Power Consumption of Diverse Speech Command Classification Methods on the Raspberry Pi Zero

Frederic Brodbeck, Paul Seidler, and Daniel Devatman Hromada

[1] Digital Education, Einstein Center Digital Future, Berlin
[2] Berlin University of the Arts, Faculty of Design
{f.brodbeck,p.seidler,dh}@udk-berlin.de
bildung.digital.udk-berlin.de

Abstract. This study compares performance of different speech command classification systems which can be executed on an Raspberry Pi Zero ARMv6 architecture. Three systems are evaluated: first one, TREELITE_MFCC, is based on Treelite system cross-compiles a MFCC-classifying Random Forest into a highly optimized shared library; second, TFLITE_MFCC, performs classification of MFCC inputs by means of convolutional neural network encoded as a lightweight TensorFlow Lite model while the third one - labeled as TFLITE_RAW - uses more complex network to directly classify the audio signal. We evaluate models not only in terms of their accuracy and precision, but also in terms of execution time, voltage, current and energy consumed. We observe that while TFLITE_RAW offers superior performance in terms of accuracy, TREELITE_RAW is also worth of consideration for low-latency real-life applications since it offers relatively good performance (avg. micro-precision=0.917) but its predictions, when executed on Raspberry Pi Zero, are significantly faster and cost less energy than TensorFlow Lite models.

Keywords: speech command classification, power consumption, ARMv6, random forests, TensorFlow Lite, TreeLite, precision-over-energy

1 Introduction

1.1 Motivations

Speech command classification (SCC) is a process wherein a classifier system attributes a label to an audio input sequence. SCC can be interpreted either as an information-theoretical phenomenon, or as physical, power consuming phenomenon, or both. While evaluation by means of information-theoretical metrics like accuracy, precision, recall, F-score etc. is a well established practice in SCC and its envelopping automatic speech recognition (ASR) community, the physical, power-consuming aspect of the process of sound classification is much less explored. It is often the case that the only bridge between the computer-scientific aspect of a certain method and its real-world performance is constrained to discussion of the algorithm in terms of its computational complexity (e.g. [12], [15]).

But given that there is, indeed, a long way between theoretical complexity of the algorithm and its material realization, a more concrete, empiric method for bridging the informational and physical is surely needed. To address this need is the first motivation of our study.

The other motivation behind this study arose in relation to a research project whose goal is the construction of a do-it-yourself (DIY) digital education artefact for elementary school pupils known as "Digital Primer" (DP). In their DP roadmap, the inceptors of the project define the property "speech-based" as one of the most important attributes of the DP and make it quite clear that corporate, cloud-based generic ASR systems are to be dismissed and replaced by offline, personalized models: "Internal speech models of the Primer should develop locally, through and by means of interaction with human Children. Ideally, speech faculties of the Primer are to adapt to speech faculties of the Child. The Child shall, in a certain sense, teach the artefact her own language, she will become its language teacher." [11]

In their later paper, the authors precise that the DP is based on a well-known off-the-shelf platform Raspberry Pi Zero [10]. Being built on top of a spartan ARMv6 architecture, the Pi Zero is an interesting piece of hardware located somewhere in the middle between task-specific microcontrollers with very low power consumption on one hand (ESP32, Artemis, Cortex) and embedded systems for generic computation (Raspberry Pi 4, Rock64) generally based on more power hungry architectures such as ARMv7 and beyond. It is most probably the combination of its low price (the cheapest variant of Pi Zero costs 5 euros), its generic computation capability, its low power consumption and its wide adoption allowing one to consider Pi Zero as a standard of sorts, which motivated the authors of the DP to base their project on the Pi Zero platform.

On the other hand, it may be the case that the authors of the DP project raise unwarranted expectations while in reality, classifiers running on Pi Zero which could offer accurate answers maybe do not even exist. Sporadic information for sound classification on microcontrollers exists [8] but often focuses on one particular method. Hence, as of March 2021, we are not aware of any study which would address the question "Is ASR on ARMv6 feasible?" in the depth its merit.

Additionally, we are not aware of any empiric study which compares different methods of sound classification on low-power architecture and does so not only in terms of information-theoretic metrics like accuracy, but also in terms of physical units describing the classifier's power consumption. The aim of the current study is to fill these gaps.

1.2 State of the art

Within this article, we compare an ASR approach based on Random Forests with connectionist "deep learning" methods. With exception of the TFLITE_-RAW model (c.f. section 3.4) where no feature extraction is necessary and the signal is fed directly to the neural network, Mel Frequency Cepstral Coefficients (MFCCs) features were extracted.

Since their introduction in 1980's in [7], MFCCs are considered to be state-of-the-art ever since [1]. As such, MFCCs are considerd to be ubiquitous in speech processing and analyzing harmonic content. [14, p. 3].

Random Forests (RFs) is an ensemble-based classifier learning method that is based on the idea of combining the votes of a multitude of decision trees. Introduced in [9], RFs have since then proven themselves as an efficient and useful classification mechanism in many domains, including acoustic event detection [13], acoustic modeling [18] or speech emotion recognition [5].

On the other hand, the domain of machine learning is currently dominated by approaches based on neural network architectures. A well-known example are machine learning systems based on TensorFlow [2] which presents a graph-oriented framework, utilizable in diverse computational enviroments. With the main goal of providing a solid core for deep neural network applications and tools to build these for different underlying hardware architectures, TensorFlow becomes relevant for the domain of natural language processing and speech recognition. With the emergence of Convolutional Neural Networks, Deep Keyword Spotting Systems were proposed and implemented in 2014[4]. The development of TensorFlow Lite makes it possible to deploy TensorFlow models on mobile, embedded and IoT devices "at the edge" of the network. With an interpreter size of ca. 1 MB, and a conversion mechanism to lite models there is a grown interest to expedite ML even on small edge devices. [6]

2 Dataset

Table 1. Summary of the dataset

Name	Files	Labels	Format	Length
Speech commands	64721	30	wav, 16bit, mono, 16kHz	1s[3]

We use the Speech Commands dataset which has been compiled by the TensorFlow team at Google and is available under an open source license[4]. "[It] has 65,000 one-second long utterances of 30 short words, by thousands of different people, contributed by members of the public [...]"[16]. One of the main focus points of the dataset is to enable testing and creation of ML models on the actual device and providing the ability to create baseline models quickly through a standardized dataset[17].

[3] Roughly 90% of the files are exactly 1 second long, the rest being shorter. For the random forest and the TensorFlow based approach that uses the raw signal the shorter files were padded with silence at the end. For the TensorFlow based approach that uses MFCCs the feature vector was padded to ensure equal shape for all files.

[4] The dataset can be downloaded from http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz

Table 2. Number of files per class

bed	bird	cat	dog	down	eight	five	four	go	happy	house	left	marvin	nine	no
1713	1731	1733	1746	2359	2352	2357	2372	2372	1742	1750	2353	1746	2364	2375
off	on	one	right	seven	sheila	six	stop	three	tree	two	up	wow	yes	zero
2357	2367	2370	2367	2377	1734	2369	2380	2356	1733	2373	2375	1745	2377	2376

As common to other publications dealing with the Speech Commands datasets 80% of the dataset is used for training, 10% for validation and 10% for testing.

3 Method(s)

3.1 Wald_ASR

Wald_ASR is our own baseline solution based on a random forest classifier. It uses Mel Frequency Cepstral Coefficents (MFCCs) as features which we extract from the audio files using the python_speech_features [5] library. One of the biggest challenges one encounters when dealing with speech data is the fact that length of a speech signal may vary while the vector which the classifier expects as input is of a fixed size N. Wald_ASR solves this problem by choosing the appropriate "window step" parameter for the MFCC extraction:

$$step = Duration/Segments$$

whereby *step* corresponds to the step between successive windows, *duration* corresponds to length of the recording and *segments* is a parameter of the Wald_ASR model.

Wald_ASR has four tuneable parameters: (1) the number of MFCCs, (2) the number of segments the audio signal is divided into, (3) the number of trees (estimators) that make up the random forest, and (4) the maximum tree depth. The optimal settings were determined experimentally using a brute-force approach. To find the combination of MFCCs and segments that yield the best results we ran 5 rounds of cross-validation (90:10 split) on the training data and averaged the results. The accuracy scores we obtained are visualized as heatmap in fig. 1.

In order to slightly reduce the noisiness a gaussian filter was applied. The result of this is shown in fig. 2.

In the third step we incrementally increased the number of trees to determine for which number of estimators we would reach the highest accuracy score (using the best combination of segments and cepstra from the previous step).

Going forward, we used $n = 350$ estimators, for which we capped the maximum tree depth at 25, beyond which no further improvement of accuracy could be observed.

[5] https://github.com/jameslyons/python_speech_features

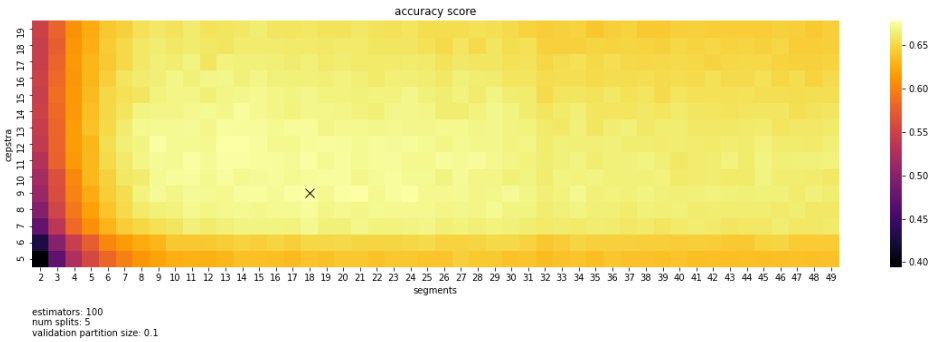


Fig. 1. Heatmap of the mean accuracy scores. The cross marks the highest value.

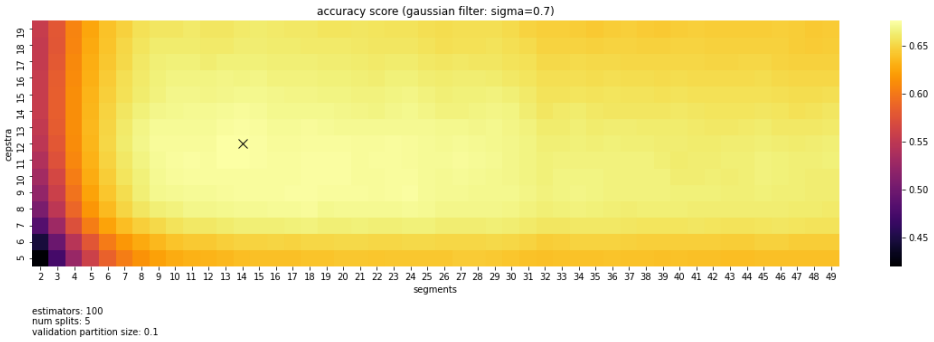


Fig. 2. Heatmap of results with a gaussian filter applied.

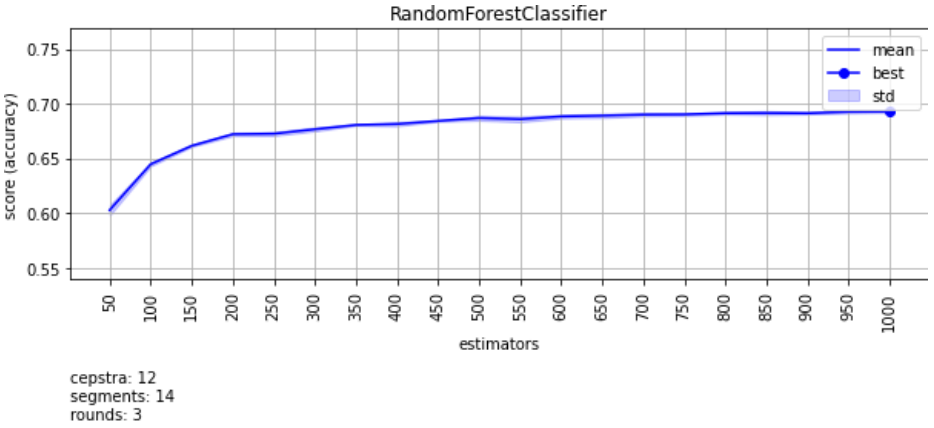


Fig. 3. Accuracy scores for different numbers of estimators

3.2 TREELITE_MFCC

As training the model on the device itself is not feasible, training was done on significantly faster multi-core machines. However, the generated files (RandomForestClassifier objects, serialized with Python's pickle module) are incompatible because of the hardware architecture mismatch (X86-64 instead of the targeted ARMv6). To be able to deploy the model to the Raspberry Pi, we used the Treelite[6] library to first convert the scikit-learn model to C code, which we then cross-compiled as a library for ARMv6. For a model with 350 estimators this yields a ca. 300 MB file which can be loaded into memory of the Pi Zero in order to perform the classification. Treelite model mentioned in this article was not further optimized with CPU branch prediction nor for float-to-integer quantization.

3.3 TensorFlow Lite

The TensorFlow model we trained broadly follows the implementation of a Convolutional Neural Network for Deep keyword spotting. "The deep neural network model is a standard feed-forward fully connected neural network with k hidden layers and hidden nodes per layer, each computing a non-linear function of the weighted sum of the output of the previous layer." [4]

Two Models with different parameters for different features were trained and converted to TensorFlow Lite (TFLITE) models, which can be run on most edge devices. To run TFLITE models on ARMv6 architecture we built bindings for the TensorFlow Lite interpreter for Python 3.7. These bindings have the advantage of not requiring the entire TensorFlow libary to be loaded but only the reduced tflite_runtime which significantly reduces the memory consumption [7].

For the purpose of this article, we have built two TFLITE models: TFLITE_MFCC which involves the MFCC extraction component and TFLITE_RAW which does not.

3.4 TFLITE_RAW

The first Model was trained on data which consisted of the PCM waveform with a choosen number of samples and fixed sample rate. The model has 847244 parameters and its underlying structure corresponds to a convolutional neural network (CNN) including two preprocessing layers: resizing and normalization.

The 1D CNN is suitable for the processing of pure PCM waveforms as these only require two dimensions and one direction as time series data. Given that TFLITE_RAW does not require a feature extraction step, we measure execution time and energy consumption only for the prediction phase.

[6] https://treelite.readthedocs.io/

[7] URL to public repository where these bindings are available will be provided in camera-ready version of the article

3.5 TFLITE_MFCC

The second 2D CNN Model was trained with 416199 parameters on MFCC features which were extracted using the python_speech_features library. 16 cepstra where used in the feature extraction, as well as an nfft size of 4096. To deal with the problem of samples having different length, the resulting matrices were zero-padded to math the shape $(16, 20)$ whereby 16 represents the number of cepstra and 20 corresponds to maximum anticipated number of frames within the sample. For this model we measure execution time and energy consumption for the feature extraction process as well as the prediction process.

4 Experimental protocol

Each simulation was repeated on three different Pi Zero devices with identical hardware and software setup. Each run consisted a of batch of measurements in which the entirety of the testing portion of the dataset (6835 samples) was forwarded to the classifier.

As will be further elaborated in the following subsections, every classification act was measured in terms of time and power consumption. Power consumption measurements were done a separate thread which was initiated right before the input feature vector is passed to classifier. The same is true for the time measurement, except it was done on the main thread.

The stop time measurement, i.e. the assessment of time after classification is over, takes place immediately after the classifier prediction function returns the label prediction vector. An instruction immediately following the stop time measurement sends the stop message to the voltage & current measurement thread. Operations happening after the prediction vector is returned - e.g. the overall statistical evaluation of the dataset in terms of prediction accuracy, etc. - were not part of the measurements.

In case the simulation includes a feature extraction (FE) component, the same experimental protocol applies for feature extraction stage as it applies for classification stage. Thus, FE measurement starts immediately before the audio signal sequence is forwarded from the buffer to the FE function and stops as soon as possible after the FE function returns. Loading of data from the SD card into the buffer was not part of the measurements.

4.1 Time measurement

The elapsed time was measured with the perf_counter()[8] function from the time module of the Python standard library, as it provides higher resolution than other functions, such as time.time() or time.clock() (for which the accuracy can vary depending on to the operating system).

[8] https://docs.python.org/3/library/time.html#time.perf_counter

> The time.perf_counter() function used in the solution provides the highest-resolution timer possible on a given platform. However, it still measures wallclock time, and can be impacted by many different factors, such as machine load. [3, p. 589]

In order not to incur any additional system load the device was not interacted with while the simulation was running. The simulations were started in a Tmux session which was detached right after the initial script invocation and only reconnected to again after the script completed.

4.2 Power consumption measurements

For the measurement of the power consumption we used a Witty Pi 3 Rev2 module[9]. This extension board adds power management functionalities to the Raspberry Pi and among other things makes it possible to get readings of the output voltage and current which feed the PiZero itself. The Witty Pi connects to the Pi Zero's GPIO pins and works as an I2C slave. The Pi Zero can read the information about from respective registers via the smbus2[10] Python library. Within all simulations reported in this study, interval between such "read from registers" was 50 miliseconds.

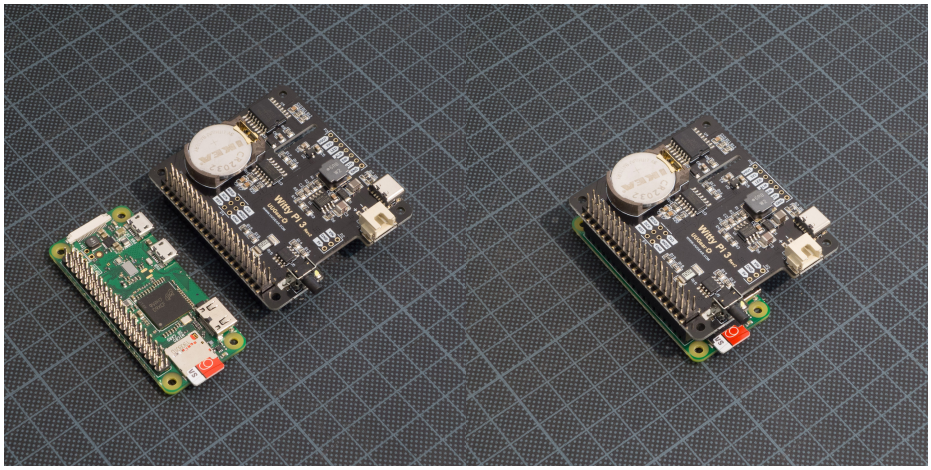


Fig. 4. Raspberry Pi Zero and WittyPi module separate (left) and assembled (right)

[9] http://www.uugear.com/product/witty-pi-3-realtime-clock-and-power-management-for-raspberry-pi/
[10] https://pypi.org/project/smbus2/

4.3 Precision-over-energy

From the current, voltage, and time measurements we calculated the energy consumption (as the product of the three).

As an experimental measure we introduce precision over energy (PoE):

$$\frac{\alpha * Precision}{\beta * Energy} = \frac{\alpha * Precision}{\beta * Avg(Voltage) * Avg(Current) * Avg(Time)}$$

as a way of expressing the classification quality in relation to the energy required to run the prediction. Given that precision - which is to be maximized by an ideal classifier - is defined in the numerator of the equation and energy - which is to be minimized by an optimally efficient classifier - is defined in the denumerator of the equation, it is obvious that increase of PoE corresponds to increase in classifier's efficiency and "intelligence".

Values for weighting parameters α and β specifying one's preference for precision or energetic efficiency were - for the purpose of didactic simplicity of this introductory study - both set to $\alpha = \beta = 1$.

5 Results

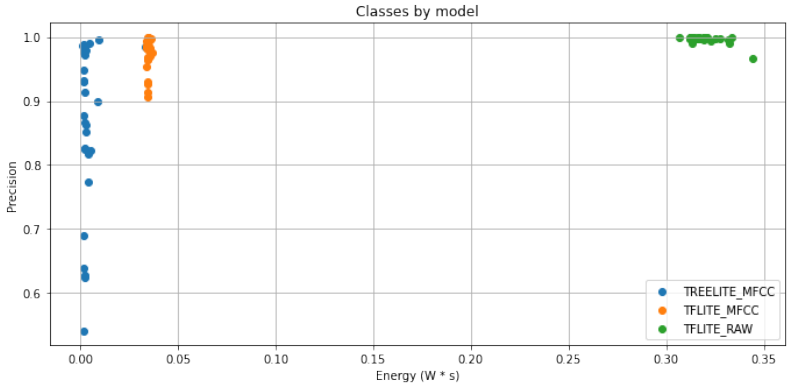
All tested models successfully ran on Pi Zero, with <1s mean processing time. As indicated by table 3, the TFLITE_RAW is able to correctly classify highest amount of testing samples (i.e. accuracy). However, this approach takes 8 times longer and thus necessitates significantly more energy than TREELITE_MFCC. TFLITE_MFCC outperforms TREELITE_MFCC in terms of precision while being about 4 times more energy efficient than TFLITE_RAW. While being the cleanest solution when it comes to time and energy, the Treelite approach comes at the cost of loading and keeping a 313MB model in RAM.

Table 3. Summary comparison of Random Forest Treelite and Tensorflow Lite speech command classification pipelines.

	TREELITE_MFCC	TFLITE_MFCC	TFLITE_RAW
Avg. time (s)	0.0912 (100%)	0.1727 (189%)	0.7898 (866%)
Avg. current (A)	0.0471 (100%)	0.0725 (154%)	0.0789 (167%)
Avg. voltage (V)	5.1368 (100%)	5.1301 (100%)	5.1249 (100%)
Avg. energy (W * s)	0.0221 (100%)	0.0643 (291%)	0.3192 (1446%)
Avg. micro-precision	0.9171 (100%)	0.9722 (106%)	0.9975 (109%)
Accuracy	0.6803 (100%)	0.8834 (130%)	0.9422 (138%)
AUC	0.8357 (100%)	0.8526 (102%)	0.9637 (115%)
PoE	41.5414 (100%)	15.1311 (36%)	3.1247 (8%)
Model size (MB)	313 (100%)	1.6 (0.5%)	3.4 (1%)

Values in parentheses are percentages of baseline (Treelite). Time, voltage and current information includes both feature extraction as well as class prediction phase.

Fig. 5. Energy to Micro-precision plot.



	Avg. time (s)	Avg. current (A)	Avg. voltage (V)	Avg. energy (W * s)	Avg. micro-precision	AUC	PoE
eight	0.178	0.036	5.145	0.033	0.985	0.937	29.413
sheila	0.058	0.030	5.147	0.009	0.996	0.972	109.654
nine	0.060	0.027	5.151	0.008	0.899	0.804	105.897
yes	0.034	0.026	5.151	0.005	0.991	0.944	211.344
one	0.041	0.026	5.151	0.005	0.822	0.729	151.188
no	0.029	0.026	5.151	0.004	0.773	0.737	193.789
left	0.025	0.028	5.150	0.004	0.817	0.798	220.264
tree	0.015	0.028	5.151	0.002	0.826	0.748	368.505
bed	0.020	0.028	5.150	0.003	0.862	0.777	296.923
bird	0.021	0.025	5.151	0.003	0.979	0.923	365.612
go	0.018	0.027	5.150	0.002	0.627	0.653	256.079
wow	0.012	0.027	5.150	0.002	0.932	0.887	535.011
seven	0.015	0.027	5.150	0.002	0.979	0.888	456.771
marvin	0.014	0.026	5.150	0.002	0.867	0.809	446.135
dog	0.013	0.027	5.146	0.002	0.541	0.690	306.897
three	0.011	0.026	5.148	0.001	0.877	0.790	614.417
two	0.012	0.027	5.148	0.002	0.930	0.862	589.297
house	0.014	0.026	5.150	0.002	0.989	0.958	532.962
down	0.014	0.026	5.148	0.002	0.689	0.655	372.188
six	0.009	0.026	5.150	0.001	0.986	0.918	770.749
five	0.012	0.026	5.149	0.002	0.638	0.704	379.180
off	0.010	0.027	5.147	0.001	0.948	0.881	662.200
right	0.017	0.030	5.146	0.003	0.852	0.781	324.926
cat	0.016	0.030	5.146	0.002	0.624	0.645	257.120
zero	0.013	0.029	5.149	0.002	0.980	0.909	526.845
four	0.014	0.029	5.147	0.002	0.972	0.913	457.840
stop	0.012	0.028	5.148	0.002	0.977	0.904	554.383
up	0.013	0.030	5.147	0.002	0.824	0.779	410.283
on	0.013	0.030	5.148	0.002	0.915	0.843	471.268
happy	0.013	0.029	5.147	0.002	0.987	0.943	512.609

Fig. 6. Observed values yielded by TREELITE_MFCC classifier (prediction phase only). Classes are sorted in order of chronological exposure to classifier, reason for higher prediction times of samples which were tested first (e.g. ”eight”, ”sheila” and ”nine”) is not fully understood yet.

	Avg. time (s)	Avg. current (A)	Avg. voltage (V)	Avg. energy (W * s)	Avg. micro-precision	AUC	PoE
eight	0.092	0.076	5.145	0.036	0.984	0.889	27.532
sheila	0.092	0.077	5.144	0.036	0.998	0.965	27.409
nine	0.092	0.078	5.143	0.037	0.976	0.812	26.613
yes	0.092	0.076	5.144	0.036	0.996	0.962	27.804
one	0.092	0.072	5.146	0.034	0.992	0.943	29.005
no	0.092	0.072	5.146	0.034	0.982	0.917	28.976
left	0.092	0.073	5.146	0.034	0.981	0.849	28.517
tree	0.092	0.072	5.147	0.034	0.915	0.740	26.802
bed	0.092	0.072	5.146	0.034	0.953	0.823	28.095
bird	0.092	0.074	5.146	0.035	0.969	0.857	27.724
go	0.092	0.073	5.146	0.034	0.931	0.790	27.140
wow	0.092	0.071	5.146	0.034	0.986	0.900	29.264
seven	0.092	0.072	5.146	0.034	0.994	0.918	29.295
marvin	0.092	0.073	5.145	0.034	0.984	0.882	28.646
dog	0.093	0.074	5.145	0.035	0.971	0.876	27.364
three	0.092	0.073	5.145	0.034	0.906	0.754	26.339
two	0.092	0.072	5.146	0.034	0.965	0.775	28.236
house	0.093	0.073	5.146	0.035	1.000	0.996	28.730
down	0.092	0.073	5.145	0.035	0.927	0.739	26.786
six	0.092	0.073	5.146	0.034	0.994	0.927	28.853
five	0.092	0.074	5.145	0.035	0.969	0.814	27.961
off	0.092	0.074	5.144	0.035	0.986	0.887	28.032
right	0.091	0.073	5.145	0.034	0.988	0.930	28.803
cat	0.092	0.074	5.145	0.035	0.984	0.887	28.233
zero	0.092	0.073	5.145	0.035	0.992	0.933	28.681
four	0.092	0.073	5.145	0.034	0.990	0.933	28.890
stop	0.092	0.074	5.145	0.035	0.993	0.941	28.461
up	0.092	0.073	5.145	0.035	0.981	0.854	28.354
on	0.092	0.073	5.146	0.034	0.965	0.804	27.973
happy	0.092	0.073	5.146	0.034	0.999	0.990	29.174

Fig. 7. Observed values yielded by TFLITE_MFCC classifier (prediction phase only).

	Avg. time (s)	Avg. current (A)	Avg. voltage (V)	Avg. energy (W * s)	Avg. micro-precision	AUC	PoE
eight	0.790	0.079	5.122	0.321	1.000	0.996	3.118
sheila	0.790	0.078	5.125	0.316	0.999	0.981	3.160
nine	0.789	0.078	5.124	0.317	0.999	0.972	3.151
yes	0.789	0.076	5.126	0.307	1.000	0.996	3.260
one	0.791	0.082	5.121	0.332	0.996	0.961	3.004
no	0.790	0.083	5.121	0.334	0.999	0.981	2.993
left	0.790	0.077	5.125	0.313	0.999	0.974	3.195
tree	0.790	0.077	5.125	0.313	0.990	0.912	3.161
bed	0.788	0.077	5.126	0.312	0.998	0.970	3.199
bird	0.791	0.080	5.125	0.323	0.994	0.925	3.081
go	0.789	0.077	5.127	0.312	0.998	0.971	3.200
wow	0.788	0.077	5.126	0.313	0.999	0.980	3.193
seven	0.791	0.077	5.126	0.313	1.000	0.995	3.196
marvin	0.790	0.078	5.125	0.315	1.000	1.000	3.180
dog	0.789	0.079	5.125	0.321	0.999	0.983	3.114
three	0.790	0.085	5.120	0.344	0.966	0.850	2.808
two	0.789	0.081	5.123	0.327	0.998	0.954	3.050
house	0.788	0.077	5.126	0.312	1.000	0.991	3.200
down	0.789	0.077	5.126	0.313	0.998	0.979	3.192
six	0.789	0.077	5.126	0.313	0.999	0.976	3.195
five	0.790	0.080	5.125	0.325	0.997	0.965	3.067
off	0.791	0.082	5.124	0.332	0.991	0.908	2.984
right	0.790	0.079	5.125	0.318	0.997	0.971	3.138
cat	0.791	0.079	5.125	0.319	0.999	0.989	3.129
zero	0.790	0.079	5.126	0.319	0.996	0.957	3.120
four	0.789	0.078	5.127	0.317	0.999	0.967	3.152
stop	0.790	0.078	5.127	0.317	1.000	0.987	3.155
up	0.790	0.079	5.127	0.319	0.999	0.975	3.131
on	0.791	0.078	5.127	0.316	0.999	0.977	3.165
happy	0.789	0.077	5.127	0.313	1.000	0.989	3.198

Fig. 8. Observed values yielded by TFLITE_RAW classifier (prediction phase only).

6 Conclusion

In a situation where it is clear that artificial systems, including AIs and ML classifiers account for an ever-growing amount of CO_2 emissions, every responsible scientist is obliged to keep the environmental impact of their work as low as possible.

One of the aspects to consider when developing ML models is how big the energy consumption is, not only for training but also continously using those models. Provided that established classifiers may well lead to execution of thousands or even millions of predictions on billions of individual devices, a reasonable choice of an appropriate classifier could potentially lead to palpable reduction in resulting carbon footprint.

As our experiments have shown, speech command classification is possible even on very low-powered devices such as the Raspberry Pi Zero. Furthermore, depending on which of the two needs to be prioritized – precision or lowest possible power consumption – there are different options to choose from and there are trade-offs to be done.

At last but not least, it may be the case that line of research analyzing and comparing different classifiers not only in terms of their classificatory score, but also in terms of other aspects physical and processual aspects like time or power consumption may yield many unexpected or even counter-intuitive surprises. For example, in spite of significant amount of effort on our side, it is still not completely clear to us why does TREELITE_MFCC tends to execute early classifications more slowly than later ones, a phenomenon which is marked - in Figure 5 - by increased average execution times of samples belonging to those classes (e.g. “eight”, “sheila”, “nine”) which were analyzed as first during the testing phase.

Be it as it may, we conclude that the execution of a domain-specific, 30-class classification of 1 second long speech commands is indeed feasible in real-time on a Raspberry Pi Zero / ARMv6 architecture with sufficient accuracy and precision. Thus, with a little bit of luck and a sufficient dose of engineering ingenuity, it can be, indeed, possible, to construct a speech-based, offline, Pi Zero-driven artefact like a Digital Primer.

References

1. Mel frequency cepstral coefficient (mfcc) tutorial, http://www.practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/
2. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: Tensorflow: A system for large-scale machine learning. In: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation. p. 265–283. OSDI'16, USENIX Association, USA (2016)
3. Beazley, D., Jones, B.: Python Cookbook: 3rd Edition. O'Reilly Media, Incorporated (2013)
4. Chen, G., Parada, C., Heigold, G.: Small-footprint keyword spotting using deep neural networks. In: 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 4087–4091 (2014). https://doi.org/10.1109/ICASSP.2014.6854370
5. Chen, L., Su, W., Feng, Y., Wu, M., She, J., Hirota, K.: Two-layer fuzzy multiple random forest for speech emotion recognition in human-robot interaction. Information Sciences 509, 150–163 (2020)
6. David, R., Duke, J., Jain, A., Reddi, V.J., Jeffries, N., Li, J., Kreeger, N., Nappier, I., Natraj, M., Regev, S., Rhodes, R., Wang, T., Warden, P.: Tensorflow lite micro: Embedded machine learning on tinyml systems (2021)
7. Davis, S., Mermelstein, P.: Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. IEEE transactions on acoustics, speech, and signal processing 28(4), 357–366 (1980)
8. Gruenstein, A., Alvarez, R., Thornton, C., Ghodrat, M.: A cascade architecture for keyword spotting on mobile devices (2017)
9. Ho, T.K.: Random decision forests. In: Proceedings of 3rd international conference on document analysis and recognition. vol. 1, pp. 278–282. IEEE (1995)
10. Hromada, D.D., Seidler, P., Kapanadze, N.: Bauanleitung einer digitalen fibel von und für ihre schüler. In: Mobil mit Informatik: 9. Münsteraner Workshop zur Schulinformatik. p. 37. BoD–Books on Demand (2020)
11. Hromada, D.D.: After smartphone: Towards a new digital education artefact. Enfance (3), 345–356 (2019)
12. Mattson, P., Reddi, V.J., Cheng, C., Coleman, C., Diamos, G., Kanter, D., Micikevicius, P., Patterson, D., Schmuelling, G., Tang, H., Wei, G., Wu, C.: Mlperf: An industry standard benchmark suite for machine learning performance. IEEE Micro 40(2), 8–16 (2020). https://doi.org/10.1109/MM.2020.2974843
13. Phan, H., Maaß, M., Mazur, R., Mertins, A.: Random regression forests for acoustic event detection and classification. IEEE/ACM Transactions on Audio, Speech, and Language Processing 23(1), 20–31 (2014)
14. Piczak, K.J.: Esc: Dataset for environmental sound classification (2015)
15. Reddi, V.J., Cheng, C., Kanter, D., Mattson, P., Schmuelling, G., Wu, C., Anderson, B., Breughe, M., Charlebois, M., Chou, W., Chukka, R., Coleman, C., Davis, S., Deng, P., Diamos, G., Duke, J., Fick, D., Gardner, J.S., Hubara, I., Idgunji, S., Jablin, T.B., Jiao, J., John, T.S., Kanwar, P., Lee, D., Liao, J., Lokhmotov, A., Massa, F., Meng, P., Micikevicius, P., Osborne, C., Pekhimenko, G., Rajan, A.T.R., Sequeira, D., Sirasao, A., Sun, F., Tang, H., Thomson, M., Wei, F., Wu, E., Xu, L., Yamada, K., Yu, B., Yuan, G., Zhong, A., Zhang, P., Zhou, Y.: Mlperf inference benchmark. CoRR abs/1911.02549 (2019), http://arxiv.org/abs/1911.02549

16. Warden, P.: Launching the speech commands dataset https://ai.googleblog.com/2017/08/launching-speech-commands-dataset.html
17. Warden, P.: Speech commands: A dataset for limited-vocabulary speech recognition (2018)
18. Xue, J., Zhao, Y.: Random forests of phonetic decision trees for acoustic modeling in conversational speech recognition. IEEE transactions on audio, speech, and language processing 16(3), 519–528 (2008)