Evaluating the practical limitations of TinyML: an experimental approach

Giovanni Delnevo
Department of Computer Science and Engineering
University of Bologna
Bologna, Italy
giovanni.delnevo2@unibo.it

Catia Prandi
Department of Computer Science and Engineering
University of Bologna
Bologna, Italy
catia.prandi2@unibo.it

Silvia Mirri
Department of Computer Science and Engineering
University of Bologna
Bologna, Italy
silvia.mirri@unibo.it

Pietro Manzoni
Department of Computer Engineering
Universitat Politècnica de València
València, Spain
pmanzoni@disca.upv.es

2021 IEEE Globecom Workshops (GC Wkshps) | 978-1-6654-2390-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/GCWkshps52748.2021.9682101

***Abstract*—Tiny Machine Learning (TinyML) is a novel research field that opens up the possibility of embedding local intelligence into frugal objects thus creating new opportunities for building "networks of collective intelligence". Energy consumption reduction is probably the main reason why TinyML deserves attention in terms of sustainability, but it is not the only one. The low costs of the hardware and the possibility to increase the level of data security and user privacy are outstanding reasons as well. In this work, we present the results of a sensitivity analysis we have conducted to evaluate the performance of a Random Forest with data collected by a state-of-the-art hardware device. We focused on assessing the sensitivity of the detection of sounds, colors, and vibrations patterns. Results show that TinyML can be absolutely used to properly discriminate among several ranges of sounds, colors, and vibrations patterns, paving the way for the development of new promising sustainable applications.**

***Index Terms*—TinyML, sustainability, Random Forest, sensitivity analysis**

I. Introduction

Tiny Machine Learning (TinyML) is a novel research field that aims to bring machine learning techniques within the reach of very small devices and embedded systems applications. In particular, in the context of the work that we are here presenting, we are considering low cost devices (MicroController Units - MCUs) that can work at 1 mW of power [1] integrating a basic CPU, a reduced amount of memory, and some basic connectivity alternatives. Combined with Edge computing [2], TinyML allows computation to be spread along the path from the end-device to the core of the Internet, thus reducing the need of the more energy hungry data-centers that are typically used for Machine Learning applications processing in cloud platforms [3], [4].

Energy consumption reduction is probably the main reason why TinyML deserves attention in terms of sustainability, even if it is not the only one. In fact, the low costs of the hardware is a critical aspect when deploying a large number of devices, too. Moreover, running on-device ML inference increases the level of data security and user privacy [5]. Data can be sensitive, and processing and analyzing them on-device do not require sending them over the Internet. Finally, computing at the edge reduces the network bandwidth requirements, allowing the use of energy-efficient communication technologies [6].

TinyML presents unique opportunities also for frugal objects [7], which have penetrated our daily lives thanks to the rapid emergence of the IoT (Internet of Things). The possibility of embedding local intelligence into frugal objects opens up new opportunities for building "networks of collective intelligence". The frugal innovation consists of a creative and problem-oriented approach to problem-solving which starts from user needs and works from the bottom up to develop contextually appropriate solutions [8]. Such a process is strictly connected to sustainability, as highlighted by Khan in [9]. According to the findings reported in that paper, frugal innovation can be a driving factors, not only in achieving social sustainability, but also in promoting the Sustainable Development Goals. For this reason, integrating TinyML within frugal objects can significantly enrich the current applications landscape of frugal objects.

Sanchez-Iborra and Skarmeta summarized five possible application contexts that could benefit from the integration of TinyML in frugal objects [7]. The first one is eHealth, in which smart end-devices could be employed for health monitoring, visual assistance, hearing aids, and personal sensing, just to cite a few [10]. The second consists of smart spaces (e.g., smart cities and cognitive buildings) where such systems could be easily deployed to make decentralized and quick decisions [11]. The third application field is vehicular services. In particular, Cooperative—Intelligent Transportation Systems could also be employed on personal mobility devices like shared bikes, electric mopeds, and scooters [12]. Another application context is Industry 4.0 where TinyML could allow to add some form of intelligence along the production chain to refine the

production processes and to track the assets [13]. Finally, the last sectors are smart agriculture and farming, where TinyML could improve the smart monitoring and control processes with the purpose of increasing the efficiency and health of crops and animals [14].

In this paper, we present a sensitivity analysis of the practical limitations of TinyML using an experimental approach. By means of a commonly used device, an "Arduino Nano 33 BLE sense", which supports a series of embedded sensors, we evaluated the level of precision that can be obtained when detecting sounds, colors, and vibrations patterns. The objective is better define the actual class of application that can be designed with the current state of the art of hardware combined with TinyML. Our methodology was based on collecting the data-set directly using the Arduino Nano sensors. Then, using these data-sets, we trained a machine learning algorithm to classify certain property of the measured variable. Finally, we tested its detection performance directly on the device. Results show that, using a Random Forest method, it is possible to discriminate among different ranges of frequencies, colors, and vibrations patterns and intensities.

To the best of our knowledge, no work in the literature has dealt with the sensitivity analysis of the combination of the microcontroller sensors with machine learning algorithms that perform the inference directly on these devices. Along with the release of several software stacks like Tensorflow lite micro [15], Edge2train [16], OpenNN [17], Edge Machine Learning library [18], and Resource Constrained Edge-Neural Networks [19], different benchmarks have been proposed. Among them, we can cite the MLPerf Tiny Benchmark [20], CoreMark [21], and TinyML Benchmark [22]. Anyway, the purpose of such benchmarks is to measure some aspects in the inference process of machine learning algorithms such as accuracy, latency, and energy consumption.

The reminder of the paper is organized as follows. Section II details each step of our approach, describing how we conducted the data collection, the model training, and its evaluation. Then, in Section III, we describe the results of the analysis on the sensitivity of sensors in detecting the frequency, the color, and the vibrations. Finally, Section IV closes the paper, highlighting some future works.

II. Methodology

In this Section, we detail our approach, composed of three main steps: data collection, model training, and model evaluation. We followed it for each aspect analyzed: sounds, colors, and vibrations patterns. Each phase will be detailed separately in the following Subsections.

A. Data and Collection

The data collection process has been carried out employing the different sensors mounted on the Arduino Nano 33 BLE. To generate the frequencies, we employed the Android mobile app *Frequency Generator*[1]. It allows to generate frequencies from 1Hz (infrasound) to 22,000Hz (ultrasound), using single or multi-frequencies with different sound waves (sinusoidal, square, triangle, and sawtooth). To record such frequencies, we employed the digital microphone MP34DT05. It records 256 readings continuously with a sampling rate of 16KHz. A pulse-density modulation is applied to obtain 128 values, of which the RMS (Root Mean Square) is computed. A threshold is applied to record only when there is sound and thus avoid silence or background noise. Hence, if the RMS is higher than a fixed threshold, the recording process starts. It simply consists of getting 32 RMS values, with a delay of 20ms between each sample, for a total period of 640ms. This 32-values array is used as a representation of the sounds recorded. For each frequency considered, we collected 100 samples.

With regard to the colors, we considered some slightly different color shades both printed on paper and visualized on a screen. We employed the APDS-9960 sensor, a digital proximity and ambient light sensor able to retrieve also the color. It returns the three RGB components, plus a value regarding the ambient light intensity. To these four, we have also added proximity, since it could significantly affect the color. Hence, each sample consists of five integers. For each color, we collected 200 samples.

Finally, to generate the vibrations, we took advantage of the Android mobile app *Vibrate Pattern Maker*[2]. It allows to generate patterns by alternating periods of vibration and "silence". For each period, it is possible to specify its duration in milliseconds but it is not possible to vary the intensity of the vibrations. Such patterns can be repeated indefinitely. We generated different patterns varying vibrations and silence periods and we tried to discriminate among them. Then, we also tried to discriminate among patterns of vibrations of different intensities. We employed a vibrations motor with five different speed settings, which we used to generate simple vibration patterns only varying the intensity of vibrations. To detect vibrations, we used the LSM9DS1 sensor that provides an acceleration sensor. The sensor returns the accelerations in the three dimensions x, y, and z. We collected 20 samples, with a delay of 50ms between each sample. This 60-values (20 samples of three values) array is used as a representation of the vibration pattern recorded and covers 1s of data.

It is important to notice that the data are collected under *laboratory conditions*. Real applications will pose problems and challenges. But our aim, here, is just to conduct a sensitivity analysis of the practical limitations of TinyML using an experimental approach

B. Model Training

Once enough data are collected, we move on to the training of the machine learning model. The training phase has been conducted on a general-purpose PC and not on the Arduino Nano. We used the Python language and we took advantage of the Scikit-learn library [23].

We used Random Forests in all the experiments. A Random Forest is a meta estimator that combines the prediction of

[1]play.google.com/store/apps/details?id=com.boedec.hoel.frequencygenerator

[2]play.google.com/store/apps/details?id=com.domago.vibratepatternmaker

several decision trees on different sub-samples of the training set with the aim of improving the prediction accuracy and prevent overfitting. The models, that we employed in the experiments, combined 50 decision trees. The information gain was evaluated by using the Gini impurity, and we did not set a maximum depth of the tree. All the other parameters, such as the minimum number of samples required to split an internal node and the minimum number of samples required to be at a leaf node, are left with their default values.

We decided to use Random Forests for several reasons [24]. First, we preferred them to neural networks since features are already available and there is no need to extract/discover them and given the numerosity of the data. Second, with respect to other ensemble methods, such as gradient boosting and xgboost, we chose only one methods that has proven effective in real scenarios [25], leaving the comparison with other methods as a future work.

In each experiment, the dataset is divided into two parts. The 80% of the samples are used for the training while the remaining 20% are used for validation. Data are divided in a stratified way with the aim of maintaining the same proportion for each class. It is important to remind that we did not employ a test set since we evaluated the accuracy of the model directly on the Arduino Nano. Furthermore, we did not conduct any tuning phase on the hyper-parameters of the model. The main reason is that usually this phase allows in the most fortunate cases to improve the accuracy of a few percentage points. Since we are interested in understanding the overall sensitivity of machine learning models deployed on the Arduino Nano, we have decided not to carry it out.

C. Model Evaluation

Once the models were trained, we evaluated their performance. The choice of evaluating the model directly on the Arduino Nano derives from the fact that we do not want to find the maximum possible accuracy using the data it detects through its sensors. Instead, in a TinyML perspective, we want to evaluate the maximum accuracy achievable by a machine learning model running on it. This is because having the Arduino Nano limited computational resources, it is possible that the performance on it differs from that obtained on a more powerful device such as a PC. To import the trained models on the Arduino Nano, we exploited the EloquentTinyML library[3]. It is a library that simplifies the deployment of some Tensorflow Lite and Scikit-learn models, such as neural networks, random forests, and extreme gradient boosting. Once deployed, we used the models to predict the class of several inputs and, at the end of the process, we evaluated its performance using several metrics. In particular, we took advantage of precision, recall, f1-score, and accuracy. We did not report the accuracy on the test set computed on a general-purpose PC, since the results were very similar, with a maximum difference of 1%.

[3]github.com/eloquentarduino/EloquentTinyML

III. Results

In this Section, we present the results of the experiments conducted to evaluate the sensitivity of a Random Forest in detecting the frequency, color, and vibrations, measured by the Arduino Nano sensors.

A. Frequency Detection Sensitivity

We started the experiments to understand the sensitivity of the microphone by evaluating frequencies from 500Hz to 3,000Hz, varying them by 500Hz. The deployed model was tested by collecting 20 examples for each frequency. The results are reported in Table I. As shown, the model is able to almost perfectly differentiate the analyzed frequencies.

TABLE I
PERFORMANCE OF THE CLASSIFIER IN DETECTING FREQUENCIES BY VARYING THEM OF 500HZ, FROM 500HZ TO 3,000HZ

Class	Precision	Recall	F1-score	Accuracy
500	1.00	1.00	1.00	0.99
1000	1.00	1.00	1.00	
1500	1.00	1.00	1.00	
2000	1.00	1.00	1.00	
2500	1.00	0.95	0.97	
3000	0.95	1.00	0.98	

Then, we decided to focus on the 2,000Hz to 3,000Hz frequency range and to reduce the step from 500Hz to 200Hz. As in the previous experiment, we evaluated the deployed model on a test set containing 20 samples for each class. Table II illustrates the performance of the classifier. Its overall accuracy, that passes from 0.99 to 0.78, is still good, even if it has some problems in distinguishing 2,200Hz from 2,400Hz and 2,800Hz from 3,000 while it perfectly recognizes 2,000Hz and 2,600Hz. This is also visible in the confusion matrix, that we depicted in Figure 1.

TABLE II
PERFORMANCE OF THE CLASSIFIER IN DETECTING FREQUENCIES BY VARYING THEM OF 200HZ, FROM 2,000HZ TO 3,000HZ

Class	Precision	Recall	F1-score	Accuracy
2000	1.00	1.00	1.00	0.78
2200	0.74	0.70	0.72	
2400	0.71	0.75	0.73	
2600	1.00	1.00	1.00	
2800	0.58	0.90	0.71	
3000	0.78	0.35	0.48	

Finally, we further reduced the frequency variation from 200Hz to 100Hz. In parallel with the previous tests, in this one too we used 20 examples for each class to evaluate the classifier. In this case, as highlighted by the results in Table III, the model is no longer able to distinguish between the different frequencies. It is interesting to notice that it has excellent performance on the classes 2,000Hz and 2,100Hz. In all the other cases, they considerably get worse such as in the case of the frequencies 2,200Hz and 2,400Hz, that has the worst F1-scores among all, that are respectively 0.36 and 0.37. As visible in Figure 2, in addition to not distinguishing well the examples between these two classes, as was the case in

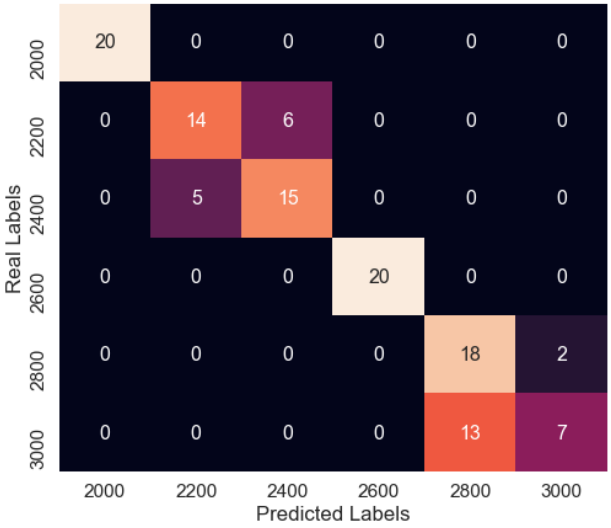

Fig. 1. Confusion matrix obtained varying the frequency of 200Hz, from 2,000Hz to 3,000Hz

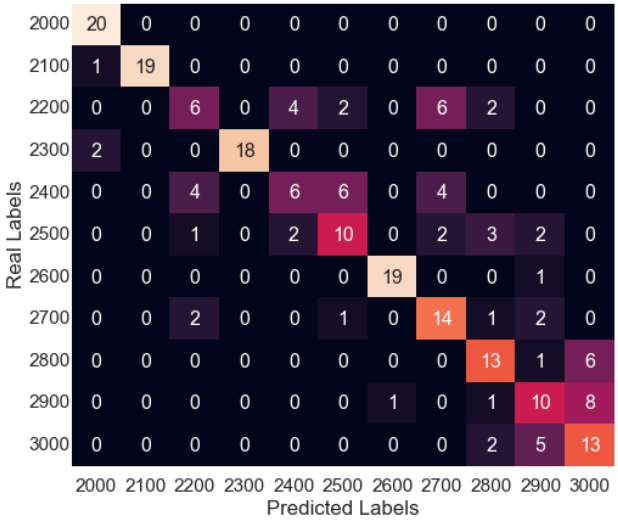

Fig. 2. Confusion matrix obtained varying the frequency of 100Hz, from 2,000Hz to 3,000Hz

the previous experiment, now their examples are also confused with those of classes 2,500Hz and 2,700Hz.

TABLE III
PERFORMANCE OF THE CLASSIFIER IN DETECTING FREQUENCIES BY VARYING THEM OF 100HZ, FROM 2,000HZ TO 3,000HZ

Class	Precision	Recall	F1-score	Accuracy
2000	0.87	1.00	0.93	
2100	1.00	0.95	0.97	
2200	0.46	0.30	0.36	
2300	1.00	0.90	0.95	
2400	0.50	0.30	0.37	
2500	0.53	0.50	0.51	0.67
2600	0.95	0.95	0.95	
2700	0.54	0.70	0.61	
2800	0.59	0.65	0.62	
2900	0.48	0.50	0.49	
3000	0.48	0.65	0.55	

B. Color Detection Sensitivity

To evaluate the sensitivity of the color sensor, we decided to employ seven different, but very similar, shades of green. Their hexadecimal RGB values are the following ones: #229658, #30A161, #3DAB6B, #49B675, #55C17F, #60CC89, and #6BD793. We depicted some squares filled with these colors in Figure 3.

We began our experiments by visualizing the squares on a monitor. In these cases, the values returned by the sensor are very precise and there is no need of using machine learning algorithms to distinguish among them. In fact, a simple set of conditional statements is enough to achieve 100% of accuracy. The situation becomes more complicated considering the colors printed on paper. In this scenario, the quality of the printer and the absence of backlighting, which is present in the monitor, certainly have a significant influence. The results of the model using all the shades of green are reported in Table IV. Even if the different shades are very similar, the system

Fig. 3. Green color scale used in the experiments.

can differentiate among them with acceptable accuracy. The F1-score on the different classes varies significantly, passing from the 0.93 of the #6BD793 until reaching the 0.63 of the #30A161.

We also conducted an additional test to understand if the performance improves using less similar colors. We simply kept only three colors. The extremes of the green scale, #229658 and #6BD793, and the one in between #49B675. In such a case, the accuracy of the model rises to 99%. The model commits only one error, misclassifying one sample of #49B675 as #6BD793.

TABLE IV
PERFORMANCE OF THE CLASSIFIER IN DETECTING COLORS

Class	Precision	Recall	F1-score	Accuracy
#229658	0.78	0.70	0.74	
#30A161	0.63	0.62	0.63	
#3DAB6B	0.72	0.84	0.78	
#49B675	0.88	0.88	0.88	0.79
#55C17F	0.80	0.72	0.76	
#60CC89	0.75	0.84	0.79	
#6BD793	0.96	0.90	0.93	

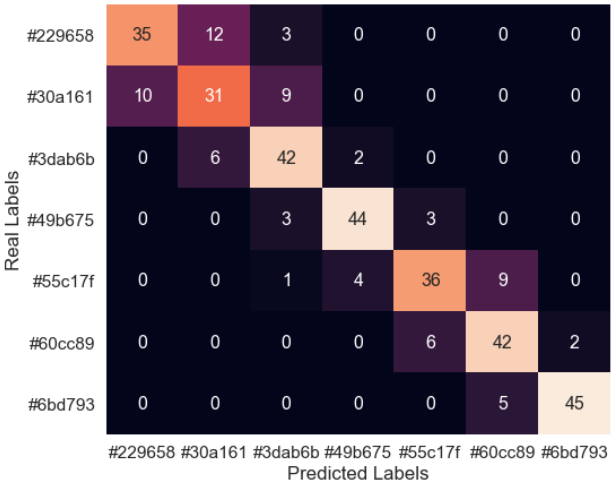

Fig. 4. Confusion matrix obtained when detecting colors

TABLE V
PERFORMANCE OF THE CLASSIFIER IN DETECTING VIBRATION PATTERNS OF 100MS, 150MS, AND 200MS

Class	Precision	Recall	F1-score	Accuracy
100ms - 100ms	1	1	1	1
150ms - 150ms	1	1	1	
200ms - 200ms	1	1	1	

C. Vibration Detection Sensitivity

Finally, we conducted some experiments to evaluate the sensitivity of the model deployed on the Arduino Nano in detecting vibration. We began trying to recognize three different patterns in which the vibrations last like the pause period, respectively 100ms, 150ms, and 200ms. The model is able to perfectly discriminate among the different patterns, as witnessed by the metrics reported in Table V.

Then, we added three further patterns which last respectively 10ms, 25ms, and 50ms. Results don't change, with the model that is still able to perfectly discriminate among the different patterns. The results are reported in Table VI.

TABLE VI
PERFORMANCE OF THE CLASSIFIER IN DETECTING VIBRATION PATTERNS OF 10MS, 25MS, 50MS, 100MS, 150MS, AND 200MS

Class	Precision	Recall	F1-score	Accuracy
10ms -10ms	1	1	1	1
25ms - 25ms	1	1	1	
50ms - 50ms	1	1	1	
100ms - 100ms	1	1	1	
150ms - 150ms	1	1	1	
200ms - 200ms	1	1	1	

TABLE VII
PERFORMANCE OF THE CLASSIFIER IN DETECTING VIBRATION PATTERNS OF 25MS AND 50MS WITH PAUSES OF 50MS AND 25MS

Class	Precision	Recall	F1-score	Accuracy
25ms - 50ms	1	1	1	1
50ms - 25ms	1	1	1	

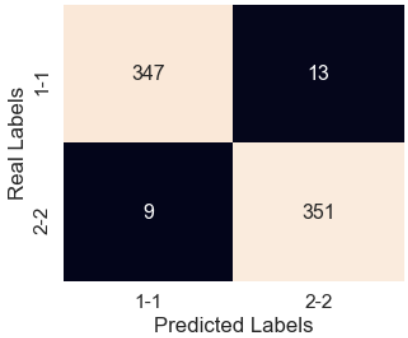

Fig. 5. Confusion matrix obtained when detecting vibrations pattern of 1ms and 2ms

TABLE VIII
PERFORMANCE OF THE CLASSIFIER IN DETECTING VIBRATION PATTERNS OF 1MS AND 2MS

Class	Precision	Recall	F1-score	Accuracy
1ms - 1ms	0.97	0.96	0.97	0.97
2ms - 2ms	0.96	0.97	0.97	

We also evaluated two patterns with different duration of vibrations and pauses. The first one composed of vibrations of 25ms followed by pauses of 50ms. In the second, instead, vibrations last 50ms while pauses last 25ms. Again, the model perfectly detects the two patterns, as shown in Table VII.

In the last test carried out, we took the patterns to extremes with two patterns of equal vibration and pause periods of 1ms and 2ms respectively. Anyway, the performances remain excellent, as reported in Table VIII, with only 22 examples misclassified, as shown in Figure 5.

With regards to detecting different vibrations intensities, we simply tried to recognize the five-speed settings of the vibration motors. Even in this case, we got excellent performance, with the classifier that reaches 99% of accuracy, as shown in Table IXand Figure 6.

TABLE IX
PERFORMANCE OF THE CLASSIFIER IN DETECTING COLORS

Class	Precision	Recall	F1-score	Accuracy
1	1.00	0.99	1.00	0.99
2	0.99	1.00	1.00	
3	1.00	0.99	1.00	
4	0.99	0.97	0.98	
5	0.97	0.99	0.98	

IV. CONCLUSION AND FUTURE WORKS

TinyML is gaining growing attention in both academia and industry, due to the advantages related to running machine learning inference on low-power devices. Its possible applications in different contexts are numerous, and, indeed, sustainable environmental sensing is one of the most promising. In this work, we evaluated the sensitivity of a Random Forest trained with data collected by the Arduino Nano 33 BLE sensors. We experimented on the detection of sounds, colors, and vibrations pattern. Results show that TinyML can be

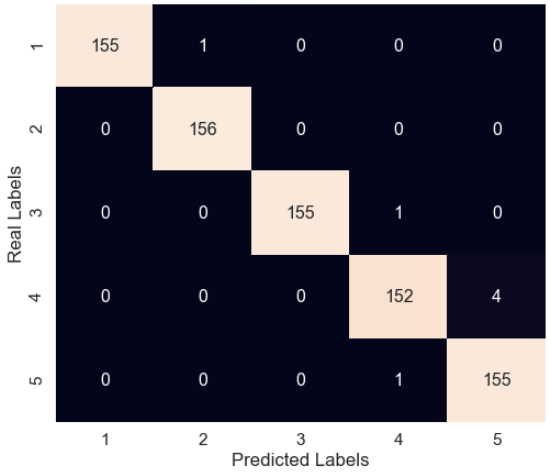

Fig. 6. Confusion matrix obtained when detecting vibrations generated by the 5 speed settings of the immersion blender

absolutely used to properly discriminate among several ranges of sounds, colors, and vibrations pattern, paving the way for the development of new promising sustainable applications. Even if it is important to remind that data used in the experiments were collected under laboratory conditions.

Future works are manifold. First, a multi-dimensional trade-offs analysis of other aspects, such as energy consumption, inference rate, and memory occupancy rates, can be conducted together with an analysis of the mathematical differences between the learning algorithms of both tiny and non-tiny implementations. Then, a comparison with other ensembling strategies could be carried out, as in [26].

ACKNOWLEDGMENTS

This work was partially supported by the "Conselleria de Educación, Investigación, Cultura y Deporte, Direcció General de Ciéncia i Investigació, Proyectos AICO/2020", Spain, under Grant AICO/2020/302.

REFERENCES

[1] P. Warden and D. Situnayake, *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. O'Reilly Media, 2019.

[2] K. Nakamura, P. Manzoni, M. Zennaro, J.-C. Cano, C. T. Calafate, and J. M. Cecilia, "Fudge: A frugal edge node for advanced iot solutions in contexts with limited resources," in *Proceedings of the 1st Workshop on Experiences with the Design and Implementation of Frugal Smart Objects*, (New York, NY, USA), p. 30–35, Association for Computing Machinery, 2020.

[3] C. Banbury, C. Zhou, I. Fedorov, R. Matas, U. Thakker, D. Gope, V. Janapa Reddi, M. Mattina, and P. Whatmough, "Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers," *Proceedings of Machine Learning and Systems*, vol. 3, 2021.

[4] H. Doyu, R. Morabito, and M. Brachmann, "A tinymlaas ecosystem for machine learning in iot: Overview and research challenges," in *2021 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pp. 1–5, IEEE, 2021.

[5] M. S. Islam, H. Verma, L. Khan, and M. Kantarcioglu, "Secure real-time heterogeneous iot data management system," in *2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pp. 228–235, IEEE, 2019.

[6] W. Mao, Z. Zhao, Z. Chang, G. Min, and W. Gao, "Energy efficient industrial internet of things: Overview and open issues," *IEEE Transactions on Industrial Informatics*, 2021.

[7] R. Sanchez-Iborra and A. F. Skarmeta, "Tinyml-enabled frugal smart objects: Challenges and opportunities," *IEEE Circuits and Systems Magazine*, vol. 20, no. 3, pp. 4–18, 2020.

[8] H. Kroll, M. Gabriel, A. Braun, F. Engasser, M. Meister, and E. Muller, "Study on frugal innovation and reengineering of traditional techniques," 2015.

[9] R. Khan, "How frugal innovation promotes social sustainability," *Sustainability*, vol. 8, no. 10, p. 1034, 2016.

[10] P. Randhawa, V. Shanthagiri, and A. Kumar, "A review on applied machine learning in wearable technology and its applications," in *2017 International Conference on Intelligent Sustainable Systems (ICISS)*, pp. 347–354, IEEE, 2017.

[11] M. Bortoli, M. Furini, S. Mirri, M. Montangero, and C. Prandi, "Conversational interfaces for a smart campus: A case study," in *Proceedings of the International Conference on Advanced Visual Interfaces*, pp. 1–5, 2020.

[12] J. Santa and P. J. Fernández, "Seamless ipv6 connectivity for two-wheelers," *Pervasive and Mobile Computing*, vol. 42, pp. 526–541, 2017.

[13] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0," *IEEE industrial electronics magazine*, vol. 11, no. 1, pp. 17–27, 2017.

[14] P. P. Jayaraman, A. Yavari, D. Georgakopoulos, A. Morshed, and A. Zaslavsky, "Internet of things platform for smart farming: Experiences and lessons learnt," *Sensors*, vol. 16, no. 11, p. 1884, 2016.

[15] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, S. Regev, *et al.*, "Tensorflow lite micro: Embedded machine learning on tinyml systems," *arXiv preprint arXiv:2010.08678*, 2020.

[16] B. Sudharsan, J. G. Breslin, and M. I. Ali, "Edge2train: A framework to train machine learning models (svms) on resource-constrained iot edge devices," in *Proceedings of the 10th International Conference on the Internet of Things*, pp. 1–8, 2020.

[17] "Opennn." https://www.opennn.net/. Accessed: 2021-07-08.

[18] Dennis, Don Kurian and Gaurkar, Yash and Gopinath, Sridhar and Goyal, Sachin and Gupta, Chirag and Jain, Moksh and Jaiswal, Shikhar and Kumar, Ashish and Kusupati, Aditya and Lovett, Chris and Patil, Shishir G and Saha, Oindrila and Simhadri, Harsha Vardhan, "EdgeML: Machine Learning for resource-constrained edge devices."

[19] B. Sudharsan, J. G. Breslin, and M. I. Ali, "Rce-nn: a five-stage pipeline to execute neural networks (cnns) on resource-constrained iot edge devices," in *Proceedings of the 10th International Conference on the Internet of Things*, pp. 1–8, 2020.

[20] C. Banbury, V. J. Reddi, P. Torelli, J. Holleman, N. Jeffries, C. Kiraly, P. Montino, D. Kanter, S. Ahmed, D. Pau, *et al.*, "Mlperf tiny benchmark," *arXiv preprint arXiv:2106.07597*, 2021.

[21] S. Gal-On and M. Levy, "Exploring coremark a benchmark maximizing simplicity and efficacy," *The Embedded Microprocessor Benchmark Consortium*, 2012.

[22] B. Sudharsan, S. Salerno, D.-D. Nguyen, M. Yahya, A. Wahid, P. Yadav, J. G. Breslin, and M. I. Ali, "Tinyml benchmark: Executing fully connected neural networks on commodity microcontrollers," in *IEEE 7th World Forum on Internet of Things (WF-IoT), New Orleans, Louisiana, USA*, 2021.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[24] P. Probst, M. N. Wright, and A.-L. Boulesteix, "Hyperparameters and tuning strategies for random forest," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 9, no. 3, p. e1301, 2019.

[25] A. Verikas, A. Gelzinis, and M. Bacauskiene, "Mining data with random forests: A survey and results of new tests," *Pattern recognition*, vol. 44, no. 2, pp. 330–349, 2011.

[26] S. González, S. García, J. Del Ser, L. Rokach, and F. Herrera, "A practical tutorial on bagging and boosting based ensembles for machine learning: Algorithms, software tools, performance study, practical perspectives and opportunities," *Information Fusion*, vol. 64, pp. 205–237, 2020.