Beating human performance at recognizing speech commands in temporal domain

Iván Vallés-Pérez[a], Fernando Mateo[a], Joan Vila-Francés[a], Antonio J. Serrano-López[a], Emilio Soria-Olivas[a]

[a]*Escola Tècnica Superior d'Enginyeria, University of Valencia, Avenida de la Universitat s/n 46100 Burjassot, Valencia, Spain*

Abstract

Speech commands recognition is a very important task nowadays due to the increase of the global interest in personal assistants. The field of conversational agents is growing fast and there is an increasing need for algorithms that perform well in order to enhance natural interaction. In this work we show how we achieved state of the art results by adapting and tweaking the *Xception* algorithm, published by François Chollet in 2017, which achieved outstanding results in several computer vision tasks. We obtained about 96% accuracy when classifying audio clips belonging to 35 different categories, beating human performance at the most complex benchmarks proposed.

Keywords: Speech Recognition, Deep Learning, Convolutional Neural Network, Representation Learning

1. Introduction

The world of voice-activated virtual assistants is booming. Several giant technological companies, (such as Amazon, Google, Apple and Baidu) have already developed their version of a virtual assistant. There is a huge research community surrounding this field, potentially promoted by the recent growth of the artificial intelligence (AI) paradigm.

There has been an outstanding evolution in this field, leveraging AI and machine learning (ML) advances for making virtual assistants behave as close as humans as possible. There are multiple open research lines, such as increasing the accuracy and the relevance of the responses [1], reducing the answer delay [2] or increasing their variability of the responses [3]. In particular, Deep Learning (DL) models have revolutionized the field of automatic speech recognition [4], as language features are highly hierarchical. This work focuses on DL models to increase the accuracy of a voice command recognition task using a limited vocabulary.

An important factor to take into account is the small size and low power specifications of the usual virtual assistants, which limit the complexity of the models to implement. Furthermore, a low latency is needed to avoid harming the users' experience. DL models usually contain millions of parameters. For that reason, the right choice of number of layers and their architecture is critical. Using the cloud for processing audio commands can partially mitigate this. However, continually transmitting to and from the cloud increases latency and may not be an efficient solution. Therefore, at least the models that process the most common words in a vocabulary, such as keyword spotting (KWS), should be implemented locally.

The current commercial virtual assistants are still not as accurate as humans at identifying human voice commands [5]. Although substantial efforts have been made and nearly human performance has been reported in several studies [6, 7, 8, 9], there is still room for improvement. In particular, bidirectional recurrent models with attention have been used [6] as these kind of structures are able to accurately model past and future dependencies in the time domain, while attention focuses on important parts of the audio clip. Howewer, the authors state that there are word pairs that are difficult to identify without extra context. Gated convolutional long-short term memory (LSTM) structures have also proven useful by other authors [10] to improve the state of the art results on

the same data. According to this work, gated convolutions help further learn the local features of speech, improving the model prediction accuracy.

Due to their architecture, convolutional neural networks (CNNs) provide a good approach to optimize computational resources for KWS. In [11], CNNs outperform other deep neural networks (DNNs) architectures, such as recurrent reural networks (RNN), for the constrained KWS task. Other works focus on the hardware implementation of neural networks for KWS [7], comparing not only their accuracy, but also their memory usage and computation efficiency. According to this work, Deepwise Separable (DS) CNNs achieve both the best accuracy and scalability among the tested architectures. Transfer learning has also been applied to CNN architectures [8], showing substantial improvements in accuracy.

The present work provides the insights of the study and implementation of an *Xception*[1] based architecture [12], which we named *Xception-1d*, to the speech commands recognition problem. The contributions of the present work are summarized in the following bullets.

- Design and implementation of a *depthwise separable* CNN-based architecture able to surpass the current state of the art results and the human performance while keeping the data in the temporal domain (using the raw version of the waveform).
- Development of a new methodology (to the best of our knowledge) for augmenting audio data to increase the size of a data set (5x in this work), and therefore enhance generalization.
- Quantification of the human performance across the different classification tasks to use it as an additional baseline for checking the results achieved by the algorithm.
- Creating a public repository where further contributions could be handled to enhance the project functionalities and to facilitate reproducibility.

The article is structured as follows. Right after this introduction, section 2 introduces the data that has been used to define the tasks to be solved and the methodology and algorithms that have been used in order to achieve the results presented in section 3. Section 4 provides a discussion of the results and section 5 summarizes the insights, takeaways and possible future work lines.

2. Material and methods

2.1. Data set

One of the major contributions to the collection of open data sets in the field of speech commands recognition has been made by *Google* with the release of the *Google Tensorflow speech commands data set* [13, 9]. It consists of a collection of thousands of utterances with a length of one second containing short words, recorded by thousands of people. The recordings were collected remotely, in uncontrolled environments (i.e. without any specific quality requirements). The subjects were asked to say a chain of words during a minute. Then, the recordings were split in clips of one second by extracting the loudest sections of the signal [13, 9].

Two different versions of the *Google Tensorflow speech commands data set* have been used for quantifying the performance of the algorithm across the different tasks, described subsequently. The versions 0.01 and 0.02 of the data set contain 64,721 and 105,829 audio clips of one second (each one containing the recording of one voice command), respectively, with a sample rate of 16 kHz and 16-bit resolution. Each of them is stored in *wav* format. The first data version has up to 30 different commands while the second one has 35 of them. The frequency distribution of the labels is described in Figure 1. For simplicity, we will refer to versions 0.01 and 0.02 as V1 and V2, respectively, from now on.

Four different tasks have been defined in order to benchmark the proposed algorithm. They are thoroughly described below in decreasing complexity order.

[1] *Xception* is a deep CNN architecture that involves Depthwise Separable Convolutions.

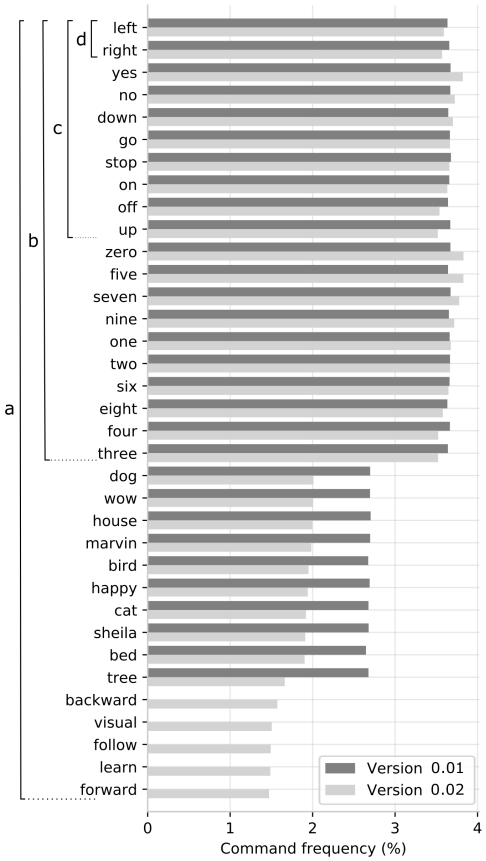

Figure 1: Command frequency distribution for both versions of the data set. The V2 is a refined and extended version of V1. In the left, the four different tasks that have been benchmarked in this work: (a) referred as *35-words-recognition* and comprising in both cases all the words for classification, (b) referred as *20-commands-recognition* (c) referred as *10-commands-recognition* (d) referred as *left-right* recognition.

- *35-words-recognition*: consists of classifying all the different existing clips (commands and plain words) in each of the different categories (“left”, “right”, “yes”, “no”, “down”, “up”, “go”, “stop”, “on”, “off”, “zero”, “one”, “two”, “three”, “four”, “five”, “six”, “seven”, “eight”, “nine”, “dog”, “cat”, “wow”, “house”, “bird”, “happy”, “sheila”, “marvin”, “bed”, “tree”, “visual”, “follow”, “learn”, “forward”, “backward”). In the version V1 of the data set there are 5 missing commands and hence, the task consists of a 30-class classification task even though it is called *35-words-recognition*.

- *20-commands-recognition*: entails the categorization of all the clips representing the most commonly used commands in robotics [9] and numbers (“left”, “right”, “yes”, “no”, “down”, “up”, “go”, “stop”, “on”, “off”, “zero”, “one”, “two”, “three”, “four”, “five”, “six”, “seven”, “eight”, “nine”), while the remaining words are grouped together in a synthetic category named “unknown”.

- *10-commands-recognition*: requires categorizing all the clips representing the typical commands in robotics (“left”, “right”, “yes”, “no”, “down”, “up”, “go”, “stop”, “on”, “off”), while the remaining clips are grouped together in a synthetic category named “unknown”.

- *left-right-recognition*: consists of categorizing the clips belonging to the “left” and “right” categories, while the rest of clips are grouped under the “unknown” category.

While it is true that, as we group the unrecognized words under the “unknown” category, the

imbalance grows, in this kind of systems (speech recognition agents), the precision should be optimized at the expense of a worse recall (in general, the cost of a false positive is higher than that of a false negative). Thus, for our purpose, having a positive imbalance towards the "unknown" class does not represent an inconvenience. A summary of the percentage of words grouped under the "unknown" category for each task is shown in Table 1.

Table 1: Percentage of words represented by the "unknown" category in each one of the proposed speech recognition tasks.

Data set version	*35-words*	*20-commands*	*10-commands*	*left-right*
V1	0.00%	26.84%	63.41%	92.71%
V2	0.00%	26.81%	63.58%	92.84%

2.2. Data augmentation

Five different augmentations[2] consisting of the application of a set of distortions have been performed to each and every audio clip. These distortions consist of a set of transformations being applied together over all the clips randomizing their parameters and intensities in each case. The different distortions used in this step are described below.

- **Resampling**: the audio clip is resampled extending or contracting its length and hence changing its pitch [14]. If the resampling factor is lower than one, the audio clip is contracted and then the audio duration is zero-padded to keep the original duration of the original clips. On the contrary, if the resampling factor is greater than one, the audio clip is extended and a *center-crop* operation is performed in order to keep the original length of one second.
- **Saturation**: the amplitude of the clip is increased by a given factor, potentially saturating the audio clip. The higher the factor, the larger the saturation of the clip [14].
- **Time offset**: the audio clip is displaced in time by appending a set of zeros to the beginning of the signal and cropping the end (right offset) or by cropping a set of samples from the beginning, and adding the same number of zeros at the end (left offset) [14].
- **White noise addition**: white noise (with gaussian distribution) is added to the clips with a given amplitude [14].
- **Pitch shift**: the pitch of the clips is increased or decreased a given amount, producing higher or lower-pitched sounds [14, 15].

All the distortions are applied together with random intensities only to the training data, producing 5 new transformations of the original recordings with high variability of results. These new versions are appended to the original data set.

2.3. Methods

We propose the use of a CNN with *depthwise separable convolution* layers and *residual connections*, based on the *Xception architecture* described by *François Chollet* in 2017 [12]. This model is a CNN-based architecture which recently achieved state of the art results in multiple computer vision tasks [16, 17, 18, 19].

[2]meaning distorted versions of each of the clips in the original data

2.3.1. Depthwise separable convolutions

The *regular convolution* operation consists of the application of a filter over a signal along the spatial/temporal dimension(s) and along the channels in a single operation.

A *depthwise separable convolution* performs an operation which, given an input tensor, produces an output tensor of the same shape that the regular convolution would produce, but in a more efficient way; i.e. it reduces the number of sums and multiplications needed to produce the output [12]. This is achieved by following the two steps described below. The whole operation is shown in the Figure 2.

1. *Depthwise convolution* [20]: consists of convolving a separate filter per channel. It produces an output tensor with potentially a different spatial/temporal dimensions size than the input (depending on the stride, the size and the padding of the convolution operation to be applied). However, the number of channels of the output tensor is constrained to be the same as the input.
2. *Pointwise convolution* [21]: consists of applying a set of size-1 convolutions (as many as number of output channels desired). This operation can modify the channels dimension, leaving the spatial/temporal dimensions intact.

Please refer to the appendix 1 to see the proper mathematical definitions.

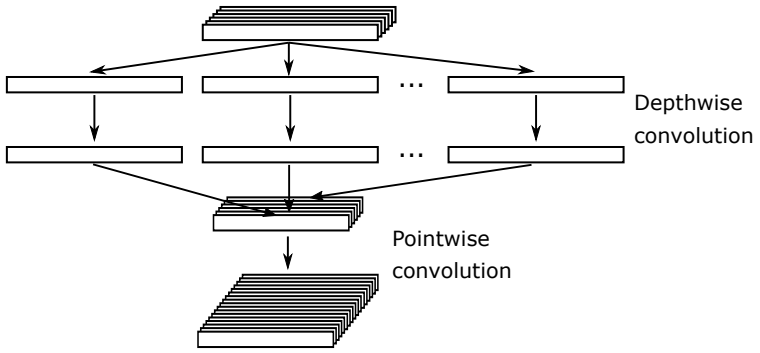

Figure 2: *Depthwise separable convolution* diagram showing how the whole computation is done in two steps: the *depthwise convolution* followed by the *pointwise convolution*.

The *depthwise convolution* does not combine different channels for producing the output, and hence always produces an output tensor which has the same number of channels as the input tensor. This difference makes this operation much more efficient than the *regular convolution*, at the cost of loosing the ability to generate new features by combining different channels. That is the reason why the *pointwise convolution* is applied after the *depthwise convolution* constituting the *depthwise separable convolution*. The reason why the *separable* term is used in the name of the operation is because we are effectively separating between the channel-wise and the spatial/temporal-wise computations. The number of sums and multiplications required by this operation is $\frac{1}{N} \cdot \frac{1}{S}$ times the number of operations required by a regular convolution [22], where S is the size of the *depthwise convolution* filters and N is the number of output channels after the *pointwise convolution* is applied. This represents a meaningful performance improvement for big networks.

2.3.2. Xception-1d architecture

The proposed architecture exploits the gain in efficiency of the *depthwise separable convolution* operation over the *regular convolution* in the same way original *Xception* does [12]. That allows a more efficient resource and time management and hence, a more complex architecture can be defined.

The *Xception-1d* architecture is shown in Figure 3. It has a total of 37 layers, 34 of which are *depthwise separable convolutional layers*, 2 of them are *regular convolutions*, and the last one is a dense layer performing a logistic regression. The network contains 12 residual connections, one in

each residual block, as shown in Figure 4. All of it builds up a network with up to 21 million parameters in the case of the *left-right recognition* task and 23 million parameters in the *35-words recognition* one[3].

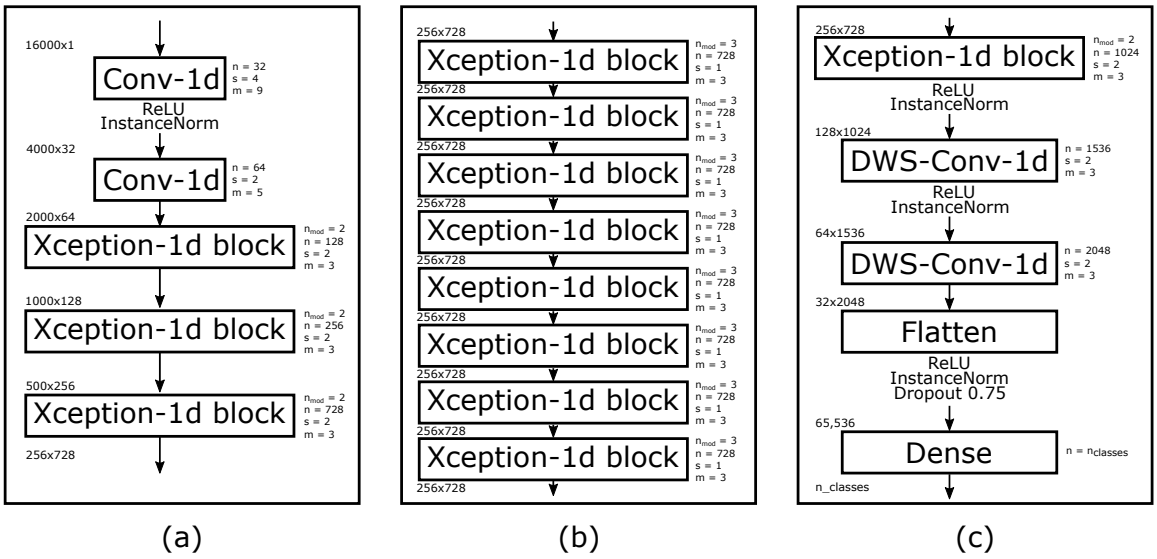

Figure 3: Diagram of the architecture used in this work where n refers to the number of channels, s is the stride of the convolutions (which in the case of the *Xception-1d blocks* is applied through the final pooling operation), m is the size of the convolution filters, n_{mod} is the number of the *depthwise convolutional* layers stacked in the *Xception-1d blocks* (see Figure 4) and $n_{classes}$ is the number of outputs of the network (i.e. the number of classes to predict, which depends on the task being solved). The architecture is built up in 3 main modules: (a) the entry module is the part of the network responsible for adapting the input wave into a condensed representation (by applying strides after each operation), (b) the middle module is responsible for learning the representation extracting the useful features that will allow the next module to distinguish between classes and (c) the classification module is responsible for mapping the extracted features into the number of outputs required for every task.

Considering that the last dense layer can be prone to overfitting due to the high number of parameters ($\sim 65,000$), a strong dropout [23, 24] has been applied after the last convolution ($p = 75\%$). In addition, a small *weight decay* [25, 26, 24] has been applied over all the network weights (specifically $\lambda = 10^{-3}$) in order to enhance regularization. *Adam* optimizer has been used to train the network [27]. The initial learning rate has been fixed to $\eta = 10^{-4}$. It has been decreased by a factor of $\frac{1}{2}$ when no improvement was observed (i.e. when a *plateau* is reached) with a patience of 4 epochs.

Instance normalization has been used to normalize the intermediate outputs of each convolution [28, 29]. It simply consists of standardizing separately each of the channels of every instance. Although it is typically used in generative modelling and style transfer efforts, it demonstrated to be very useful to improve training time of convergence and it has no undesirable effects at test time[4] [28]; it is considered a key element of this architecture. The input of the dense layer has been normalized using *layer normalization* [30]. *Batch normalization* [31] has been avoided because it is prone to introduce *covariance shift*, degrading generalization [30]. Figure 5 summarizes the way these three normalization techniques operate.

[3]The difference lays on the last layer implementing the logistic regression of the architecture. Depending on the number of outputs required by each task, the number of parameters in this layer varies. The maximum difference occurs between the *35-words recognition* task and the *left-right recognition*

[4]In the case of *Batch Normalization*, there can appear big differences of performance between train time and test time [30]

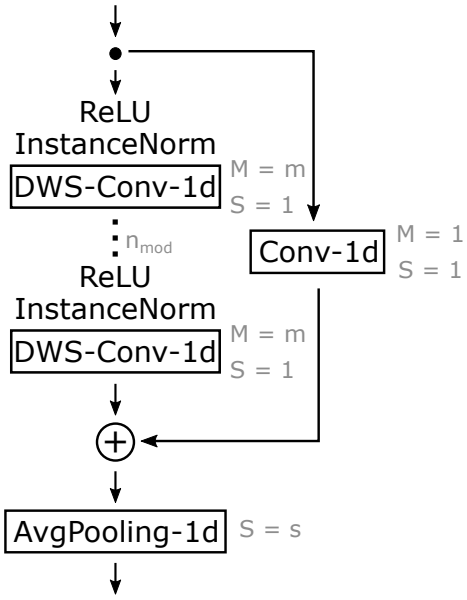

Figure 4: Neural network module refered as *Xception-1d block*. The *Xception-1d block* represents the building block of the architecture proposed in this work. It consists of a set of n_{mod} *1-D depthwise separable convolution* layers with a *residual connection* and with a *1-D average pooling* layer at the end. The activation function used is the *rectified linear unit* (ReLU) and the normalization procedure after each convolutional layer is the *instance normalization* for one dimensional sequences.

3. Results

The train/development/test split provided by the authors of the data set [13] has been used as *cross validation* (CV) setting in order to facilitate future benchmarking efforts. A simple split CV has been used instead of K-fold CV for the following reasons: (1) these models are expensive to train. Forty deep models (2 data set versions $\times$ 4 tasks $\times$ 5 random initializations) were trained per experiment. If we added K-Fold CV it would become unfeasible; (2) for the sake of reproducibility and benchmarking, as the authors of the dataset provide a default list of clips to use as validation and test.

Versions V1 and V2 of the data set have 16,000 and 9,981 hold out samples for development purposes and 16,000 and 11,005 hold out samples for testing purposes, respectively. The development set has been used to manually tune the hyperparameters and for *early stopping* purposes. The model has been trained for 50 epochs in each case, with a batch size of 32 clips, and the weights of the epoch that achieved the best performance in the development set were checkpointed. The checkpointed models have been used to calculate and report the performance of the algorithm.

The source code that has been used for this study has been uploaded to *GitHub* and can be found here: `https://github.com/ivallesp/Xception1d`

All the results shown in this section have been measured over the test set. Five different models have been trained for each task in order to explore and report the effect of different random initializations of the weights of the network. With the aim of providing a baseline, human performance has been measured by 4 human subjects, who manually labeled $\sim$ 1000 commands achieving different results. These results are reported in Table 2 along with the results of the proposed algorithm and the results reported by [6, 7, 8, 9] on the matching tasks.

Besides the global results, Figure 6 shows the precision and the recall obtained for the most complex model (*35-words-recognition* for data version V2). In conjunction with this figure, precision, recall and f1-score metrics have been included in the tables of the appendix 2, at the end of the article.

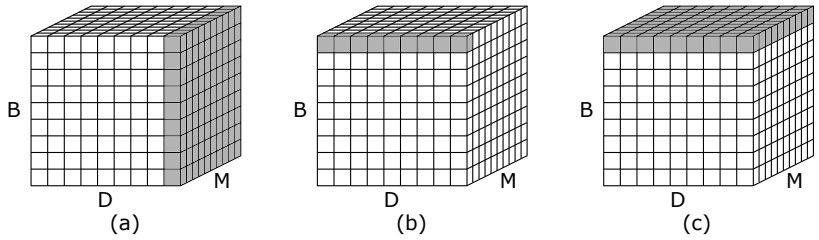

Figure 5: Each cube represents a batch of data where B is the instances axis (with the batch size), M is the channels axis and D is the spatial/temporal dimension. (a) represents *batch normalization*, (b) is *instance normalization* and (c) shows *layer normalization*. The shaded areas in the cubes represent the dimensions that are aggregated in each case for computing the statistics used to normalize the data (generally the mean and the standard deviation). As it can be noticed, the only normalization technique that depends on instances of data in the batch is the *batch normalization*, that is why it is more sensitive to train-test distribution differences. As it can be noticed in the picture, *instance normalization* is a specific version of *layer normalization* where each channel is normalized separately

Table 2: Accuracy (mean ± standard deviation) obtained by the proposed solution on the different tasks compared to other algorithms (described in [6, 8, 9, 7]) and compared to our measurement of human accuracy (through 4 manual evaluations). The results of best performing algorithms for each task have been highlighted in bold in each case. Results better than human performance (given a statistical significance level of $\alpha = 0.05$) have been tagged with a star mark (*).

(a) Results for version 1 of the data set.

	Andrade et al.[6]	McMahan et al. [8][a]	Warden [9]	*Xception-1d*	Human	p-value[b]
35-words	94.30	84.35	-	**95.85 ± 0.12 ***	94.15 ± 1.03	$1.46 \cdot 10^{-2}$
20-commands	94.10	85.52	-	**95.89 ± 0.06 ***	94.56 ± 0.98	$3.14 \cdot 10^{-2}$
10-commands	95.60	-	85.40	**97.15 ± 0.03**	97.22 ± 0.85	$8.75 \cdot 10^{-1}$
left-right	**99.20**	95.32	-	98.96 ± 0.09	99.54 ± 0.16	$5.24 \cdot 10^{-4}$

(b) Results for version 2 of the data set.

	Andrade et al. [6]	Zhang et al. [7][a]	Warden [9]	*Xception-1d*	Human	p-value [b]
35-words	93.90	-	-	**95.85 ± 0.16 ***	94.15 ± 1.03	$1.50 \cdot 10^{-2}$
20-commands	94.50	-	-	**95.96 ± 0.16 ***	94.56 ± 0.98	$2.70 \cdot 10^{-2}$
10-commands	96.90	95.40	88.20	**97.54 ± 0.08**	97.22 ± 0.85	$4.84 \cdot 10^{-1}$
left-right	**99.40**	-	-	99.25 ± 0.07	99.54 ± 0.16	$1.27 \cdot 10^{-2}$

[a] the best results obtained among all the trials performed by the autors have been selected
[b] Student's t-test for the comparison of two means. $\alpha = 0.05$

4. Discussion

The presented method, *Xception-1d*, offers better performance than the existing methods in the literature for three out of the four tested tasks, using different sets of limited-size vocabularies. According to the results in Table 2, *Xception-1d* performed voice recognition better than the state of the art methods [6, 7, 8, 9] in three out of four tasks: *35-words-recognition*, *20-commands-recognition*, and *10-commands-recognition*. In the only task where *Xception-1d* did not achieve the best results (left-right), the leading method was the one proposed by Andrade *et al.* [6] which was only marginally better (<0.5% performance difference on the test set) than the presented method. *Xception-1d* even surpassed human performance (with statistical significance level) in the two first tasks, including the most difficult one (*35-words-recognition*).

With regard to per-class accuracy, it can be noticed from Figure 6 that the algorithm performs generally well for all the classes in the most difficult scenario (35-words task) as the majority of precision and recall values lay between 90-100%. Nonetheless, we found that the algorithm has more difficulties differentiating some groups of similar words like the following pairs: "three" and "tree", "follow" and "four", "bed" and "bird", etc. No comparison with other existing models has been included because no such detailed results have been found in the related work.

The limitation of this method is mainly linked to inherent computational cost. Depthwise separable convolutions allow for a more resource-efficient network architecture than regular convolutions, but that means that more complex architectures can be built. The tested network architectures

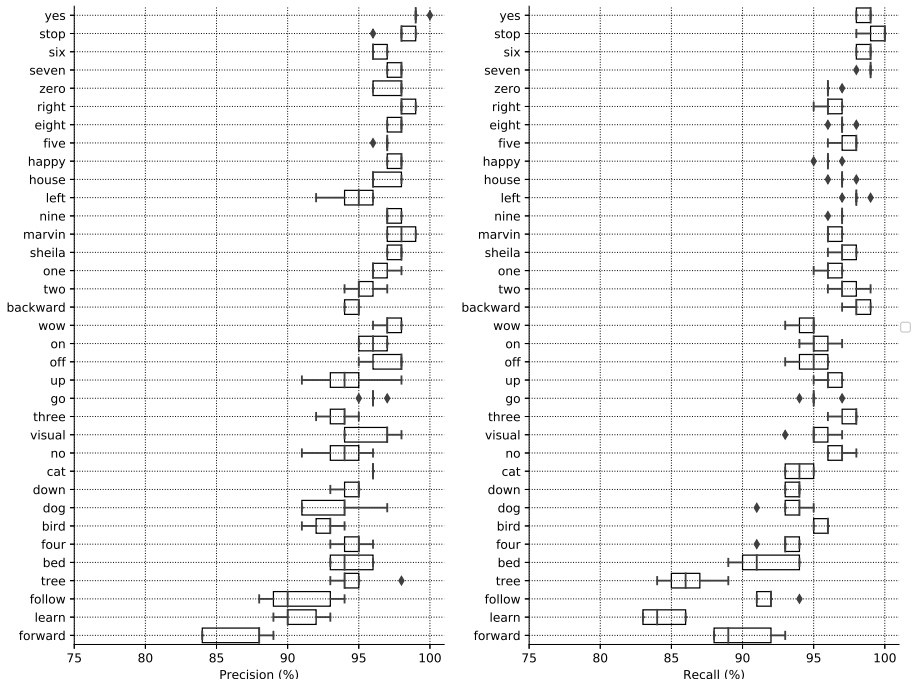

Figure 6: Precision and recall for each of the different classes using the *35-words-recognition* model trained with data version V2. Classes are sorted by descending f1-score.

have more than 20 million parameters, which makes training computationally expensive, even with datasets composed by a few thousands instances. Overfitting may be an issue due to this amount of parameters, but it has been prevented by using regularization with a small weight decay in all layers, and a strong dropout after the last dense layer. In spite of the aforementioned limitation, once the model has been trained with a particular vocabulary corpus, it may be locally implemented in voice-controlled systems that work in real time such as commercial speech agents.

Finally, our findings show that the per-class performance of the speech commands in the *left-right* task is lower than for the other tasks (in particular the recall values). However, the per-class performance for these two classes was higher when they were included in a multiclass classification task like the *35-words-recognition* task. This fact suggests that auxiliary tasks (in this case the 35-words vocabulary task) can benefit primary tasks (left-right), as more features may be extracted from more complex tasks. This hypothesis has been proven for Reinforcement Learning [32], but may also apply to DL efforts.

5. Conclusion

This work shows how a neural network architecture which succeeded in the computer vision field, with an adaption and a set of tweaks, is able to surpass human performance at a speech recognition task with limited vocabulary achieving state of the art results. This is why we suggest *Xception-1d* as the *de facto* architecture when facing a voice command recognition task with restricted vocabulary, considering the computing power is not a limiting factor. The algorithm presented can have multiple applications for improving voice-controlled systems.

A possible future line of work could be the use of *Xception-1d* architecture with a global pooling layer at the end as an encoder of a *sequence-to-sequence* architecture for tackling a speech recognition task with free vocabulary (e.g. a *speech to text* engine). In addition, pruning and complexity reduction techniques may be implemented over the final network to try to reduce its computational cost.

References

[1] I. Serban, C. Sankar, M. Germain, S. Zhang, Z. Lin, S. Subramanian, T. Kim, M. Pieper, S. Chandar, N. Rosemary Ke, S. Mudumba, A. de Brebisson, J. M. R. Sotelo, D. Suhubdy, V. Michalski, A. Nguyen, J. Pineau, Y. Bengio, A deep reinforcement learning chatbot, in: Proceedings of the Neural Information Processing Systems Conference, 2017.

[2] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, H. Yang, W. B. J. Dally, ESE: Efficient speech recognition engine with sparse LSTM on FPGA, in: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA, New York, NY, USA, 2017, pp. 75–84. doi:10.1145/3020078.3021745.

[3] J. Li, W. Monroe, T. Shi, A. Ritter, D. Jurafsky, Adversarial learning for neural dialogue generation, in: Proceedings of the conference on Empirical Methods in Natural Language Processing, 2017, p. 2157–2169.

[4] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, K. Shaalan, Speech recognition using deep neural networks: A systematic review, IEEE Access 7 (2019) 19143–19165.

[5] A. H. Michaely, X. Zhang, G. Simko, C. Parada, P. Aleksic, Keyword spotting for google assistant using contextual speech recognition, in: 2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), IEEE, 2017, pp. 272–278.

[6] D. Coimbra de Andrade, S. Leo, M. Loesener Da Silva Viana, C. Bernkopf, A neural attention model for speech command recognition, Computing Research Repository CoRR (August 2018). arXiv:1808.08929.

[7] Y. Zhang, N. Suda, L. Lai, V. Chandra, Hello edge: Keyword spotting on microcontrollers, Computing Research Repository CoRR abs/1711.07128 (November 2017). arXiv:1711.07128.

[8] B. McMahan, D. Rao, Listening to the world improves speech command recognition, Computing Research Repository CoRR abs/1710.08377 (October 2017). arXiv:1710.08377.

[9] P. Warden, Speech commands: A dataset for limited-vocabulary speech recognition, Computing Research Repository CoRR abs/1804.03209 (April 2018). arXiv:1804.03209.

[10] D. Wang, S. Lv, X. Wang, X. Lin, Gated convolutional LSTM for speech commands recognition, in: Proceedings of the International Conference of Computer Science, 2018, pp. 669–681.

[11] T. N. Sainath, C. Parada, Convolutional neural networks for small-footprint keyword spotting, in: Proceedings of the annual conference of the International Speech Communication Association, INTERSPEECH, ISCA, 2015, pp. 1478–1482.

[12] F. Chollet, Xception: Deep learning with depthwise separable convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 1800–1807. doi:10.1109/CVPR.2017.195.

[13] P. Warden, Speech commands: A public dataset for single-word speech recognition., Datasets available from http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz and http://download.tensorflow.org/data/speech_commands_v0.02.tar.gz (August 2017).

[14] J. G. Proakis, D. G. Manolakis, Digital Signal Processing (4rd Ed.): Principles, Algorithms, and Applications, 4th Edition, Prentice-Hall, Inc., Upper Saddle River, NJ, USA (2007).

[15] S. Buś, K. Jedrzejewski, Digital signal processing techniques for pitch shifting and time scaling of audio signals, in: Proceedings of SPIE - The International Society for Optical Engineering, Vol. 10031, 2016, pp. 1003157–1. doi:10.1117/12.2249374.

[16] C. Liu, L. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, L. Fei-Fei, Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation, Computing Research Repository CoRR abs/1901.02985 (January 2019). arXiv:1901.02985.

[17] T. S. Nazaré, R. F. de Mello, M. A. Ponti, Are pre-trained CNNs good feature extractors for anomaly detection in surveillance videos?, Computing Research Repository CoRR abs/1811.08495 (November 2018). arXiv:1811.08495.

[18] L. Song, J. Liu, B. Qian, M. Sun, K. Yang, M. Sun, S. Abbas, A deep multi-modal CNN for multi-instance multi-label image classification, IEEE Transactions on Image Processing 27 (12) (2018) 6025–6038. doi:10.1109/TIP.2018.2864920.

[19] O. Arriaga, M. Valdenegro-Toro, P. Plöger, Real-time convolutional neural networks for emotion and gender classification, Computing Research Repository CoRR abs/1710.07557 (October 2017). arXiv:1710.07557.

[20] Y. Guo, Y. Li, R. Feris, L. Wang, T. Rosing, Depthwise Convolution is All You Need for Learning Multiple Visual Domains, Association for the Advancement of Artificial Intelligence (February 2019).

[21] H. Gao, Z. Wang, S. Ji, Channelnets: Compact and efficient convolutional neural networks via channel-wise convolutions, in: Proceedings of Neural Information Processing Systems, 2018.

[22] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications., Computing Research Repository CoRR abs/1704.04861 (April 2017). arXiv:1704.04861.

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, Journal of machine learning research 15 (1) (2014) 1929–1958.

[24] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, http://www.deeplearningbook.org.

[25] A. Krogh, J. A. Hertz, A simple weight decay can improve generalization, in: Proceedings of the 4th International Conference on Neural Information Processing Systems, NIPS'91, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1991, pp. 950–957.

[26] S. Haykin, Neural Networks: A Comprehensive Foundation, 2nd Edition, Prentice Hall PTR, Upper Saddle River, NJ, USA (1998).

[27] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization., in: 3rd International Conference of Learning Representations (ICLR), 2014.

[28] D. Ulyanov, A. Vedaldi, V. S. Lempitsky, Instance normalization: The missing ingredient for fast stylization, Computing Research Repository CoRR abs/1607.08022 (July 2016). arXiv:1607.08022.

[29] Z. Xu, X. Yang, X. Li, X. Sun, The effectiveness of instance normalization: a strong baseline for single image dehazing, Computing Research Repository CoRR abs/1805.03305 (May 2018). arXiv:1805.03305.

[30] J. Ba, R. Kiros, G. E. Hinton, Layer normalization, Computing Research Repository CoRR abs/1607.06450 (July 2016). arXiv:1607.06450.

[31] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: 32nd International Conference on Machine Learning - Volume 37, International Conference of Machine Learning, JMLR.org, 2015, pp. 448–456.

[32] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, K. Kavukcuoglu, Reinforcement learning with unsupervised auxiliary tasks, arXiv preprint arXiv:1611.05397 (2016).

Appendix 1

The *regular convolution* and *depthwise convolution* operations [22] are defined[5] in equations 1 and 2 for illustrative purposes, where

- $\mathbf{F}$ is the feature map over which the convolution operation is intended to be applied. It has a shape of $D \times M$, where D represents the spatial/temporal dimension and M the number of input channels.
- $\mathbf{K}$ is the convolution kernel and has a shape of $S \times M$ where S is the size of the kernel.
- $\mathbf{G}$ is the output tensor produced by applying the *regular convolution* operation with the filter K over the input F resulting in a vector of length D.
- $\hat{\mathbf{G}}$ is the output tensor produced by applying the *depthwise convolution* with the filter K over the input F, resulting in a matrix with shape $D \times M$. The number of output channels is restricted to be equal to the number of input channels M.

$$\mathbf{G}_x = \sum_{s,m}^{S,M} \mathbf{K}_{s,m} \cdot \mathbf{F}_{x+s-\frac{S-1}{2},m} \tag{1}$$

$$\hat{\mathbf{G}}_{x,m} = \sum_{s}^{S} \mathbf{K}_{s,m} \cdot \mathbf{F}_{x+s-\frac{S-1}{2},m} \tag{2}$$

See that $\mathbf{G}_x = \sum_m^M \hat{\mathbf{G}}_{x,m}$. A *pointwise convolution* would be equivalent of applying a *regular* convolution with $S = 1$.

The *depthwise separable convolution* is composite function of a *pointwise convolution* and a *depthwise convolution* as follows: $PW(DW(F, W_d), W_p)$ where PW and DW refer to the *pointwise* and *depthwise convolutions*, and W_p, W_d to their weights, similarly.

The *regular convolution* layer targeting N output channels has a computational cost of $D \cdot M \cdot N \cdot S$. The *depthwise convolution* has a computational cost of $D \cdot M \cdot S$, and the *separable convolution* $D \cdot M \cdot N$. Therefore the *depthwise separable convolution* has a computational cost of $D \cdot M \cdot S + D \cdot M \cdot N$, i.e. the sum of its two constituting operations. Hence, by using the *depthwise separable convolution* in place of the *regular* one we have a cost reduction of:

$$\frac{D \cdot M \cdot S + D \cdot M \cdot N}{D \cdot M \cdot N \cdot S} = \frac{1}{N} + \frac{1}{S}$$

[5] in both cases assuming a stride of one, *SAME* padding and odd size convolution kernels

Appendix 2

Table 3: Detailed results for task *35-words-recognition* and V1 data set, in decreasing f1-score order. The columns "precision", "recall" and "f1-score" have been represented as the mean ± the standard deviation in percentage scale.

	precision	recall	f1-score	support
happy	98.00±0.89	97.80±0.75	98.00±0.63	180
cat	98.20±0.40	97.80±0.40	98.00±0.00	166
house	97.60±1.36	98.60±0.49	97.80±0.75	150
dog	97.80±0.75	98.00±0.00	97.80±0.40	180
marvin	99.00±0.63	96.80±0.75	97.80±0.40	162
stop	97.20±1.47	98.20±0.40	97.60±1.02	249
yes	98.60±1.02	96.40±0.80	97.40±0.49	256
sheila	97.20±1.33	97.00±0.63	97.20±0.75	186
wow	96.80±1.17	97.40±0.49	97.20±0.40	165
seven	95.80±1.47	98.40±0.80	97.20±0.40	239
four	96.40±0.80	97.20±0.75	97.00±0.63	253
two	95.80±1.33	97.60±0.49	96.80±0.75	264
nine	95.40±1.50	98.20±0.75	96.80±0.40	259
on	95.80±1.60	97.00±0.63	96.60±1.02	246
six	95.80±0.40	97.20±0.40	96.40±0.49	244
bird	95.80±0.98	96.40±0.49	96.20±0.75	158
eight	96.80±1.94	95.40±0.49	96.00±0.89	257
five	96.00±0.63	96.00±0.63	96.00±0.63	271
one	98.00±0.89	93.80±1.33	95.80±0.40	248
down	96.00±0.63	95.00±0.89	95.60±0.80	253
bed	95.00±1.67	96.20±1.47	95.40±1.02	176
left	93.00±1.67	97.20±0.40	95.20±0.75	267
off	96.80±1.47	94.00±1.26	95.20±0.40	262
right	97.40±1.20	92.80±0.40	95.20±0.40	259
zero	95.60±1.36	94.40±1.02	95.00±0.63	250
up	94.80±0.75	94.80±0.98	94.80±0.75	272
no	94.60±1.02	93.00±0.89	93.80±1.17	252
go	94.60±1.62	93.40±0.80	93.80±0.75	251
tree	92.40±1.50	90.60±1.36	91.60±0.49	193
three	89.60±0.49	92.80±0.75	91.20±0.40	267
avg/total	96.00±0.00	96.00±0.00	96.00±0.00	6835

Table 4: Detailed results for task *35-words-recognition* and V2 data set, in decreasing f1-score order. The columns "precision", "recall" and "f1-score" have been represented as the mean ± the standard deviation in percentage scale.

	precision	recall	f1-score	support
yes	99.20±0.40	98.60±0.49	99.00±0.00	419
stop	98.00±1.10	99.40±0.80	98.60±0.49	411
seven	97.60±0.49	98.80±0.40	98.40±0.49	406
six	96.40±0.49	98.60±0.49	97.60±0.49	394
right	98.40±0.49	96.20±0.75	97.40±0.49	396
sheila	97.60±0.49	97.20±0.75	97.20±0.75	212
nine	97.40±0.49	96.80±0.40	97.20±0.40	408
eight	97.60±0.49	97.00±0.63	97.20±0.40	408
marvin	98.00±0.89	96.40±0.49	97.20±0.40	195
five	96.80±0.40	97.40±0.80	97.00±0.00	445
house	96.80±0.98	97.00±0.63	96.80±0.75	191
happy	97.60±0.49	96.00±0.63	96.80±0.40	203
zero	97.20±0.98	96.20±0.40	96.60±0.49	418
left	94.60±1.50	98.00±0.63	96.40±0.80	412
backward	94.60±0.49	98.20±0.75	96.40±0.49	165
one	96.60±0.80	96.20±0.75	96.40±0.49	399
two	95.40±1.02	97.40±1.02	96.20±0.75	424
wow	97.20±0.75	94.40±0.80	96.00±0.89	206
off	97.00±1.26	94.80±1.17	95.80±0.75	402
on	96.00±0.89	95.40±1.02	95.80±0.40	396
visual	96.00±1.67	95.20±1.33	95.60±0.80	165
go	96.00±0.63	95.20±0.98	95.40±0.49	402
no	93.80±1.72	96.80±0.75	95.40±0.49	405
up	94.20±2.32	96.20±0.75	95.20±0.98	425
cat	96.00±0.00	94.00±0.89	95.20±0.75	194
three	93.60±1.02	97.40±0.80	95.20±0.75	405
four	94.60±1.02	93.00±1.10	94.00±0.63	400
bird	92.60±1.02	95.60±0.49	94.00±0.63	185
down	94.40±0.80	93.60±0.49	94.00±0.00	406
dog	93.40±2.24	93.40±1.36	93.40±0.80	220
bed	94.40±1.36	91.60±2.06	93.20±1.17	207
follow	90.80±2.32	92.00±1.10	91.60±1.36	172
tree	94.80±1.72	86.20±1.72	90.40±1.20	193
forward	86.60±2.15	90.00±2.10	88.20±1.72	155
learn	90.80±1.47	84.40±1.36	87.60±1.02	161
avg/total	96.00±0.00	96.00±0.00	96.00±0.00	11005

Table 5: Detailed results for task *20-commands-recognition* V1 data set, in decreasing f1-score order. The columns "precision", "recall" and "f1-score" have been represented as the mean ± the standard deviation in percentage scale.

	precision	recall	f1-score	support
nine	97.80±1.17	97.60±1.02	97.80±0.40	259
stop	97.00±1.79	98.00±0.63	97.60±1.02	249
yes	98.60±0.80	96.40±0.49	97.60±0.49	256
seven	96.40±0.49	98.20±0.75	97.40±0.49	239
six	97.20±0.75	97.40±0.49	97.20±0.40	244
unknown	96.60±0.49	97.00±0.00	97.00±0.00	1716
on	96.40±1.20	97.20±0.40	96.80±0.40	246
five	96.80±1.17	95.60±0.80	96.20±0.75	271
one	98.00±0.63	94.20±0.40	96.20±0.40	248
zero	96.60±1.50	94.60±1.02	95.80±0.98	250
four	94.00±1.10	97.60±0.49	95.80±0.75	253
two	94.80±1.60	96.40±0.80	95.60±1.02	264
left	93.60±1.02	97.20±0.75	95.40±0.80	267
eight	95.60±0.49	95.40±0.80	95.40±0.49	257
right	96.60±1.02	93.80±1.17	95.20±0.98	259
off	97.20±1.17	93.60±1.02	95.20±0.75	262
up	95.80±0.40	94.40±1.50	95.00±0.63	272
down	95.80±0.75	93.40±0.80	94.60±0.49	253
no	93.20±1.33	94.80±0.75	94.00±0.63	252
go	94.00±1.55	91.40±1.62	92.60±0.49	251
three	91.00±0.89	92.00±1.55	91.40±1.02	267
avg/total	96.00±0.00	96.00±0.00	96.00±0.00	6835

Table 6: Detailed results for task *20-commands-recognition* and data version V2, sorted by decreasing f1-score order. The columns "precision", "recall" and "f1-score" have been represented as the mean ± the standard deviation in percentage scale.

	precision	recall	f1-score	support
seven	98.80±0.40	98.60±0.49	98.80±0.40	406
yes	98.80±0.40	98.60±0.49	98.60±0.49	419
stop	98.80±0.40	99.00±0.63	98.60±0.49	411
six	97.60±1.02	98.20±0.75	97.60±0.49	394
eight	98.00±0.63	96.40±0.49	97.00±0.00	408
zero	97.80±0.75	96.40±0.49	96.80±0.40	418
nine	98.00±0.63	95.40±0.49	96.60±0.49	408
right	97.40±1.36	96.00±0.89	96.60±0.49	396
two	95.80±0.75	96.80±0.75	96.40±0.49	424
five	96.60±1.02	95.40±1.20	96.00±0.63	445
one	97.40±0.49	95.00±0.00	96.00±0.00	399
left	94.40±0.80	97.60±0.49	96.00±0.00	412
off	97.20±0.75	94.60±1.20	95.80±0.75	402
no	95.20±1.47	96.40±0.49	95.80±0.75	405
on	95.60±1.62	95.20±0.75	95.40±0.49	396
up	95.40±0.49	95.20±0.40	95.40±0.49	425
three	94.40±1.02	96.20±1.17	95.00±0.00	405
unknown	94.40±0.49	96.00±0.63	95.00±0.00	2824
go	95.00±1.41	94.80±0.75	94.80±0.40	402
down	94.80±0.40	93.40±0.49	94.20±0.40	406
four	95.20±0.98	92.00±1.67	93.60±0.49	400
avg/total	96.00±0.00	96.00±0.00	96.00±0.00	11005

Table 7: Detailed results for task *10-commands-recognition* and data version V1, sorted by decreasing f1-score order. The columns "precision", "recall" and "f1-score" have been represented as the mean ± the standard deviation in percentage scale.

	precision	recall	f1-score	support
unknown	97.60±0.49	99.00±0.00	98.20±0.40	4268
stop	98.20±1.17	97.60±0.80	97.80±0.40	249
yes	98.60±1.50	95.60±0.80	97.00±0.89	256
on	97.60±1.02	95.80±0.40	96.80±0.40	246
left	95.60±1.02	95.60±0.80	95.60±0.49	267
right	96.60±1.50	92.80±0.75	94.40±0.80	259
up	95.60±1.36	93.00±0.89	94.40±0.80	272
off	96.40±1.50	92.40±1.50	94.40±0.49	262
down	95.40±0.49	92.80±0.75	94.20±0.40	253
no	94.80±0.75	91.80±0.40	93.20±0.40	252
go	93.60±2.24	92.40±1.62	92.80±1.33	251
avg/total	97.00±0.00	97.00±0.00	97.00±0.00	6835

Table 8: Detailed results for task *10-commands-recognition* and data version V2, sorted by decreasing f1-score order. The columns "precision", "recall" and "f1-score" have been represented as the mean ± the standard deviation in percentage scale.

	precision	recall	f1-score	support
unknown	98.20±0.40	99.00±0.00	99.00±0.00	6931
yes	99.00±0.63	98.40±0.49	98.80±0.40	419
stop	98.40±0.49	98.60±0.49	98.40±0.49	411
right	97.80±0.75	95.60±1.02	96.80±0.75	396
left	94.40±1.02	97.40±0.49	95.80±0.40	412
go	95.40±1.02	94.80±0.75	95.00±0.63	402
up	96.20±0.75	93.60±0.49	95.00±0.00	425
no	93.80±1.33	95.20±1.33	94.80±0.75	405
off	95.40±0.80	93.80±0.40	94.60±0.49	402
on	95.60±1.02	93.80±0.75	94.40±0.49	396
down	94.80±0.98	93.00±0.89	94.00±0.63	406
avg/total	97.60±0.49	97.60±0.49	97.60±0.49	11005

Table 9: Detailed results for task *left-right* and data version V1, sorted by decreasing f1-score order. The columns "precision", "recall" and "f1-score" have been represented as the mean ± the standard deviation in percentage scale.

	precision	recall	f1-score	support
unknown	99.00±0.00	100.00±0.00	99.20±0.40	6309
left	95.40±1.96	92.00±0.89	93.60±0.80	267
right	96.20±1.94	87.60±1.85	91.80±0.75	259
avg/total	99.00±0.00	99.00±0.00	99.00±0.00	6835

Table 10: Detailed results for task *left-right* and data version V2, sorted by decreasing f1-score order. The columns "precision", "recall" and "f1-score" have been represented as the mean ± the standard deviation in percentage scale.

	precision	recall	f1-score	support
left	95.80±0.98	94.00±0.89	95.00±0.63	412
right	98.20±1.47	90.60±2.65	94.00±1.10	396
unknown	99.40±0.49	100.00±0.00	100.00±0.00	10197
avg/total	99.00±0.00	99.00±0.00	99.00±0.00	11005