy.

After this analysis, we selected the best performing network both between the DS-CNNs and the TCNs making a trade-off between accuracy and size. We identified DSNN-3 and TCN-2 as the best networks and for these two networks we plot accuracies (see Figure 6 and 7, respectively) and loss for the different epochs (see Figure 8 and 9, respectively).

At first, the loss of the validation set is lower than the training loss due to the dropout factor. In the last epochs instead, the ability of the network to extract features does not depends on

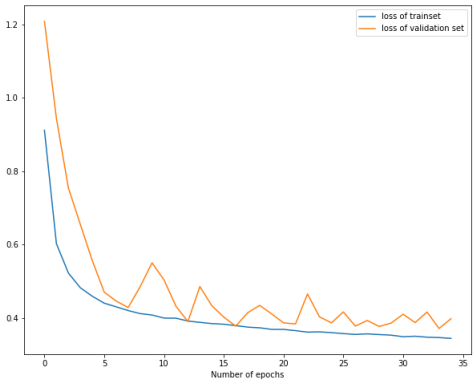

Fig. 6: Accuracy for training and validation set for the DS-CNN.

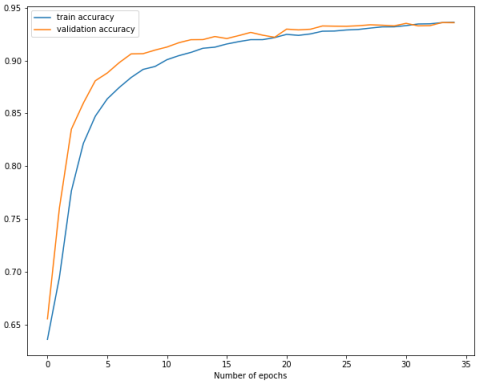

Fig. 7: Accuracy for training and validation set for the TCN.

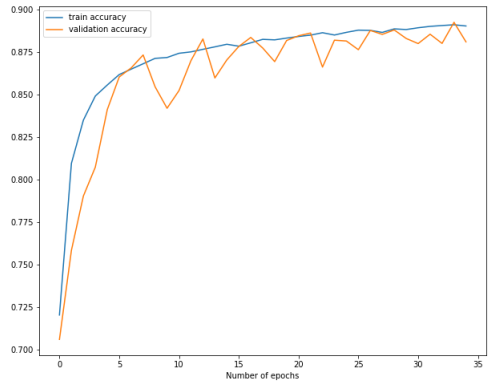

Fig. 8: Loss for training and validation set for DS-CNN.

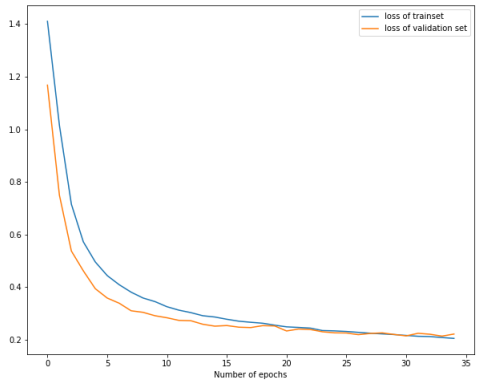

Fig. 9: Loss for training and validation set for TCN.

single nodes and the losses converge.
To have a different visualization of the performance of the models, we constructed the confusion matrix for the DS-CNN and TCN. These are visible in the Figure 10 and Figure 11, respectively.

The matrices are close to the identity matrix, which means that the models are able to recognize the keywords in most of the cases, also when they sound similar. While false positive are rare, false negative have higher probability to occur. In a scenario where energy management is critical, false negative are preferable than false positive (useless activation).

VII. Concluding Remarks

In this paper, methods for training and implementing a DS-CNN-based and a TCN-based KWS systems were presented and evaluated. The scalability of both the architecture is also evaluated. While it is possible to design bigger and bigger networks, the accuracy does not grow linearly with size. In a scenario where mobile devices are playing a main role in the digitalization process, a solution that takes into account their constrained resources is demanded. DS-CNN-3 and TCN-2 show remarkable results in terms of accuracy,

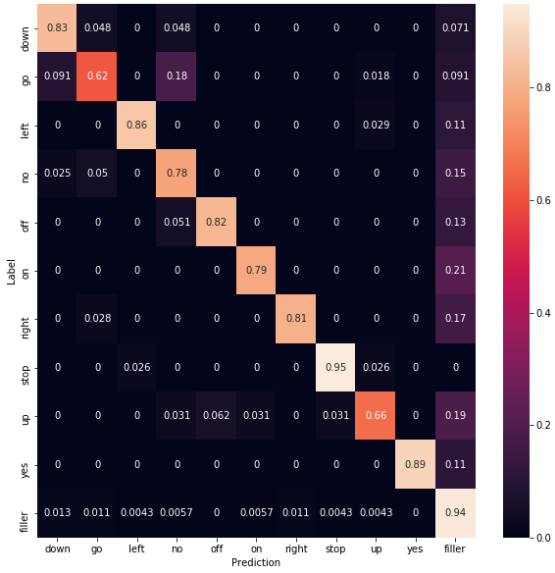

Fig. 10: Confusion matrix for the DS-CNN.

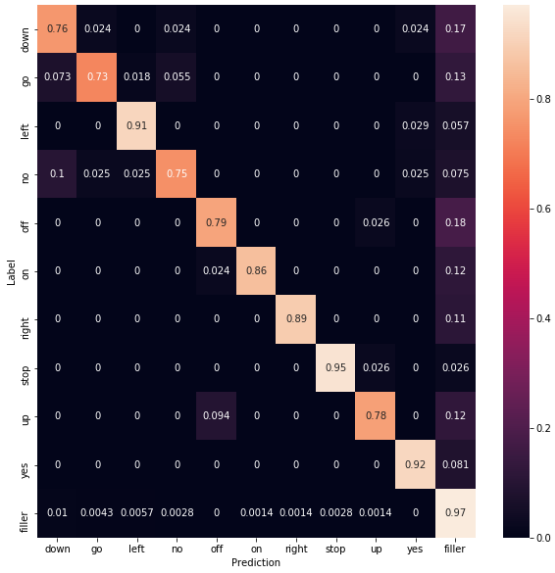

Fig. 11: Confusion matrix for the TCN.

keeping the number of parameters below 20k. Moreover, the TCN architecture can be implemented exploiting its parallel inference nature. Further analysis are required in terms of number of operations and hyperparameters tuning. Also, new architecture are continuously explored for KWS task, such as the Shared-Weight Self-Attention Network described by Ye Bau et al. in [12]. This network overcomes the high number of parameters required for self attention mechanism using a single weight matrix. This solution shows promising results using just 12k parameters. Finally, different audio features can be explored in place of MFCC.

References

[1] P. M. Sorensen, B. Epp, and T. May, "A depthwise separable convolutional neural network for keyword spotting on an embedded

system," *EURASIP Journal on Audio, Speech, andMusic Processing*, vol. 2020:10, 2020.

[2] M. Elbayad, L. Besacier, and J. Verbeek, "Pervasive Attention: 2D Convolutional Neural Networks for Sequence-to-Sequence Prediction," *CoRR*, vol. abs/1808.0, 2018.

[3] C. C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, and E. Gonina, "State-of-the-Art Speech Recognition with Sequence-to-Sequence Models," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

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

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

[6] E. A. Ibrahim, J. Huisken, H. Fatemi, and J. Pineda de Gyvez, "Keyword spotting using time-domain features in a temporal convolutional network," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*, pp. 313–319, 2019.

[7] P. Warden, "Speech commands: a public dataset for single-word speech recognition," *Dataset available from http://download.tensorflow.org/data/speech˙commands˙v0.02.tar.gz*, 2017.

[8] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *CoRR*, vol. abs/1803.01271, 2018.

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

[10] 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," *CoRR*, vol. abs/1705.02411, 2017.

[11] S. Ö. Arik, M. Kliegl, R. Child, J. Hestness, A. Gibiansky, C. Fougner, R. Prenger, and A. Coates, "Convolutional recurrent neural networks for small-footprint keyword spotting," *CoRR*, vol. abs/1703.05390, 2017.

[12] Y. Bai, J. Yi, J. Tao, Z. Wen, Z. Tian, C. Zhao, and C. Fan, "A time delay neural network with shared weight self-attention for small-footprint keyword spotting," pp. 2190–2194, 09 2019.