Internet of Things 22 (2023) 100729

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot

Research article

An evaluation methodology to determine the actual limitations of a TinyML-based solution

Giovanni Delnevo [a,*], Silvia Mirri [a], Catia Prandi [a], Pietro Manzoni [b]

[a] *Department of Computer Science and Engineering, University of Bologna, Via Dell'Università, Cesena, 47521, Italy*
[b] *Department of Computer Engineering, Universitat Politècnica de València, Camino de Vera, València, 46022, Spain*

A R T I C L E I N F O

MSC:
68T01

Keywords:
TinyML
Machine learning algorithms
Evaluation methodology
Actual limitations

A B S T R A C T

Tiny Machine Learning (TinyML) is an expanding research area based on pushing intelligence to the edge and bringing machine learning techniques to very small devices and embedded systems applications. TinyML reduces energy expenditure, uses low bandwidth communications technologies, and adds more privacy to the developed applications. This work, proposes an evaluation methodology to determine the limitations of a TinyML-based solution starting from creating and preparing the required dataset. Then, the training of the selected machine learning algorithms is detailed, together with the consequent evaluation, and how the experiments must be structured. Four metrics were usedto evaluate the performance of the machine learning algorithms in the various tasks: precision, recall, f1-score, and accuracy. Finally, a comparison ofthe performance of a wide range of machine learning algorithms (i.e., Random Forest, Decision Tree, Support Vector Classifier, Logistic Regression, Gaussian Naive Bayes, and Multi-Layer Perceptron) is presented.

1. Introduction

Tiny Machine Learning (TinyML) is an expanding research area based on pushing intelligence to the edge and bringing machine learning techniques to very small devices and embedded systems applications. The early definition by Warden and Situnayake [1] refers to the use of MicroController Units (MCU) that can work at 1 mW of power, provided with a reduced amount of memory and some basic connectivity alternatives. Computation is less energetically costly than communication, and preprocessing of the data locally allows sending, only when necessary, some summary data. This increases the duration of batteries that power IoT devices and allows the use of low-power, low-bandwidth communication technologies, such as those belonging to the LPWAN family (low-power wide area networks) [2]. Moreover, running on-device ML inference increases the level of data security and user privacy [3]. Data can be sensitive, and processing and analyzing them on-device do not require sending them over the Internet.

There are various examples of how TinyML is used in IoT contexts. In [4], the authors developed an IoT system based on Micro-Electro-Mechanical System (MEMS) sensor that collects acceleration data for machine learning training and, using a neural network model can recognize the running state of rail vehicles. In [5], Diab et al. focus on health and care applications and analyze the various challenges and specifications trade-offs associated with the existing hardware options and the newly developed software tools when using MCUs as inference devices. In this endeavour, they consider TinyML as an enabling technology. Zaidi et al. [6] go one step further and present TMLaaS an architecture based on the idea of ML-as-a-Service (MLaaS) for future IoT deployments. They describe how a TMLaaS architecture, for example based on the TinyML approach, can be implemented, deployed, and maintained

[*] Corresponding author.
E-mail addresses: giovanni.delnevo2@unibo.it (G. Delnevo), silvia.mirri@unibo.it (S. Mirri), catia.prandi2@unibo.it (C. Prandi), pmanzoni@disca.upv.es (P. Manzoni).

https://doi.org/10.1016/j.iot.2023.100729
Received 14 December 2022; Received in revised form 14 February 2023; Accepted 14 February 2023
Available online 22 February 2023

for large-scale IoT deployment, demonstrating its feasibility. More related to the computation part, in [7], the authors present a hardware–software framework to accelerate machine learning inference on edge devices using a modified TensorFlow Lite for Microcontroller. They base their proposal on using a dedicated Neural Processing Unit (NPU) custom hardware accelerator, referred to as MCU-NPU. Also, Kocacinar et al. [8] propose a lightweight real-time model as TinyML to evaluate the use of deep learning approaches as a computational model. They compared the overall learning performance on small data and, using only full facial images, proposed a model that allows only recognition from the eyes.

The examples above show that this area is very active, and the range of applications is broad. Therefore, we considered it necessary to propose an evaluation methodology to determine the limitations of a TinyML-based solution.

We based our methodology on a previous work [9] where, using a widely adopted device, an “Arduino Nano 33 BLE sense”, we evaluated the level of precision that can be obtained when detecting sounds, colors, and vibrations patterns. In this paper, we extend that previous work by considering more tasks, namely keyword spotting and hand-gesture recognition, we compare more ML algorithms, and we extend the tuning phase for each algorithm.

In the proposed methodology, it is described how to collect the required dataset to conduct a sensitivity analysis of the practical limitations of the TinyML solution being designed, employing an experimental approach. Data are collected under *laboratory conditions*; for each task class, one-hundred samples are collected for training and validation, while twenty ones are collected for evaluation. After completing the data collection process, the training of the selected machine learning algorithms and the consequent evaluation is carried out. The experiments were structured as follows. For each task, the data set was divided into two parts. The former, consisting of the 80% of the samples, are used for training; the latter 20% is used as the validation set. In order to maintain the same proportion for each class, data were split in a stratified way. During the evaluation phase, different metrics were computed to evaluate the performance of the machine learning algorithms in the various tasks. They are: (i) precision, (ii) recall, (iii) f1-score, and (iv) accuracy. Finally, a comparison of the performance of a wide range of machine learning algorithms (i.e., Random Forest, Decision Tree, SVC, Logistic Regression, Gaussian Naive Bayes, and Multi-Layer Perceptron) is presented .

The remainder of the paper is organized as follows. Section 2 lists some works in the field of TinyML. Section 3 details the overall methodology, describing how the data collection, the model training, and the final evaluation are carried out. Then, Section 4 presents the results of the analysis of the sensitivity of sensors in the different tasks while Section 5 presents some reflections, also on the memory consumption and the inference rate of the different algorithms. Finally, Section 6 closes the paper, highlighting some future work.

2. Related works

Growing attention is being paid to the TinyML research field, with many studies that are investigating different aspects and dimensions of it. Many TinyML-based applications are being developed. For example, Kocacinar et al. [10] proposed a TinyML approach for masked face recognition. The authors fine-tuned some state-of-the-art convolutional neural network architectures, namely ResNet, VGG-16, and MobileNet, to detect three classes: masked, unmasked, and incorrect masked usage. Then, they adopted a fine-tuned low-dimensional model MobileNet architecture. Despite the significant reduction in model complexity, the model continues to perform well, achieving an overall accuracy of 90%. Other studies, instead, addressed the need for the efficient deployment of neural networks in edge devices. Manor and Greenberg [11] presented an efficient hardware–software framework to accelerate machine learning inference on edge devices. At the hardware level, it employs a dedicated Neural Processing Unit custom hardware accelerator. From the software point of view, a modified TensorFlow Lite for Microcontroller is used with the aim of efficiently mapping the computational load onto hardware and software. In particular, it supports weight compression of pruned quantized neural networks and exploits the sparsity of the pruned model to further reduce computational complexity. Finally, new devices specifically designed for TinyML are being studied. Giordano et al. [12] proposed a proof-of-concept device to continuously assess the usage of hand-held power tools and to detect possible construction working tasks. A TinyML algorithm is used to distinguish among four classes of usage (i.e., tool transportation, no-load, metal, and wood drilling) taking as input the three-axis accelerations.

Several software stacks have been released specifically for TinyML. Among them, we can list Tensorflow lite micro [13], Edge2train [14], OpenNN [15], Edge Machine Learning library [16], and Resource Constrained Edge-Neural Networks [17]. Some works, instead, focused on toolkits that take ML algorithms as input and generate code that can be executed on microcontrollers. Wang et al. [18] presented FANN-on-MCU, an open-source toolkit to generate neural networks developed using the fast artificial neural network library. The generated code can be run on both the ARM Cortex-M series and the novel RISC-V-based parallel ultralow-power platform. Contextually, many benchmarks have been proposed to evaluate the performance of ML algorithms running on tiny devices. Just to cite a few, there are TinyML Benchmark [19] and MLPerf Tiny Benchmark [20]. Sudharsan et al. [19] designed three fully connected neural networks: the first with a single hidden layer composed of ten neurons, the second with two hidden layers, respectively, with ten and fifty neurons, and the last one with ten hidden layers with ten neurons each. Each model has been trained on ten freely available datasets, thus obtaining thirty neural networks. Then, they evaluated the onboard models performance on seven of the most common MCU boards. Several aspects were analyzed: (i) the inference performance, (ii) the onboard accuracy, (iii) the memory consumption on MCUs, and (iv) the price-performance ratio. Banbury et al. [20], instead, proposed an application-level benchmark. It includes the following tasks: Keyword Spotting, Visual Wake Words, Image Classification, and Anomaly Detection. The frameworks measure latency, accuracy, and energy. It is important to notice that both the latency and the energy are evaluated five times and the median value is returned.

Anyway, the purpose of such benchmarks is to measure some aspects of the inference process of machine learning algorithms such as accuracy, latency, and energy consumption. To our knowledge, no work in the literature has yet provided an evaluation methodology to determine the limitations of a TinyML-based solution considering a sensitivity analysis of the combination of the microcontroller sensors possibilities with machine learning algorithms.

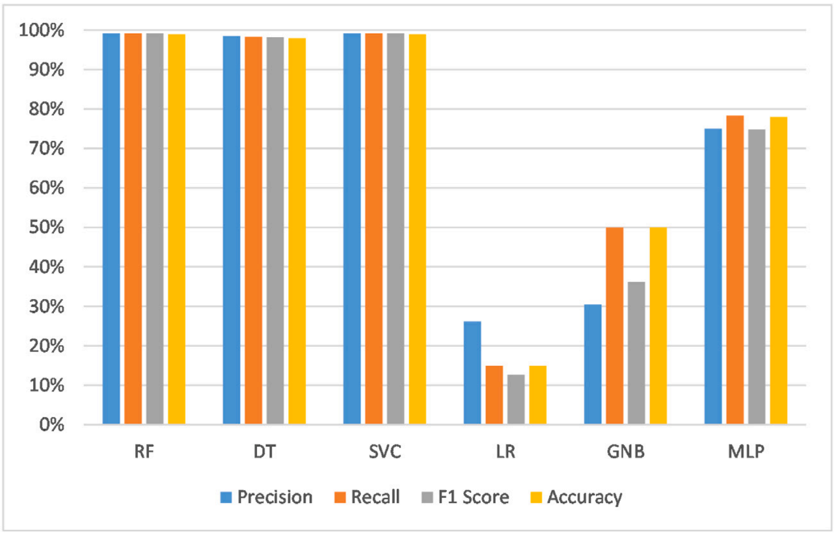

Fig. 1. Performance of the classifier in detecting six different sound frequencies (500 Hz, 1,000 Hz, 1,500 Hz, 2,000 Hz, 2,500 Hz, and 3,000 Hz).

3. Materials and methods

This Section describes the analyzed task, how the data collection and the training processes have been carried out, and the implementation details.

3.1. Task

This study focuses on the recognition of the following tasks:

- Sound Frequencies in the range from 500 Hz to 3,000 Hz.
- Colors shade, with some slight differences.
- Vibration Patterns, which alternate periods of vibration and “silence”.
- Vibration Intensities, generated by a vibration motor with different speeds.
- Keywords, from two to five words.
- Hand-gesture, depicting some alphabet letters.

3.2. Data collection process

For all the tasks, the data sets were collected using the sensors of the Arduino Nano 33 BLE. As already anticipated, the purpose of this work is to carry out a sensitivity analysis of the practical limitations of TinyML, employing an experimental approach. For this reason, data were collected under *laboratory conditions*, even if we are aware that real case studies applications will provide practical challenges and further problems. For each class of each task, one-hundred and twenty samples are collected. Two different splits for training and validation were evaluated. In the former one, one hundred samples are used for training and twenty ones for validation while in the latter one, ninety samples are used for training and thirty ones for validation.

The sound frequencies were generated using the Frequency Generator, an Android mobile app. This mobile app can generate sound frequencies from infra-sound (i.e., 1 Hz) to ultra-sound (i.e., 22,000 Hz). It is possible to choose among single or multifrequency, specifying several sound waves (e.g., sinusoidal, sawtooth, triangle, and square). The MP34DT05 digital microphone from Arduino Nano was used to record the different sound frequencies. The feature used consists of an array of 32 values. Each value is computed as the Root Mean Square (RMS) of 256 readings, recorded continuously with a sampling rate of 16 KHz, after a pulse density modulation. Between each sample, a delay of twenty milliseconds is applied. Hence, each feature is relative to a 640 ms period. During the data collection process, a threshold is applied to avoid background noise and silence, recording only sounds.

The color shades were considered to only be printed on paper. In fact, some experiments have shown that, if visualized on a screen, it is possible to discriminate among colors with a simple if. In particular, slight variations of green were considered. The APDS-9960 sensor from the Arduino Nano is employed. It is a sensor for digital proximity, ambient light, RGB, and gesture sensor. The feature vector is composed of five values. Three are relative to the three RGB components. The other two values are the ambient light intensity and proximity since the light in the room and the distance between the sensor and the paper/screen can significantly affect the sensed color values.

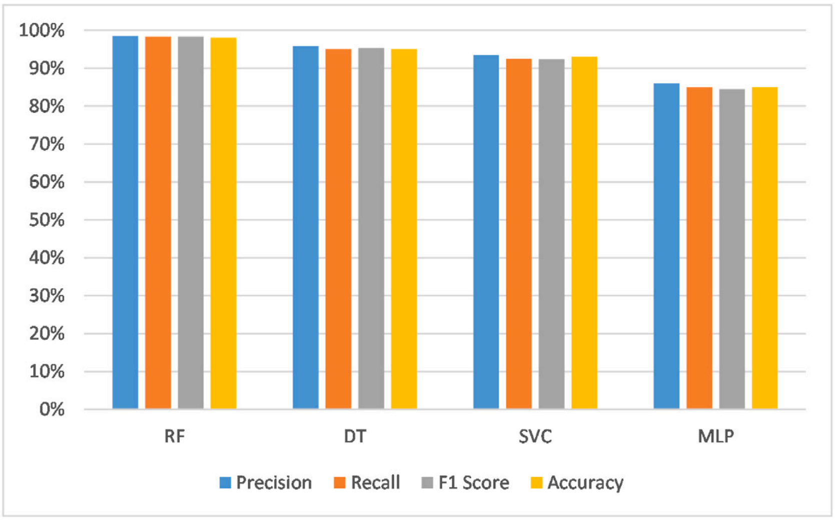

Fig. 2. Performance of the classifier in detecting six different sound frequencies (2,000 Hz, 2,200 Hz, 2,400 Hz, 2,600 Hz, 2,800 Hz, and 3,000 Hz).

Also, for the generation of vibration patterns, an Android mobile app, Vibrate Pattern Maker, was employed. It allows to create patterns alternating periods of vibration and “silence”. For both periods, silence and vibration, it is possible to set their duration in milliseconds. Unfortunately, it does not allow one to specify the intensity of the vibrations. Once a pattern is created, it can be repeated indefinitely.

With regard to the detection of vibration intensities, a vibration motor with five different speed settings was used. Patterns with no periods of silence were considered, just varying the intensity of vibrations, and changing the speed of the motor.

In both cases, detection of vibration patterns and vibration intensities, the LSM9DS1 sensor, which is an acceleration sensor, was used. It provides the accelerations in the three dimensions (i.e. x, y, and z). The feature vector is composed of sixty values, consisting of twenty samples for each dimension, with a delay of fifty milliseconds between each sample. Thus, the feature vector covers one second of the data.

The recordings of keywords were made by the same person who repeated one hundred times each keyword. To record them, the MP34DT05 microphone was always employed. Also, in this case, a period of 640 ms has been set for capturing the words, as happened for the sound frequencies.

Finally, for the hand-gesture recognition, the dataset was generated by people that, keeping the sensor in their hands, drew on air the letters of the alphabet. During the drawing, the movements were retrieved using the accelerometer sensor (i.e., LSM9DS1). For each letter, the feature vector consists of forty-two values that represent fourteen samples for x, y, and z dimensions. A delay of fifty milliseconds was added between each sample. Hence, the resulting feature vector covers a time span of 700 ms.

3.3. Model training and evaluation

After completing the data collection process, the selected machine learning algorithms have been trained and evaluated. The training phase has been carried out on a general-purpose PC. The experiments were structured as follows. For each task, the data set is divided into two parts. The former, consisting of 80% of the samples, is used for training while the latter 20% is used as the validation set. To maintain the same proportion for each class, the data were split in a stratified way.

During training, a tuning phase for each algorithm was conducted. In particular, the ten-fold cross-validation was used on the training set, varying the hyperparameters considered. We did not report all the details of the main parameters considered for each algorithm, since we are just interested in understanding the overall sensitivity of machine learning models and for the sake of conciseness.

Also, in this work, the accuracy of the model was evaluated directly on the Arduino Nano. This is because, from a TinyML perspective, we are interested in evaluating its limitations in several tasks, and determining the maximum accuracy achievable by a machine learning model running on the device. In fact, having the Arduino Nano limited computational resources, it is possible that the performance of machine learning algorithms that run on it differs from the one if they run on a more powerful device. To compare the performance of several machine learning algorithms directly on the device, during the evaluation of the first algorithm, the samples were collected and then used for the evaluation of the further algorithms. In this way, it is possible to evaluate the accuracy of the model directly on the Arduino Nano with the same set of data. During the various evaluations, twenty samples for each class were collected as the test set.

During the evaluation phase, different metrics was evaluated to measure the performance of the machine learning algorithms in the various tasks : (i) precision, (ii) recall, (iii) f1-score, and (iv) accuracy.

Table 1
Performance of the classifier in detecting eleven different sound frequencies (2,000 Hz, 2,100 Hz, 2,200 Hz, 2,300 Hz, 2,400 Hz, 2,500 Hz, 2,600 Hz, 2,700 Hz, 2,800 Hz, 2,900 Hz, and 3,000 Hz).

Model	Class	Precision	Recall	F1 Score	Accuracy
RF	2000	83%	100%	91%	65%
	2100	100%	95%	97%	
	2200	50%	35%	41%	
	2300	100%	85%	92%	
	2400	55%	30%	39%	
	2500	43%	45%	44%	
	2600	100%	90%	95%	
	2700	56%	70%	62%	
	2800	52%	55%	54%	
	2900	50%	60%	55%	
	3000	42%	55%	48%	
DT	2000	79%	55%	65%	52%
	2100	86%	95%	90%	
	2200	22%	20%	21%	
	2300	68%	85%	76%	
	2400	25%	15%	19%	
	2500	24%	25%	24%	
	2600	80%	80%	80%	
	2700	46%	60%	52%	
	2800	41%	35%	38%	
	2900	43%	50%	47%	
	3000	50%	55%	52%	
SVC	2000	62%	25%	36%	55%
	2100	100%	85%	92%	
	2200	70%	35%	47%	
	2300	93%	70%	80%	
	2400	20%	90%	32%	
	2500	86%	60%	71%	
	2600	100%	55%	71%	
	2700	87%	65%	74%	
	2800	44%	35%	39%	
	2900	79%	75%	77%	
	3000	50%	10%	17%	
MLP	2000	50%	10%	17%	34%
	2100	53%	100%	69%	
	2200	23%	15%	18%	
	2300	74%	70%	72%	
	2400	50%	30%	37%	
	2500	0%	0%	0%	
	2600	30%	100%	46%	
	2700	33%	25%	29%	
	2800	10%	10%	10%	
	2900	6%	10%	8%	
	3000	0%	0%	0%	

3.4. Machine learning algorithms and implementation details

In this work, the performance of almost all the machine learning algorithms supported by the `micromlgen` package of the EloquentTinyML library [21] were compared: Random Forest, Decision Tree, Support Vector Classifier, Logistic Regression, Gaussian Naive Bayes, and Multi-Layer Perceptron.

A Random Forest (RF) is a meta-estimator that combines the prediction of several decision trees on different subsamples of the training set with the aim of improving the prediction accuracy and preventing overfitting [22]. We employed its implementation in the Scikit-learn library [23], `sklearn.ensemble.RandomForestClassifier`.

A Decision Tree (DT) is a tree-based technique in which any path beginning from the root is described by a data separating sequence until the classification outcome is provided in the leaf node [24]. For this algorithm, we also took advantage of the Scikit-learn library.

A Support Vector Classifier (SVC) is a classifier based on support vector machines. They are a machine learning algorithm that determines a decision boundary to separate the classes in a transformed space, maximizing the margins (i.e., the distance between the decision boundary and the objects belonging to the different classes) [25]. We used its implementation within the Scikit-learn library, sklearn.svm.SVC.

In the Logistic Regression (LR), the classes are modeled as a linear combination of features of a logistic model [26]. Also in this case, we take advantage of the Scikit-learn library (`sklearn.linear_model.LogisticRegression`).

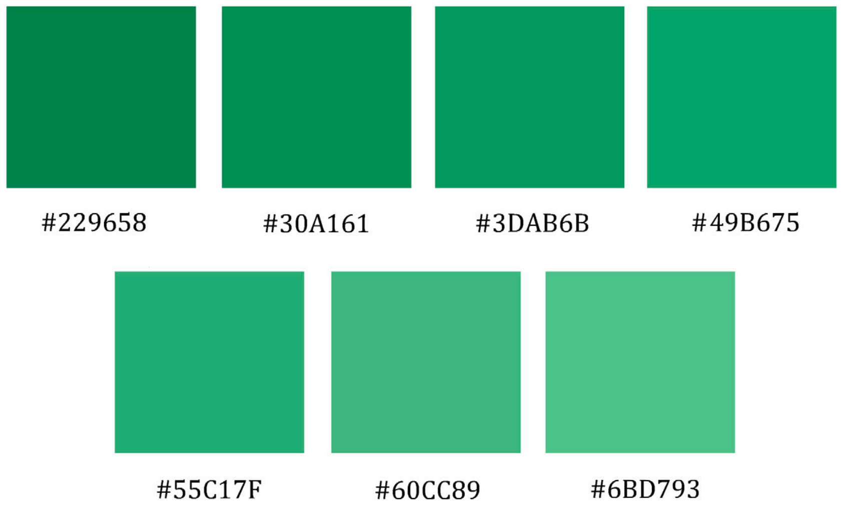

Fig. 3. Green color scale used in the experiments.

The Gaussian Naive Bayes (GNB) belongs to the family of probabilistic classifiers based on the Bayes theorem with strong (naive) independence assumptions between the features [27]. The GNB assumes that continuous values are distributed according to a Gaussian distribution [28]. We took advantage of the class `sklearn.naive_bayes.GaussianNB` within the Scikit-learn library.

Finally, a Multilayer Perceptron (MLP) is a fully connected feedforward artificial neural network [29]. We implemented them employing the Tensorflow framework [30].

All the experiments were conducted using Python, version 3.6.13. The following Python libraries, with the specific version, were used:

- Tensorflow, version 2.3.0;
- Scikit-learn library, version 0.21.3;
- Pandas, version 1.1.5;
- Micromlgen, version 1.1.27;
- Tinymlgen, version 0.2.

To import the trained models on the Arduino Nano, the EloquentTinyML library [21] was used. It is a library for the deployment of machine learning algorithms, developed using the Scikit-learn and the Tensorflow libraries.

4. Results

This Section presents the results obtained using several machine learning algorithms in the different tasks.

4.1. Sound frequency

The experiments for the evaluation of the sensitivity of the microphone startedy training the models to discriminate among sound frequencies from 500 Hz to 3,000 Hz, varying them by 500 Hz. For each frequency, 20 samples were collected. The results are reported in Fig. 1. The RF, the DT, and the SVC got excellent performance on the test, with an overall accuracy of over 98%. The MLP got lower, but still good performance, with an accuracy of 78%. Instead, poor performance was obtained by the LR and GNB.

In the next experiments, the range was narrowed between 2,000 and 3,000 Hz, but the frequencies were varied by 200 Hz. Also, in these experiments, twenty samples per class were collected. Fig. 2 illustrates the performance of the various classifiers. Given their poor performances in the previous experiment, the LR and the GNB were no more evaluated. The RF maintained the same level of performance, while the DT and the SVC performed slightly worse. Finally, the MLP increases the overall accuracy, passing from 78% to 85%.

In the last experiment, the same frequency range (i.e., from 2,000 Hz to 3,000 Hz) was considered, but the step was further reduced from 200 Hz to 100 Hz. In line with previous experiments, twenty examples per class were used. The results are reported in Table 1. As shown, the models are no longer able to distinguish well among the various frequencies. The results are similar to those of the previous experiment. RF still performs better than the other algorithms, reaching an accuracy of 65%. The DT and the SVC have slightly worse performance, whereas poor performance was obtained by the MLP. Analyzing the various metrics on each class obtained by the RF, it is interesting to notice that has excellent performance in some classes (2,000 Hz, 2,100 Hz, 2,300 Hz, and 2,600 Hz) and poor performances in the other ones, with no intermediate performance between them.

Table 2
Performance of the classifier in detecting colors.

Model	Class	Precision	Recall	F1 Score	Accuracy
RF	#229658	79%	65%	71%	70%
	#30A161	60%	97%	74%	
	#3DAB6B	72%	33%	45%	
	#49B675	83%	50%	62%	
	#55C17F	61%	82%	70%	
	#60CC89	60%	68%	64%	
	#6BD793	95%	97%	96%	
DT	#229658	69%	68%	68%	68%
	#30A161	67%	78%	72%	
	#3DAB6B	52%	40%	45%	
	#49B675	82%	45%	58%	
	#55C17F	61%	82%	70%	
	#60CC89	57%	70%	63%	
	#6BD793	95%	93%	94%	
SVC	#229658	79%	57%	67%	68%
	#30A161	69%	78%	73%	
	#3DAB6B	55%	53%	54%	
	#49B675	66%	47%	55%	
	#55C17F	62%	80%	70%	
	#60CC89	57%	68%	62%	
	#6BD793	93%	93%	93%	
LR	#229658	38%	30%	33%	38%
	#30A161	39%	28%	32%	
	#3DAB6B	26%	40%	32%	
	#49B675	32%	78%	45%	
	#55C17F	14%	3%	4%	
	#60CC89	46%	30%	36%	
	#6BD793	82%	57%	68%	
GNB	#229658	100%	5%	10%	26%
	#30A161	71%	30%	42%	
	#3DAB6B	0%	0%	0%	
	#49B675	10%	3%	4%	
	#55C17F	36%	95%	52%	
	#60CC89	15%	53%	23%	
	#6BD793	0%	0%	0%	
MLP	#229658	67%	5%	9%	51%
	#30A161	51%	97%	67%	
	#3DAB6B	42%	45%	43%	
	#49B675	30%	45%	36%	
	#55C17F	90%	23%	36%	
	#60CC89	50%	65%	57%	
	#6BD793	86%	75%	80%	

4.2. Color

The sensitivity of the sensor to detect colors was evaluated using seven shades of green. Their hexadecimal RGB code are: #229658, #30A161, #3DAB6B, #49B675, #55C17F, #60CC89, and #6BD793. A set of squares filled with these colors are reported in Fig. 3.

In the first experiment, various models were used to distinguish among all the shades of green. The results are reported in Table 2. Although the differences between the different shades of green are minimal, some models (i.e. RF, DT, and SVC) can differentiate among them with acceptable accuracy, while the other models have poor performance.

A further experiment was carried out to understand if the performance improves using less similar colors. Only three colors were used. The extremes of the green scales, that are #229658 and #6BD793, and the one in between (i.e., #49B675). As shown in Fig. 4, the GNB continues to perform poorly. The LR and the MLP have acceptable performance, with an accuracy of more than 70%. Finally, the RF, the DT, and the SVC continued to be the most accurate algorithms, since they achieved an average accuracy of over 90%.

4.3. Vibration pattern

In the evaluation of the sensitivity of the Arduino Nano in detecting vibration patterns, several patterns were evaluated, alternating periods of vibration and silence. Firstly, three different patterns were used, in which the vibrations last like the pause period, respectively 100 ms, 150 ms, and 200 ms. The model is able to perfectly discriminate among the different patterns, as witnessed by the metrics reported in Fig. 5.

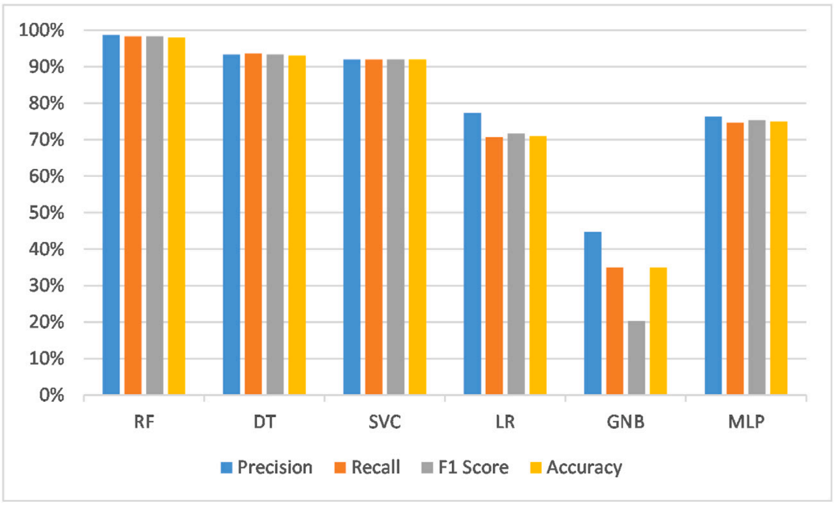

Fig. 4. Performance of the classifier in detecting colors among three shades of green (#229658, #49B675, and #6BD793).

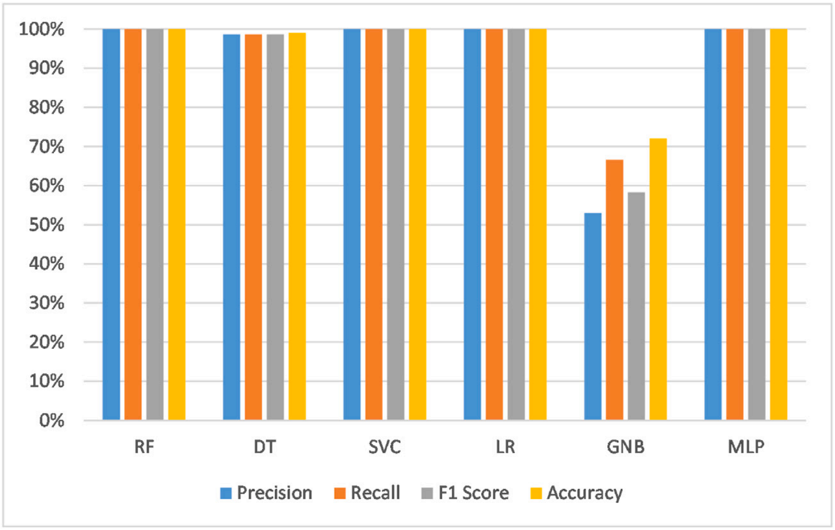

Fig. 5. Performance of the classifier in detecting vibration patterns of 100 ms, 150 ms, and 200 ms.

In the next experiment, three additional patterns were added, always with equal periods of the vibrations and pauses. They last respectively 10 ms, 25 ms, and 50 ms. As highlighted by the results reported in Fig. 6 , all algorithms are able to perfectly discriminate among the different patterns, with the only exception of the GNB. It is important to notice that the results for the SVC are not reported. The reason is that the sketch with the model weighed too much to be loaded on Arduino and it was therefore not possible to evaluate the test set with it directly on the device.

Then, patterns composed of periods of vibrations and pauses with different durations were considered. In the former, vibrations of 25 ms are followed by pauses of 50 ms. In the latter, the durations are reversed. Vibrations last 50 ms, while pauses are just 25 ms. As shown in Fig. 7 , all algorithms have perfect performance, with the only exception of the SVC that committed few errors.

In the final experiment, an extreme test was conducted using patterns with very small pause periods and vibrations, 1 ms and 2 ms, respectively. As shown in Fig. 8, the RF and MLP still obtained excellent performances. The DT still got a good accuracy while the LR and the GNB got bad performance. Also, for this experiment, the resulting SVC cannot be loaded in the Arduino because of the weight of the relative sketch.

4.4. Vibration intensity

Considering the detection of different vibration intensities, the aim of the experiment was to distinguish among the five-speed settings available in a vibration motor. The results are illustrated in Table 3. Even in this task, many classifiers (i.e., RF, SVC, and

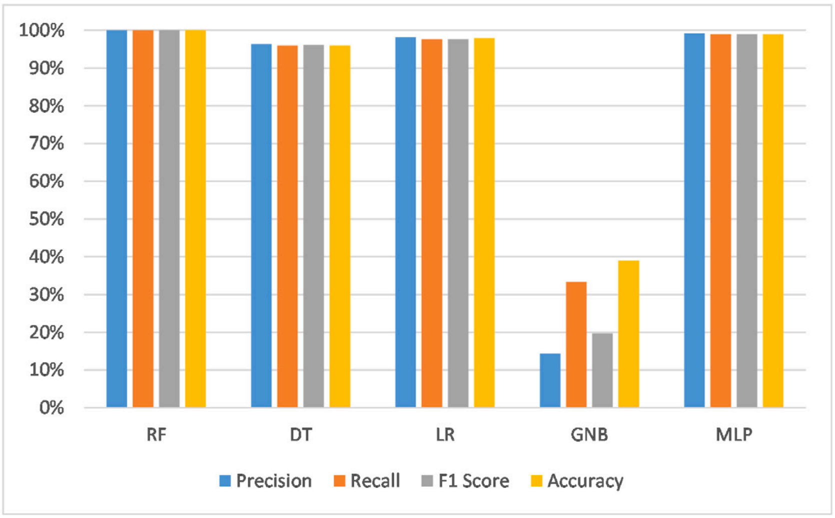

Fig. 6. Performance of the classifier in detecting vibration patterns of 10 ms, 25 ms, 50 ms, 100 ms, 150 ms, and 200 ms.

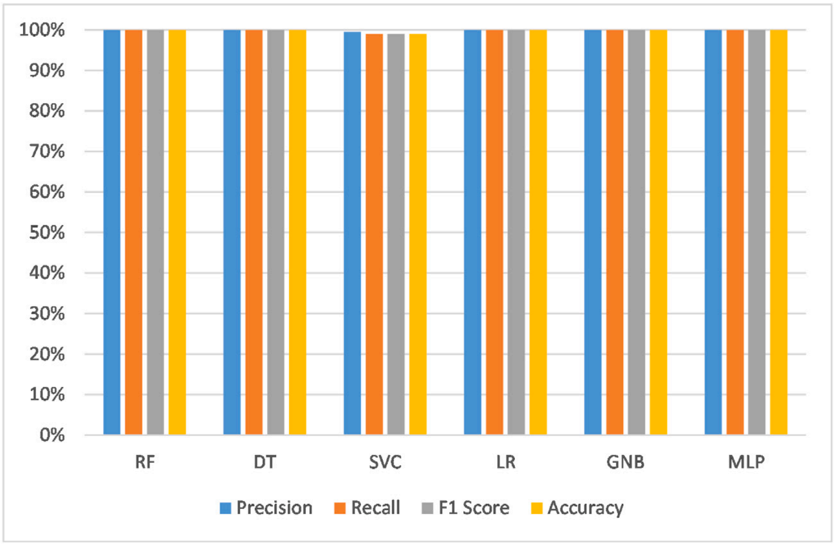

Fig. 7. Performance of the classifier in detecting vibration patterns of 25 ms and 50 ms with pauses of 50 ms and 25 ms.

MLP) got excellent performances with accuracy values of over 90%. The DT got lower performance while the LR and the GNB have the worst ones.

4.5. Keyword

With regard to the keyword spotting task, the first experiment had the aim of distinguish between two different keywords, *yes* and *no*. The results are summarized in Table 4. As shown, only the RF can cross the threshold of 80%. The MLP and the DT are slightly lower, respectively 77% and 78%. The SVC got performance slightly better than a random classifier, while GNB and LR got very poor performance.

In further experiments, more words were gradually added to the initial two ones. The words were the following ones: *ok*, *start*, and *stop*. After adding each word, the various models were trained to distinguish among the words, respectively three, four, and five. The results using three, four and five words are reported respectively in Figs. 9, 10, and 11. Generally, the performance of all algorithms was reduced. The algorithm with the highest is still the RF, whose accuracy remains in the range from 60% to 69%. Then, there is the SVC with slightly lower accuracy. The other algorithms have an accuracy of 50% or less. It is interesting to note that the LR with three and five words got an accuracy of more than 60%, higher than the one obtained using only two words.

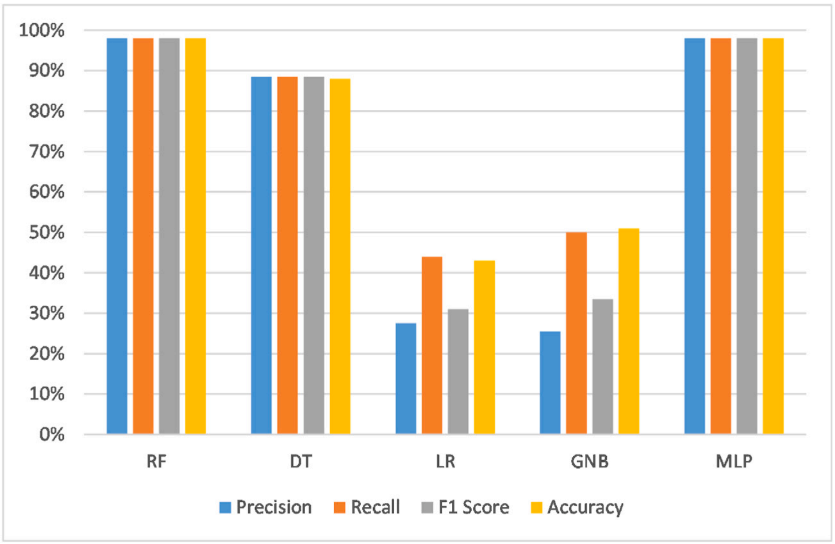

Fig. 8. Performance of the classifier in detecting vibration patterns of 1 ms and 2 ms.

Table 3
Performance of the classifier in detecting different vibration intensities.

Model	Class	Precision	Recall	F1 Score	Accuracy
RF	1	100%	99%	100%	99%
	2	99%	100%	100%	
	3	100%	99%	100%	
	4	99%	97%	98%	
	5	97%	99%	98%	
DT	1	83%	87%	85%	79%
	2	80%	78%	79%	
	3	83%	75%	79%	
	4	72%	74%	73%	
	5	79%	82%	81%	
LR	1	38%	30%	34%	52%
	2	38%	51%	43%	
	3	86%	98%	92%	
	4	18%	8%	11%	
	5	55%	71%	62%	
GNB	1	0%	0%	0%	20%
	2	0%	0%	0%	
	3	20%	100%	33%	
	4	0%	0%	0%	
	5	0%	0%	0%	
MLP	1	91%	96%	93%	92%
	2	96%	97%	96%	
	3	96%	97%	96%	
	4	88%	76%	82%	
	5	87%	92%	90%	

4.6. *Hand-gesture*

Finally, the sensitivity in the task of hand-gesture recognition was evaluated. In the first experiment, the model had to discriminate among five different hand gestures, made to represent the vowels (that is, *a*, *e*, *i*, *o*, and *u*) in air. An example of gestures made to depict them is reported in Fig. 12. Fig. 13 illustrates the results. As shown, all the algorithms are able to perfectly discriminate among the various vowels. In fact, the lowest accuracy value, achieved by the GNB, was 85%.

Then, more hand gestures for depicting some consonants were added: *b*, *d*, *g*, *l*, and *m*. The relative gestures are shown in Fig. 14. Also in this case, the various classifiers continued to perform well, as shown by the various metrics reported in Table 5.

To conclude the experiments on the hand-gesture recognition, the gestures relative to the remaining consonants of the alphabet were added, whose gestures are pictured in Fig. 15. In this experiment, both the RF and the SVC sketches are too heavy to be loaded

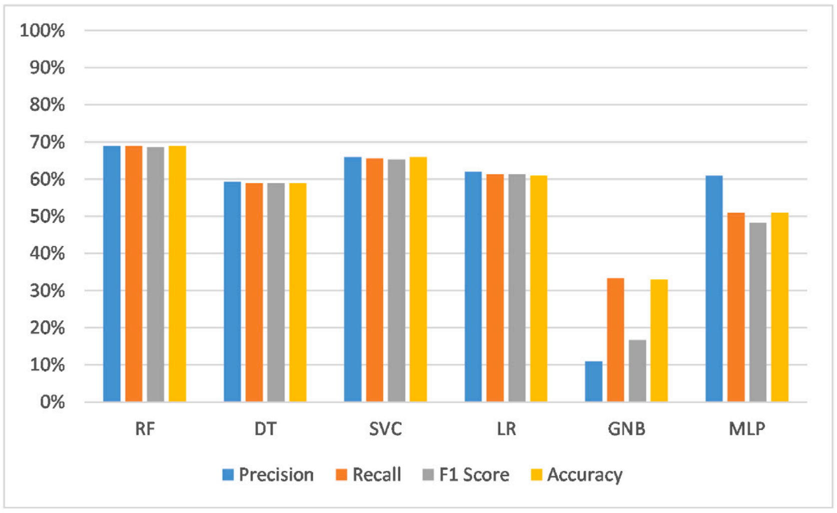

Fig. 9. Performance of the classifier in spotting three keywords: yes, no, and ok.

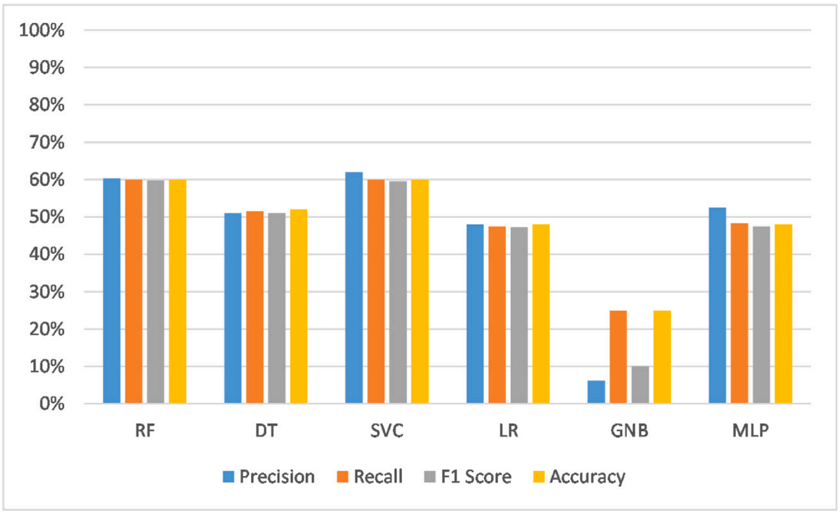

Fig. 10. Performance of the classifier in spotting four keywords: yes, no, ok, and start.

Table 4
Performance of the classifier in spotting two keywords: yes and no.

Model	Class	Precision	Recall	F1 Score	Accuracy
RF	no	83%	83%	83%	83%
	yes	83%	83%	83%	
DT	no	79%	77%	78%	78%
	yes	77%	80%	79%	
SVC	no	60%	90%	72%	65%
	yes	80%	40%	53%	
LR	no	33%	33%	33%	33%
	yes	33%	33%	33%	
GNB	no	50%	100%	67%	50%
	yes	0%	0%	0%	
MLP	no	77%	77%	77%	77%
	yes	77%	77%	77%	

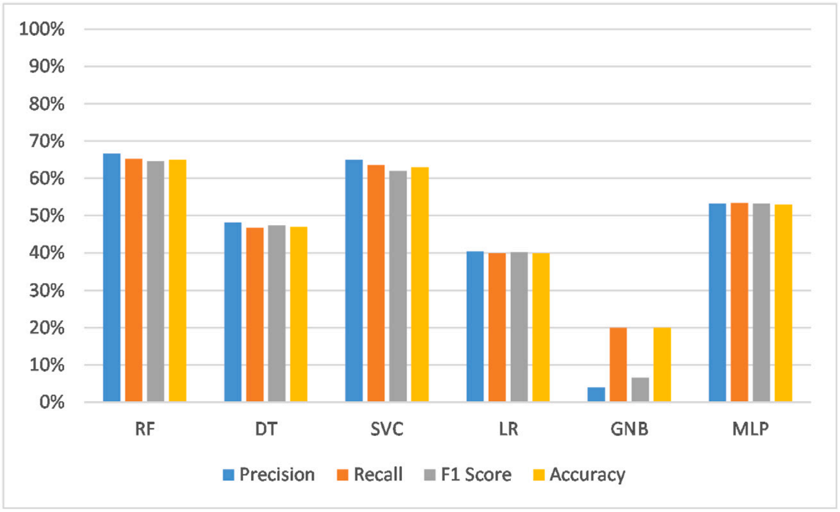

Fig. 11. Performance of the classifier in spotting five keywords: yes, no, ok, start, and stop.

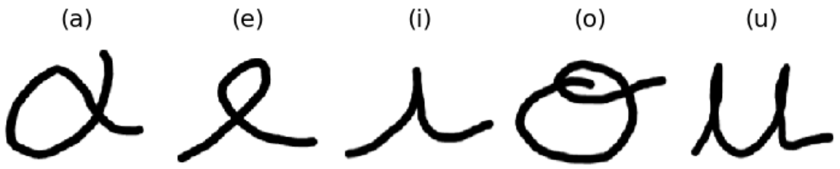

Fig. 12. Hand-gestures of vowels.

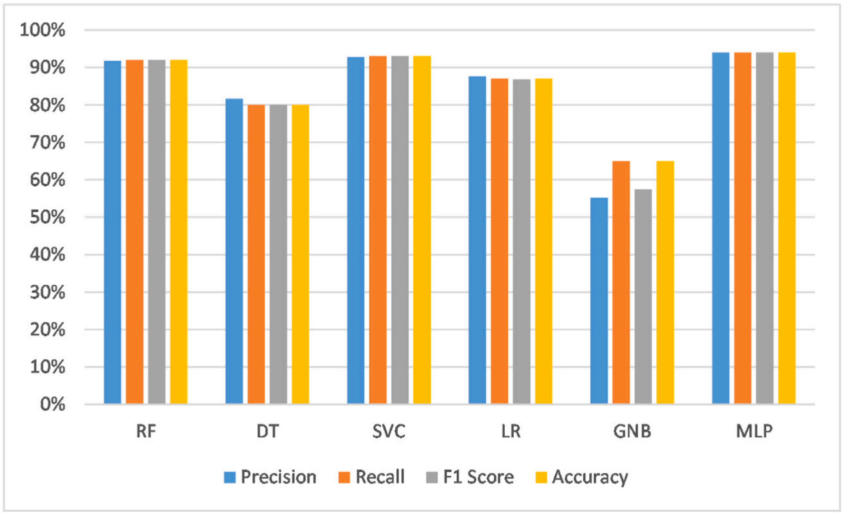

Fig. 13. Performance of the classifier in detecting hand-gestures, that depicts the five vowels.

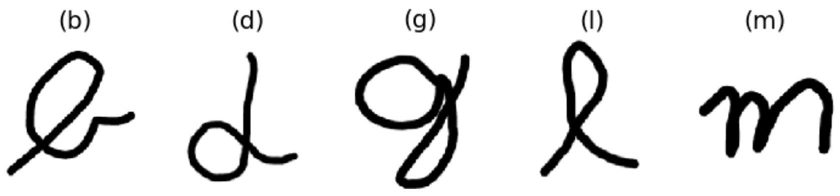

Fig. 14. Hand-gestures of the five consonants added to the vowels.

Table 5
Performance of the classifier in detecting hand gestures that depict five vowels and five consonants.

Model	Class	Precision	Recall	F1 Score	Accuracy
RF	A	94%	80%	86%	91%
	B	83%	95%	88%	
	D	86%	95%	90%	
	E	85%	85%	85%	
	G	95%	90%	92%	
	I	91%	100%	95%	
	L	88%	75%	81%	
	M	100%	90%	95%	
	O	87%	100%	93%	
	U	100%	95%	97%	
DT	A	81%	65%	72%	75%
	B	58%	75%	65%	
	D	83%	75%	79%	
	E	50%	60%	55%	
	G	100%	80%	89%	
	I	78%	90%	84%	
	L	55%	55%	55%	
	M	83%	75%	79%	
	O	89%	85%	87%	
	U	90%	90%	90%	
SVC	A	86%	95%	90%	94%
	B	90%	90%	90%	
	D	100%	100%	100%	
	E	89%	80%	84%	
	G	100%	95%	97%	
	I	95%	100%	98%	
	L	90%	95%	93%	
	M	100%	90%	95%	
	O	91%	100%	95%	
	U	100%	95%	97%	
LR	A	94%	80%	86%	88%
	B	73%	80%	76%	
	D	90%	95%	93%	
	E	71%	85%	77%	
	G	100%	85%	92%	
	I	95%	95%	95%	
	L	84%	80%	82%	
	M	90%	90%	90%	
	O	86%	95%	90%	
	U	100%	90%	95%	
GNB	A	14%	100%	24%	28%
	B	0%	0%	0%	
	D	100%	10%	18%	
	E	0%	0%	0%	
	G	0%	0%	0%	
	I	63%	95%	76%	
	L	0%	0%	0%	
	M	100%	10%	18%	
	O	0%	0%	0%	
	U	74%	70%	72%	
MLP	A	94%	80%	86%	91%
	B	89%	80%	84%	
	D	95%	90%	92%	
	E	81%	85%	83%	
	G	95%	95%	95%	
	I	83%	100%	91%	
	L	91%	100%	95%	
	M	95%	90%	92%	
	O	90%	95%	93%	
	U	100%	95%	97%	

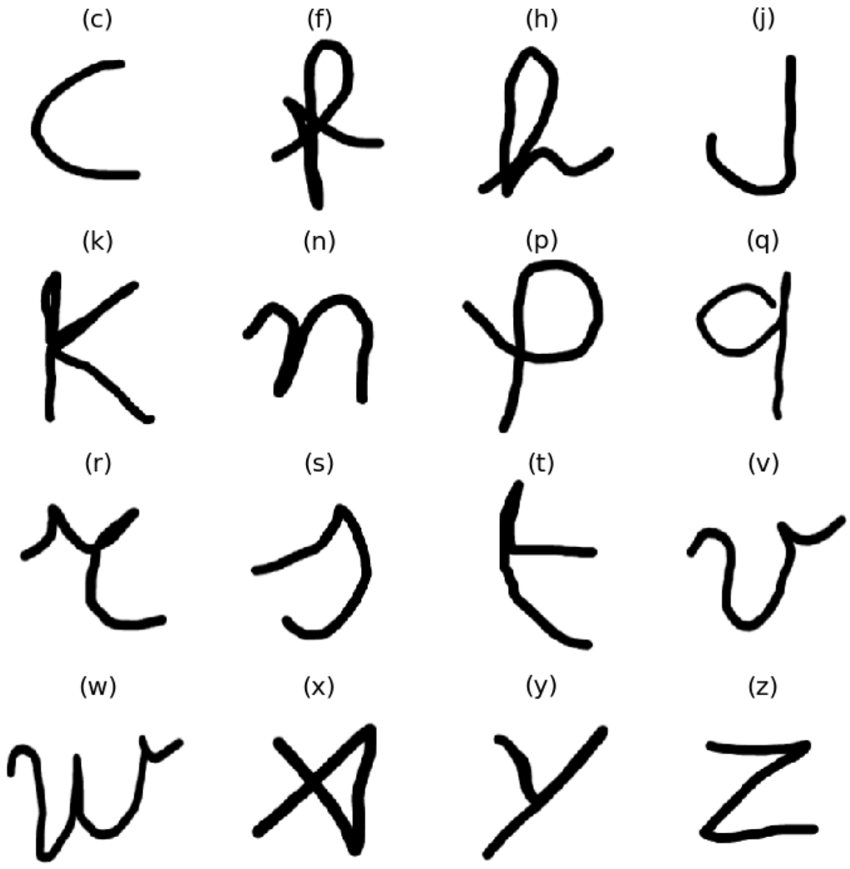

Fig. 15. Hand-gestures of the remaining consonants.

into the device, hence it is not possible for them to evaluate the test set. The other algorithms continue to perform well, with the highest accuracy value that is achieved by the MLP, as shown in Fig. 16.

5. Discussion

In this study, a sensitivity analysis of the practical limitations of TinyML using an experimental approach was presented. The Arduino Nano 33 BLE sense was chosen as the TinyML device and the precision that can be obtained with it in several classification tasks was evaluated: frequencies, colors, vibration patterns, vibration intensities, keywords, and hand gestures.

For each task considered, it was possible to train an ML algorithm capable of discriminating among the various classes. In all the tasks, the GNB has the worst performance. This is probably due to the fact that it based its prediction on the assumption that each class follows a Gaussian distribution. Even if it is the most simple algorithm, the Logistic Regression in some cases has performances in line with the ones obtained by the other algorithms. The MLP generally performs worse than the RF, the DT, and the SVC probably because of the limited number of examples in the datasets (that is, one hundred per class). Furthermore, for some tasks like the frequency detection one, it would work better if use the raw sound data directly as input. Finally, the RF, the DT, and the SVC have similar performances but in general, RF performs slightly better than the other two algorithms.

It is also interesting to evaluate the memory occupancy and the inference time of each algorithm in the various experiments. They are reported respectively in Tables 6 and 7. In particular, the memory occupancy consists of the weight (expressed in KiloBytes) of the sketch containing the model, that is loaded into the Arduino Nano 33 BLE Sense. Instead, the inference time represents the time (expressed in milliseconds) taken by the different algorithms to evaluate the test sets. It is important to recall that each test set is composed of 20 samples per class.

It is possible to notice that the DT, the LR, and the GNB are the most lightweight algorithms and also the fastest in the evaluation of the test set in many cases. The SVC is the algorithm that takes up the most memory space, followed by the RF and MLP. Parallelly, SVC is significantly the slowest algorithm, followed by the MLP and then the RF.

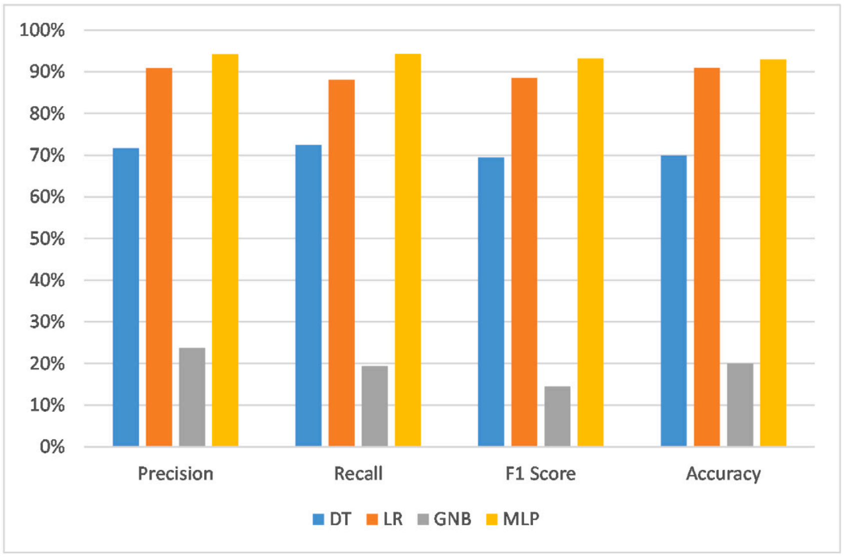

Fig. 16. Performance of the classifiers in detecting hand-gestures, that depict all the letters of the alphabet.

Table 6
Memory occupancy (KB) of the algorithms for the different tasks.

Task	Experiment	Algorithm					
		RF	DT	SVC	LR	GNB	NN
SF	6 (500 Hz)	108	97	201	101	101	289
	6 (200 Hz)	114	97	216	101	101	276
	11	315	115	880	117	117	299
C	3	115	85	112	85	86	190
	7	149	88	222	88	89	197
VP	3	224	191	254	194	195	339
	6	467	281	–	284	283	466
	2 (25–50)	222	209	410	210	211	344
	2 (1–2)	741	261	–	256	258	391
VI	5	755	277	–	273	273	443
K	2	119	90	235	90	91	225
	3	156	95	275	95	96	230
	4	202	100	366	100	100	346
	5	761	106	449	104	104	278
HG	5	209	99	215	102	102	250
	10	364	117	403	122	122	271
	26	–	111	–	118	118	271

With respect to these data, it can be deduced that RF represents the best combination in terms of performance, memory occupancy, and inference time. Instead, for some tasks, the use of DT can also be evaluated. Even if the SVC generally obtained excellent accuracy in many tasks, it requires a lot of memory space and is significantly the slowest algorithm in predictions.

A final consideration regarding the use of MLP is needed. Even if in the analyzed tasks it had performance similar, or slightly lower, than other algorithms, for more complex cases that require, for example, the analysis of images or in which there is more data, they are the best solution.

6. Conclusion

In this paper, we proposed an evaluation methodology to determine the limitations of a TinyML-based solution. Such a methodology can be used to evaluate the limitations of an ‘‘Arduino Nano 33 BLE sense’’. The evaluated classification tasks were sound frequencies, colors, vibration patterns and intensities, keywords, and hand gestures. Several algorithms were compared, including Random Forest, Decision Tree, SVC, Logistic Regression, Gaussian Naive Bayes, and Multi-Layer Perceptron. Finally, the results were discussed along with some reflections on the memory occupancy and inference rate of each algorithm.

Table 7
Inference time (ms) for the test set evaluation of the algorithms for the different tasks.

Task	Experiment	Algorithm					
		RF	DT	SVC	LR	GNB	NN
SF	6 (500 Hz)	19	1	3661	17	88	559
	6 (200 Hz)	22	1	4191	17	88	482
	11	84	3	99999	54	287	992
C	3	6	1	1240	1	7	72
	7	23	1	10465	11	43	226
VP	3	139	5	28800	58	299	1166
	6	384	13	–	204	1095	3689
	2 (25–50)	221	7	138529	26	234	1029
	2 (1–2)	382	12	–	34	337	1379
VI	5	635	17	–	160	837	2939
K	2	13	1	3320	1	14	117
	3	27	1	6092	6	31	180
	4	44	1	12642	10	54	883
	5	266	1	19951	16	87	590
HG	5	55	1	7639	15	82	270
	10	140	3	32215	58	304	592
	26	–	1	–	74	395	376

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgment

The authors are deeply grateful to Luigi Borriello for his precious support.

References

[1] P. Warden, D. Situnayake, Tinyml: Machine Learning with Tensorflow Lite on Arduino and Ultra-Low-Power Microcontrollers, O'Reilly Media, 2019.
[2] W. Mao, Z. Zhao, Z. Chang, G. Min, W. Gao, Energy efficient industrial internet of things: Overview and open issues, IEEE Trans. Ind. Inform. (2021).
[3] M.S. Islam, H. Verma, L. Khan, 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, IEEE, 2019, pp. 228–235.
[4] S. Zhou, Y. Du, B. Chen, Y. Li, X. Luan, An intelligent IoT sensing system for rail vehicle running states based on TinyML, IEEE Access 10 (2022) 98860–98871, http://dx.doi.org/10.1109/ACCESS.2022.3206954.
[5] M.S. Diab, E. Rodriguez-Villegas, Embedded machine learning using microcontrollers in wearable and ambulatory systems for health and care applications: A review, IEEE Access 10 (2022) 98450–98474, http://dx.doi.org/10.1109/ACCESS.2022.3206782.
[6] S.A.R. Zaidi, A.M. Hayajneh, M. Hafeez, Q.Z. Ahmed, Unlocking edge intelligence through tiny machine learning (tinyml), IEEE Access 10 (2022) 100867–100877, http://dx.doi.org/10.1109/ACCESS.2022.3207200.
[7] E. Manor, S. Greenberg, Custom hardware inference accelerator for TensorFlow lite for microcontrollers, IEEE Access 10 (2022) 73484–73493, http://dx.doi.org/10.1109/ACCESS.2022.3189776.
[8] B. Kocacinar, B. Tas, F.P. Akbulut, C. Catal, D. Mishra, A real-time CNN-based lightweight mobile masked face recognition system, IEEE Access 10 (2022) 63496–63507, http://dx.doi.org/10.1109/ACCESS.2022.3182055.
[9] G. Delnevo, C. Prandi, S. Mirri, P. Manzoni, Evaluating the practical limitations of TinyML: an experimental approach, in: 2021 IEEE Globecom Workshops (GC Wkshps), IEEE, 2021, pp. 1–6.
[10] B. Kocacinar, B. Tas, F.P. Akbulut, C. Catal, D. Mishra, A real-time CNN-based lightweight mobile masked face recognition system, IEEE Access 10 (2022) 63496–63507.
[11] E. Manor, S. Greenberg, Custom hardware inference accelerator for TensorFlow lite for microcontrollers, IEEE Access 10 (2022) 73484–73493.
[12] M. Giordano, N. Baumann, M. Crabolu, R. Fischer, G. Bellusci, M. Magno, Design and performance evaluation of an ultra low-power smart IoT device with embedded TinyML for asset activity monitoring, IEEE Trans. Instrum. Meas. (2022).
[13] 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, 2020, arXiv preprint arXiv:2010.08678.
[14] B. Sudharsan, J.G. Breslin, 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, 2020, pp. 1–8.
[15] OpenNN, https://www.opennn.net/. (Accessed 08 May 2021).
[16] D.K. Dennis, Y. Gaurkar, S. Gopinath, S. Goyal, C. Gupta, M. Jain, S. Jaiswal, A. Kumar, A. Kusupati, C. Lovett, S.G. Patil, O. Saha, H.V. Simhadri, EdgeML: Machine Learning for resource-constrained edge devices. URL https://github.com/Microsoft/EdgeML.

[17] B. Sudharsan, J.G. Breslin, 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, 2020, pp. 1–8.
[18] X. Wang, M. Magno, L. Cavigelli, L. Benini, FANN-on-MCU: An open-source toolkit for energy-efficient neural network inference at the edge of the Internet of Things, IEEE Internet Things J. 7 (5) (2020) 4403–4417.
[19] B. Sudharsan, S. Salerno, D.-D. Nguyen, M. Yahya, A. Wahid, P. Yadav, J.G. Breslin, 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.
[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, 2021, arXiv preprint arXiv:2106.07597.
[21] EloquentTinyML, 2022, https://github.com/eloquentarduino/EloquentTinyML. (Accessed 22 May 2022).
[22] G. Biau, E. Scornet, A random forest guided tour, Test 25 (2) (2016) 197–227.
[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, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[24] B. Charbuty, A. Abdulazeez, Classification based on decision tree algorithm for machine learning, J. Appl. Sci. Technol. Trends 2 (01) (2021) 20–28.
[25] S. Suthaharan, Support vector machine, in: Machine Learning Models and Algorithms for Big Data Classification, Springer, 2016, pp. 207–235.
[26] D.G. Kleinbaum, K. Dietz, M. Gail, M. Klein, M. Klein, Logistic Regression, Springer, 2002.
[27] I. Rish, et al., An empirical study of the naive Bayes classifier, in: IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence, Vol. 3, No. 22, 2001, pp. 41–46.
[28] A.H. Jahromi, M. Taheri, A non-parametric mixture of Gaussian naive Bayes classifiers based on local independent features, in: 2017 Artificial Intelligence and Signal Processing Conference, AISP, IEEE, 2017, pp. 209–212.
[29] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning Internal Representations by Error Propagation, Tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.
[30] P. Goldsborough, A tour of tensorflow, 2016, arXiv preprint arXiv:1610.01178.