Beating human performance at recognizing speech commands in temporal domain

Iván Vallés Pérez[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 speech 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* [1] algorithm which achieved outstanding results in several computer vision tasks [2, 3, 4, 5]. We got about 96% accuracy when classifying audio clips belonging to 35 different categories, beating human performance in the most complex tasks proposed.

Keywords: Speech Recognition, Deep Learning, Representation Learning

1. Introduction

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

Although there has been an outstanding evolution in this domain, there is a real room for improvement in making virtual assistants behave as close as humans as possible. There are several possible work lines, for instance: improving the accuracy of the responses [6], reducing the delay of their answers [7] or increasing their variability of responses [8]. This work focuses on improving the accuracy of the voice commands recognition for the limited vocabulary case.

The current commercial virtual assistants are still not as accurate as humans at identifying human voice commands. Several current efforts have been made and nearly human performance has been reported in several studies [9, 10, 11, 12, 13], but there are still opportunities for enhancement.

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

- Design and implementation of a convolutional neural network based architecture able to surpass the current state of the art results and the human performance keeping the data in the temporal domain (using the raw version of the waveform).
- Development of a new methodology for augmenting audio data to increase the size of a data set (5x in this work).
- 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.

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. Section 3 explains the methodology and algorithms that have been used in order to get to the results presented in section 4. Section 5 makes a summary of the insights, take aways and possible future work lines.

2. Data

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* [14, 15]. 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 [14, 15].

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[1] set contain 64,721 and 105,829 audio clips of 1 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.

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

- *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”). It is important to notice that in the version V1 of the data set there are 5 missing commands and hence, the task consists of a 30 classes classification even though the task is called *35-words-recognition*.

- *20-commands-recognition*: entails the categorization of all the clips representing the most commonly used commands in robotics [15] and numbers

[1]this is the original notation suggested by the authors of the data set, but for simplicity we will refer to versions 0.01 and 0.02 as V1 and V2, respectively, from now on

("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.

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, which have been 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 [16]. 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 [16].

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

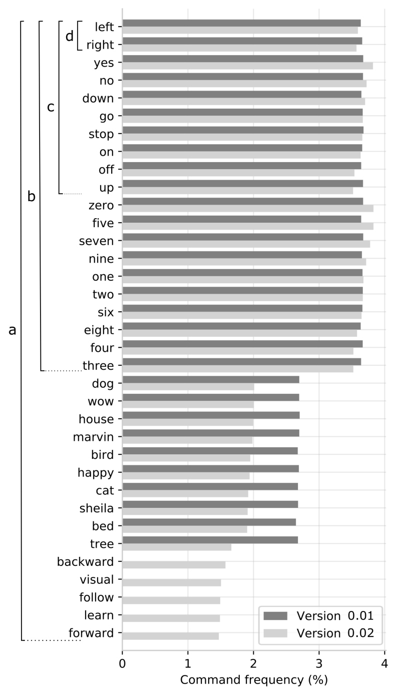

Figure 1: Command frequency distribution for both versions of the data set. As it can be noticed, the V2 is a refined and extended version of V1. In the left, the four different tasks that have been benchmarked in this work: (a) referred as *35-words-recognition* and comprising in both cases all the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

- **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 in the end (left offset) [16].
- **White noise addition**: white noise (with gaussian distribution) is added to the clips with a given amplitude [16].
- **Pitch shift**: the pitch of the clips is increased or decreased a given amount, producing more acute or more severe voice tones [16, 17].

All the distortions are applied together with random intensities only over 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.

3. Methods

We propose the use of a *convolutional neural network* with *depthwise separable convolution* layers and *residual connections*, based on the *Xception architecture* described by *François Chollet* in 2017 [1]. This model is a convolutional neural network based architecture which recently achieved state of the art results in multiple computer vision tasks [2, 3, 4, 5].

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 [1]. This is achieved by following the two steps described below. The whole operation is shown in the figure 2.

1. *Depthwise convolution* [18]: consists of applying a single filter to each channel. It produces an output tensor with potentially different spatial/temporal dimensions than the input one (depending on the stride, the size and the padding of the convolution operation to be applied). The channels dimension of the output tensor remain the same as the input.
2. *Pointwise convolution* [19]: consists of applying a set of size-1 convolutions (as many as number of output channels desired). This operation potentially modifies the channels dimension, leaving the spatial/temporal dimensions intact.

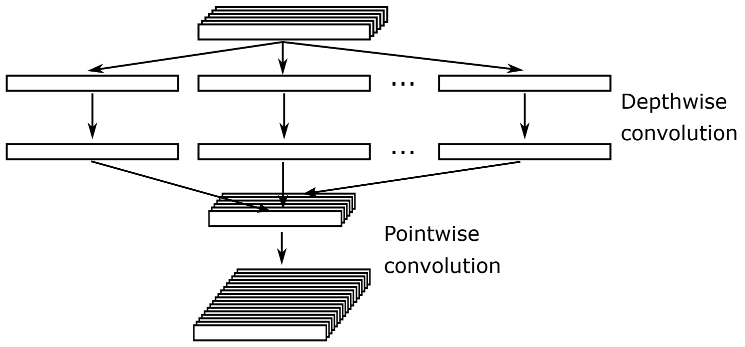

Figure 2: *Depthwise separable convolution* diagram showing how the whole computation is done in two steps: the *depthwise convolution* followed by the *pointwise convolution*.

A *regular convolution* [20] is described in equation 1 and the *depthwise convolution* [20] can be described[3] as shown in equation 2, where

- $\mathbf{F}$ represents the feature map over which the convolution operation is intended to be applied. It has a shape of $D_X \times M$, where D_X represents the spatial/temporal dimension and M the number of input channels.
- $\mathbf{G}$ is the filtered output tensor produced by the *regular convolution* operation, with shape $D_X \times N$, where N is the number of output channels.
- $\hat{\mathbf{G}}$ is the filtered output tensor produced by the *depthwise convolution* operation, with shape $D_X \times N$, where N is the number of output channels.
- $\mathbf{K}$ represents the convolution kernel of a *regular convolution* and has a shape of $D_X \times M \times N$.
- $\hat{\mathbf{K}}$ represents the convolution kernel of a *depthwise convolution* and has a shape of $D_X \times M$, where the m_{th} filter is applied to the m_{th} channel in $\mathbf{F}$ producing the m_{th} channel of the output tensor $\hat{\mathbf{G}}$

$$\mathbf{G}_{x,c} = \sum_{i,m} \mathbf{K}_{i,m,n} \cdot \mathbf{F}_{x+i-1,m} \tag{1}$$

[3]in both cases assuming a stride of one and *SAME* padding

$$\hat{\mathbf{G}}_{x,c} = \sum_{i,m} \hat{\mathbf{K}}_{i,m} \cdot \mathbf{F}_{x+i-1,m} \tag{2}$$

As it can be noticed in equation 2, 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. That is why a size-1 regular convolution (also known as *pointwise convolution*) is applied after the *depthwise convolution*. Both operations make up the *depthwise separable convolution*. The reason why the *separable* term is used in the name of the operation is because there is a separation between the channel-wise and the spatial/temporal-wise computations. The number of sums and multiplications required by this operation is $\frac{1}{N} \cdot \frac{1}{D_X}$ times the number of operations required by a regular convolution [20], which represents a meaningful performance improvement for big networks.

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 [1]. 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[4].

[4]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*

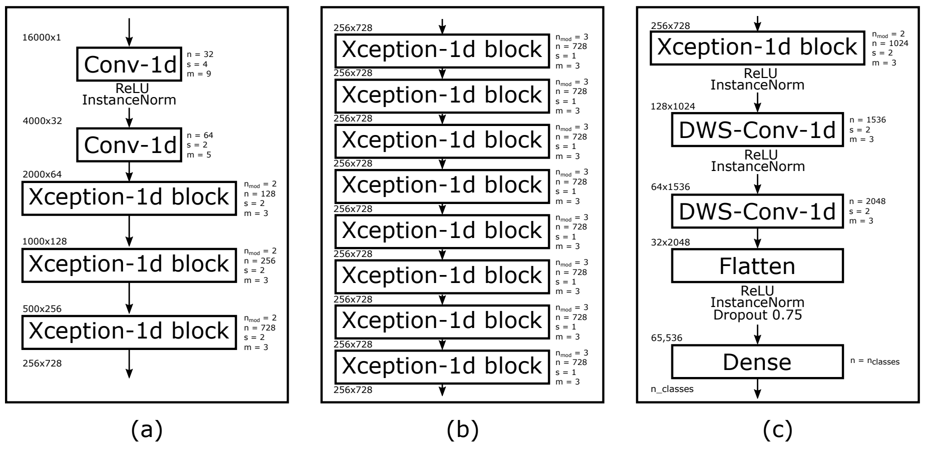

Figure 3: Diagram of the architecture used in this work where n refers to the number of channels, s is the stride of the convolutions (which in the case of the *Xception-1d blocks* is applied through the final pooling operation), m is the size of the convolution filters, n_{mod} is the number of the *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 model to distinguish between classes and (c) the exit module is in charge of 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 [21, 22] has been applied after the last convolution ($p = 75\%$). In addition, a small *weight decay* [23, 24, 22] 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 [25]. 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. in the *plateaus*) with a patience of 4 epochs.

Instance normalization has been used to normalize the intermediate outputs of each convolution [26, 27]. It simply consists of standardizing separately each of the samples across channels. Although it is typically used in generative

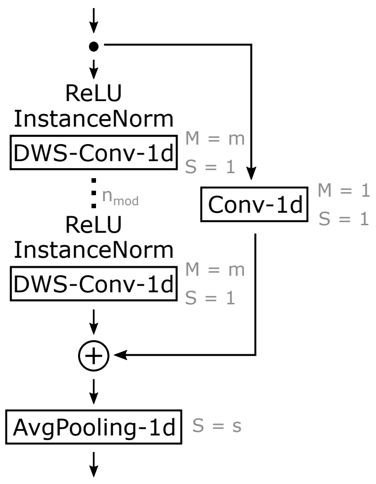

Figure 4: Neural network module refered as *Xception-1d block*. The *Xception-1d block* represents the building block of the architecture proposed in this work. It consists of a set of n_{mod} *1-D depthwise separable convolution* layers with a *residual connection* and with a *1-D average pooling* layer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

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[5] [26]; it is considered a key element of this architecture. The input of the dense layer has been normalized using *layer normalization* [28]. *Batch normalization* [29] has been avoided because it is prone to introduce train-test differences which hurt final performance [28]. Figure 5 summarizes the way these three normalization techniques operate.

4. Results

The train/development/test split provided by the authors of the data set [14] has been used as *cross validation* setting in order to facilitate future results

[5]In the case of *Batch Normalization*, there can appear big differences of performance between train time and test time [28]

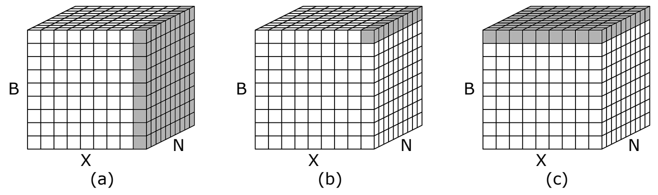

Figure 5: Each cube represents a batch of data where the dimension marked as B is the batch size, the one labeled as N is the number of channels and X is the number of temporal samples. (a) represents *batch normalization*, (b) is *instance normalization* and (c) shows *layer normalization*. The shaded 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 the batches of data is *batch normalization*, that is why it has undesirable effects when the train and test data distributions are different. As it can be noticed in the picture, *instance normalization* is a restricted version of *layer normalization*

comparisons. 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 and the weights of the epoch, achieving the best performance in the development set have been checkpointed. The checkpointed models have been used to 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 with 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 ~ 1000 commands achieving different results. These results are reported in table 1 along with the results of the proposed algorithm and the results reported by [9, 13, 15, 11] 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, at the end of the article.

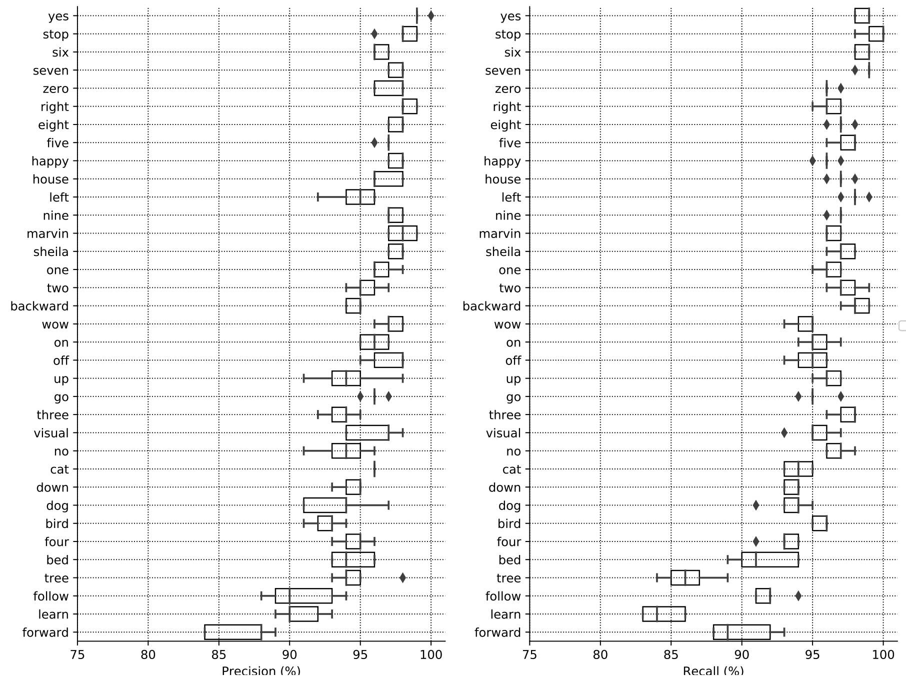

Figure 6: Precision and recall for each of the different classes using the *35-words-recognition* model trained with data version V2. It can be noticed that the algorithm performs generally well in all the classes, but it has more difficulties differentiating some groups of similar words like the following pairs: “three” and “tree”, “follow” and “four”, “bed” and “bird”, etc. There has not been included any comparison with other models because no such detailed results have been found in the related work.

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 recognition task with limited vocabulary achieving state of the art results. This is why we suggest *Xception-1d* as the *de facto*

Table 1: Accuracy obtained by our algorithm in the different tasks compared to other algorithms (described in [9, 13, 15, 11]) and compared to our measurement of the human accuracy (through 4 manual evaluations). The results of best performing algorithms for each task have been marked in bold in each case. The results performing statistically better than human (given a statistical significance level of $\alpha = 0.05$) have been marked with a star (*).

(a) Results for version 1 of the data set

	Andrade et al.[9]	McMahan et al. [13][a]	Warden [15]	Ours	Human	p-value[b]
35-words	94.30	84.35	-	**95.85 $\pm$ 0.12 ***	94.15 $\pm$ 1.03	$1.46 \cdot 10^{-2}$
20-commands	94.10	85.52	-	**95.89 $\pm$ 0.06 ***	94.56 $\pm$ 0.98	$3.14 \cdot 10^{-2}$
10-commands	95.60	-	85.40	**97.15 $\pm$ 0.03**	97.22 $\pm$ 0.85	$8.75 \cdot 10^{-1}$
left-right	**99.20**	95.32	-	98.96 $\pm$ 0.09	99.54 $\pm$ 0.16	$5.24 \cdot 10^{-4}$

(b) Results for version 2 of the data set

	Andrade et al. [9]	Zhang et al. [11][a]	Warden [15]	Ours	Human	p-value [b]
35-words	93.90	-	-	**95.85 $\pm$ 0.16 ***	94.15 $\pm$ 1.03	$1.50 \cdot 10^{-2}$
20-commands	94.50	-	-	**95.96 $\pm$ 0.16 ***	94.56 $\pm$ 0.98	$2.70 \cdot 10^{-2}$
10-commands	96.90	95.40	88.20	**97.54 $\pm$ 0.08**	97.22 $\pm$ 0.85	$4.84 \cdot 10^{-1}$
left-right	**99.40**	-	-	99.25 $\pm$ 0.07	99.54 $\pm$ 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. $\alpha = 0.05$

architecture when a voice commands recognition with restricted vocabulary task arises; considering the computing power is not a limiting factor. The algorithm presented can have multiple applications for improving voice-controlled systems.

Future work is 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).

References

[1] 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 (July 2017). `doi:10.1109/CVPR.2017.195`.

[2] 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`.

[3] 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`.

[4] 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 (December 2018). `doi:10.1109/TIP.2018.2864920`.

[5] 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`.

[6] 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 (September 2017).

[7] 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 (2017). `doi:10.1145/3020078.3021745`.

[8] 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 (September 2017).

[9] 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`.

[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 (June 2018).

[11] 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`.

[12] 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 (September 2015).

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

[14] 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).

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

[16] 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).

[17] 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 (September 2016). `doi:10.1117/12.2249374`.

[18] 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).

[19] 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 (September 2018).

[20] 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`.

[21] 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 (January 2014).

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

[23] A. Krogh, J. A. Hertz, A simple weight decay can improve generalization, in: Proceedings of the 4th International Conference on Neural Informa-

tion Processing Systems, NIPS'91, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1991, pp. 950–957 (1991).

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

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

[26] 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`.

[27] 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`.

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

[29] 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 (February 2015).

Appendix

Table 2: Detailed results for task *35-words-recognition* and data version V1, sorted by decreasing f1-score order. The columns “precision”, “recall” and “f1-score” have been represented as the mean $\pm$ 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 3: Detailed results for task *35-words-recognition* and data version V2, sorted by decreasing f1-score order. The columns "precision", "recall" and "f1-score" have been represented as the mean $\pm$ 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 4: Detailed results for task *20-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
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 5: 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 6: 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 7: 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 8: 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 9: 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