# BinaryCmd: Keyword Spotting with deterministic binary basis

Javier Fernández-Marqués[†], Vincent W.-S. Tseng[‡], Sourav Bhattacharya[*], Nicholas D. Lane[†,*]

[†] University of Oxford, [‡] Cornell University, [*]Nokia Bell Labs

## ABSTRACT

We present a compact binary architecture with 60% fewer parameters and 50% fewer operations (OP) during inference compared to the current state of the art for keyword spotting (KWS) applications at the cost of 3.4% accuracy drop. This architecture makes use of binary orthogonal codes to analyze speech features from a voice command resulting in a model with minimal memory footprint and computationally cheap, making possible its deployment in very resource-constrained microcontrollers with less than 30kB of RAM.

## 1 INTRODUCTION

KWS has become a popular always-on feature in smartphones, wearables and smart home devices. It serves as the entry point for speech based applications once a predefined command (e.g. "Ok Google", "Hey Siri") is detected from a continuous stream of audio. Because KWS applications are always running they follow a very efficient architectural design and are often implemented on small dedicated microcontrollers. These devices are constrained in terms of memory and compute capabilities, limiting the complexity and memory footprint of the deployed model.

We present BinaryCmd, read as "binary command", a novel neural network (NN) architecture for audio that represents the weights as a combination of predefined orthogonal binary basis that can be generate very efficiently on-the-fly. This property enables the off-loading of the convolutional filters from the model, resulting in models with smaller memory footprint and a more efficient inference stage. Inspired by ResNet's *bottleneck* layer [5] and LBCNN [9], where the suitability of sparse binary filters for image classification tasks is proven, we present a vastly reduced architecture from those generally use for vision problems and adjusted it at both macroarchitectural and microarchitectural levels to better capture the temporal dimension of input audio commands.

We compare our work to *HelloEdge* [15] following their microcontroller classification scheme and particularly focusing on the Small (S) group, where the model size limit is set to 80kB and the maximum number or OPs during inference is 6M. Likewise, we use Google's Speech Commands Dataset [13] to train and evaluate our architecture. BinaryCmd requires significantly less parameters and OPs than the best architecture in [15] that relies in depthwise separable convolutional neural networks (DS-CNN) [4, 6, 14] and that is, to the best of our knowledge, the current state of the art for KWS applications.

## 2 A BINARY NETWORK FOR KWS

**System Overview**. The implemented KWS system is comprised of two fundamental blocks where speech features are first extracted from the 1*s* voice command input and are fed to a NN-based block that outputs the id of the detected voice command. The system's macroarchitecture is depicted in Figure 1. We follow the same strategy as in [15] to extract an array of $49 \times 10$ MFCC speech features from the input speech signal and feed them to our network.

**Architecture**. We present a novel NN block containing the following elements: three nested *on-the-fly* convolutional layers (they represent BinaryCmd's core, Figure 2) followed by a standard convolutional, max-pooling and fully connected layers. All convolutional layers use ReLu as activation functions and have been trained using batch normalization [7].

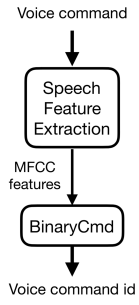


**Figure 1: System architecture.**

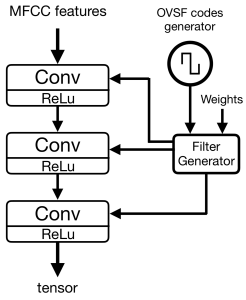


**Figure 2: BinaryCmd's core.**

***On-the-fly* convolutions**. Unlike standard convolutional neural networks (CNN), our architecture does not learn convolutional filters directly. Instead, it learns weighting coefficients of deterministic binary basis that are combined in a linear fashion manner to generated the filters. We use orthogonal variable spreading factor, OVSF[1], binary codes of length $2^n$, $n \in \mathbb{N}$, to generate these basis. The filter creation process using the OVSF basis can be didacti-

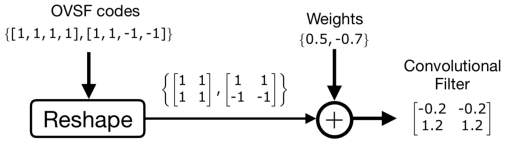


**Figure 3: Filter generation process. Here we show an example of using two OVSF basis to generate a $1 \times 2 \times 2 \times 1$ convolutional filter.**

cally illustrated as in Figure 3: First the generator outputs a set of $2^n-$dimensional arrays of $[+1, -1]$ elements; then the arrays get reshaped to match the dimensions of the convolutional filter (a 4-dimensional tensor); and finally they get combined using the learned weights. We use the generated filters in our convolutional layers.

This work has been implemented in TensorFlow [1] using as base the source code provided in [15].

[1] OVSF codes were introduced for 3G communication systems as channelizations codes aiming to increase system capacity in multi-user access scenarios[2].

## 3 EVALUATION

We evaluate three configurations of BinaryCmd with a focus on reducing on-device memory footprint and number of OPs per inference pass while maintaining acceptable accuracy rates that outperform other models [3, 10–12] found in the recent literature with comparable number of OPs. The three configurations share the majority of network parameters and only differ in the number of filters, stride and ratio parameters used in our on-the-fly convolutional layers. The parameters used in our experiments are shown in Table 2. For a given filter dimensions $dim = inCh \times w \times h \times outCh$ there are $dim$ $dim$-dimensional OVSF orthogonal binary basis. The ratio parameter, $\rho \in [0, 1]$, specifies the percentage of basis that are used to generate a filter. Intuitively, the smaller the ratio, the coarser the filters and the bigger the model size savings would be, and vice-versa.

| Model | Acc. | Memory | OPs | A2S/A2OPs |
|---|---|---|---|---|
| DS-CNN [15] | **94.4%** | 38.6kB | 5.4M | 2.45/17.48 |
| CRNN [15] | 94.0% | 79.7kB | 3.0M | 1.18/31.33 |
| GRU [15] | 93.5% | 78.8kB | 3.8M | 1.19/24.6 |
| LSTM [15] | 92.9% | 79.5kB | 3.9M | 1.17/23.82 |
| Basic LSTM [15] | 92.0% | 63.3kB | 5.9M | 1.45/15.59 |
| CNN [15] | 91.6% | 79.0kB | 5.0M | 1.16/18.32 |
| BinaryCmd-A | 91.4% | 24.5kB | **1.8M** | 3.73/**50.78** |
| BinaryCmd-B | 91.2% | 22.8kB | 2.3M | 4.00/39.57 |
| BinaryCmd-C | 91.0% | **15.8kB** | 2.6M | **5.76**/34.96 |
| DNN [15] | 84.6% | 80.0kB | **0.16M** | 1.05/**528.75** |

**Table 1: Comparison of three BinaryCmd configurations against DS-CNN, the current state of the art for KWS applications, and other baselines presented in [15] for microcontrollers limited to a maximum of 80kB of memory and 6M OPs. Details for each BinaryCmd configuration are shown in Table 2.**

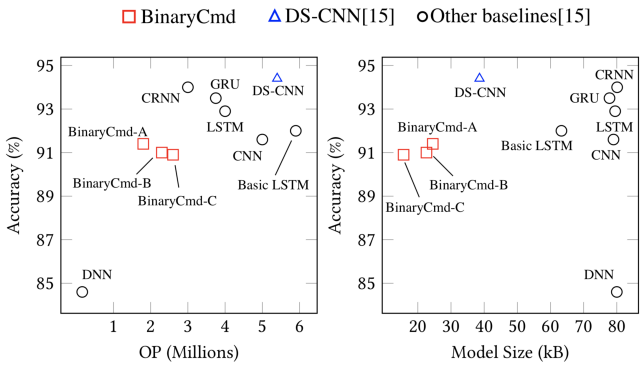


**Figure 4: Results comparison against architectures in [15] for the category of small (S) microcontrollers. DS-CNN has never been tuned below 38.6kB and 5.4M OPs. All other configurations in [15] result in larger and computationally more expensive models: (189kB |19.8M OPs) and (497kB | 56.9M OPs), respectively for medium and large categories of microcontrollers.**

We compare BinaryCmd against DS-CNN and all the baselines analysed in [15]. Our configurations explore the void space of 1M-3M OPs and 10kB-25kB. Architectures lying in those range of values would fit in some commercially available mid-low end ARM microcontrollers, as show in Table 3. Our preliminary results, Figure 4, show the potential of our binary architecture: up to 59% model size and 67% number of OPs reduction at the expense of no more than 3.4% accuracy loss when compared to DS-CNN. All three of our configurations simultaneously achieve top *accuracy-to-size* (A2S) and *accuracy-to-OPs* (A2OPs) ratios meaning that BinaryCmd is a good first step towards the design of architecture capable of providing over 90% accuracy levels with minimal memory footprint and low computational costs.

| Config | #Filters | Strides $[x, y]$ | Ratios ($\rho$) |
|---|---|---|---|
| A | [64, 8, 32] | [2, 2], [2, 2], [1, 1] | [1.0, 1.0, 1.0] |
| B | [64, 16, 16] | [2, 2], [2, 2], [1, 1] | [1.0, 0.5, 1.0] |
| C | [16, 16, 16] | [2, 2], [1, 1], [1, 1] | [1.0, 1.0, 1.0] |

**Table 2: Parameters in BinaryCmd for each configuration. All parameters are given in triplets since there are three on-the-fly convolutional layers (see Figure 2).**

Unlike in [15], where each architecture has been optimally trained after performing an exhaustive search for feature extraction and NN model hyperparameters, the work here presented only modifies the number of training steps from the default parameters provided in [15] source code, leaving room for more efficient training setups. Furthermore, our implementation only applies standard 8-bit quantisation to the weights of the second and third on-the-fly convolutional layers as well as the standard convolutional and fully connected layers that come later in the pipeline, meaning that further reducing the model's memory footprint is also possible. In addition, implementing a quantisation scheme [8] that jointly provides model size reductions as well as more efficient inference is a path worth exploring for applications like KWS.

| Core | Freq. | SRAM | Flash |
|---|---|---|---|
| Cortex-M0+ | 30 MHz | 16 kB | 64 kB |
| Cortex-M0+ | 48 MHz | 32 kB | 256 kB |
| Cortex-M3 | 72 MHz | 64 kB | 250 kB |

**Table 3: Selection of mid-low end ARM Cortex-M microcontrollers.**

## 4 SUMMARY AND FUTURE WORK

We have presented a binary architecture capable of giving near start of the art accuracy levels while requiring a fraction of model parameters and considerable less operations per inference pass. We make this possible by generating convolutional filters on-the-fly using binary orthogonal codes that can be generated efficiently and reduce then number of trainable parameters. In the next iterations of this work we will explore different architectures suitable for speech applications that exploit further the on-the-fly generation nature of our architecture enabling a more complex design while maintaining acceptable computational costs and model size.

## REFERENCES


[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Fumiyuki Adachi, Mamoru Sawahashi, and Hirohito Suda. 1998. Wideband DS-CDMA for next-generation mobile communications systems. *IEEE communications Magazine* 36, 9 (1998), 56–69.

[3] Sercan Ömer Arik, Markus Kliegl, Rewon Child, Joel Hestness, Andrew Gibiansky, Christopher Fougner, Ryan Prenger, and Adam Coates. 2017. Convolutional Recurrent Neural Networks for Small-Footprint Keyword Spotting. *CoRR* abs/1703.05390 (2017). arXiv:1703.05390 http://arxiv.org/abs/1703.05390

[4] François Chollet. 2016. Xception: Deep Learning with Depthwise Separable Convolutions. *CoRR* abs/1610.02357 (2016). arXiv:1610.02357 http://arxiv.org/abs/1610.02357

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). arXiv:1512.03385 http://arxiv.org/abs/1512.03385

[6] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR* abs/1704.04861 (2017). arXiv:1704.04861 http://arxiv.org/abs/1704.04861

[7] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR* abs/1502.03167 (2015). arXiv:1502.03167 http://arxiv.org/abs/1502.03167

[8] Benoît Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. 2017. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference.

[9] Felix Juefei-Xu, Vishnu Naresh Boddeti, and Marios Savvides. 2016. Local Binary Convolutional Neural Networks. *CoRR* abs/1608.06049 (2016). arXiv:1608.06049 http://arxiv.org/abs/1608.06049

[10] Tara N. Sainath and Carolina Parada. 2015. Convolutional neural networks for small-footprint keyword spotting. In *INTERSPEECH*.

[11] Ming Sun, Anirudh Raju, George Tucker, Sankaran Panchapagesan, Gengshen Fu, Arindam Mandal, Spyros Matsoukas, Nikko Strom, and Shiv Vitaladevuni. 2017. Max-Pooling Loss Training of Long Short-Term Memory Networks for Small-Footprint Keyword Spotting. *CoRR* abs/1705.02411 (2017). arXiv:1705.02411 http://arxiv.org/abs/1705.02411

[12] Zhiming Wang, Xiaolong Li, and Jun Zhou. 2017. Small-footprint Keyword Spotting Using Deep Neural Network and Connectionist Temporal Classifier. *CoRR* abs/1709.03665 (2017). arXiv:1709.03665 http://arxiv.org/abs/1709.03665

[13] Pete Warden. 2017. Speech Commands: A public dataset for single-word speech recognition. (2017). http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz

[14] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2017. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. *CoRR* abs/1707.01083 (2017). arXiv:1707.01083 http://arxiv.org/abs/1707.01083

[15] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. 2017. Hello Edge: Keyword Spotting on Microcontrollers. *CoRR* abs/1711.07128 (2017). arXiv:1711.07128 http://arxiv.org/abs/1711.07128