Future Generation Computer Systems 148 (2023) 211–224

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Embedded real-time objects' hardness classification for robotic grippers

Youssef Amin, Christian Gianoglio [*], Maurizio Valle

DITEN Department, University of Genoa, Via all'opera Pia 11a, Genoa, 16145, Italy

ARTICLE INFO

Article history:
Received 29 November 2022
Received in revised form 22 May 2023
Accepted 3 June 2023
Available online 7 June 2023

Dataset link: https://github.com/YoussifAmin/Object_Hardness_Classification

Keywords:
Resource-constrained devices
Machine learning
Tactile sensing
Robotic grippers
Hardness classification

ABSTRACT

Robotic grippers can be equipped with tactile sensing systems to extract information from a manipulated object. The real-time classification of the physical properties of a grasped object on resource-constrained devices requires efficient and effective pre-processing techniques and machine-learning (ML) algorithms. In this paper, we propose a tactile sensing system mounted on the Baxter robot for the hardness classification of objects. In particular, we pre-processed the raw data with low computational cost techniques, and we designed three ML algorithms to provide real-time, energy-efficient, and low-memory impact classification on a resource-constrained microcontroller. Results show that convolutional neural networks (CNNs) achieve the best accuracy ($>$ 98%), while the support vector machine (SVM) presents the lowest memory occupation (1576 bytes), inference time ($<$ 0.077 ms), and energy consumption ($<$ 5.74 μJ).

1. Introduction

For both humans and robots, tactile sensing is crucial for interaction with the environment. Tactile information conveys to humans the physical properties of objects, such as hardness, texture, weight, shape, etc., allowing us to identify them even without any source of light [1]. Among the object properties, hardness is considered one of the most important attributes. It represents the resistant force of solid materials subject to a localized compressive force. It is also described as the ratio between the applied force and the displacement created by indentation. For humans, one of the common ways to estimate hardness is by tapping the surface of an object with a fingertip [2]. In contrast, hardness detection in robots is still a major limitation due to the lack of techniques to estimate it. Nevertheless, applying embedded intelligence to tactile sensing systems may help to overcome this problem.

Embedding intelligence near the sensor location has become increasingly important as it enables tactile sensing systems to be involved in a variety of applications such as prosthetics [3–5], sensorized gloves [6,7], and robotics [8–11]. Basically, tactile sensing systems consist of an array of distributed sensors to measure the applied mechanical stimuli, a readout circuit for signal conditioning and data acquisition, and embedded electronics for elaborating the information. The elaboration procedure could be either simple or complex depending on the task. As an example, location and force estimation require simple processing algorithms, however, extracting high-level information regarding object properties demands more sophisticated processing methodologies. The literature presents some works dealing with high-level information extracted from tactile data in robotics such as [12–17] for hardness classification, [18–20] for texture recognition, and [5,21] for slippage detection. In general, the more detailed the information to be retrieved, the more complex techniques have to be exploited such as Machine Learning (ML) algorithms [22,23].

ML has gained popularity as an effective technique in various fields. Several researchers have focused on developing smart tactile sensing systems based on ML algorithms [19,22,24,25]. In this context, ML algorithms such as Support Vector Machine (SVM) and Artificial Neural Networks (ANN) have proven to be effective for object recognition on tactile data collected from piezoelectric sensors [15,26].

Moreover, nowadays many companies provide on-demand network access to a shared pool of configurable computing resources with ML services on the cloud (such as google cloud [27]), allowing to run machine learning workloads on GPUs and TPU hardware accelerators using specific ML and AI software libraries (e.g., TensorFlow). However, cloud computing is often not the best solution for processing raw streaming data due to data loss, privacy, network downtime, energy consumption, and cost.

[*] Corresponding author.
E-mail addresses: youssef.amin@edu.unige.it (Y. Amin), christian.gianoglio@unige.it (C. Gianoglio), maurizio.valle@unige.it (M. Valle).

https://doi.org/10.1016/j.future.2023.06.002
0167-739X/© 2023 Published by Elsevier B.V.

Moreover, it is not feasible for tactile sensing applications due to time requirements (< 50 ms) [28]. Therefore, the general trend now focuses on embedding intelligence near the sensors by providing edge computing capabilities on the system's dedicated interface electronics. Moreover, due to the recent advancement in chip software and hardware designs, and VLSI integrated circuit technology used in micro-controllers manufacturing, embedded systems are now capable of supporting the more advanced machine and deep learning techniques [29,30]. Nevertheless, implementing complex algorithms that allow the processing of input signals from multiple sensors on resource-constrained devices, along with extremely tight latency requirements and low energy consumption, is a major challenge [31–40]. One of the solutions is TinyML [41], which enables intelligence with low memory, low power, and low latency, without the need for cloud support. These are the main requirements for a tactile sensing system that can be used in many wearables, prosthetics, and robotics applications.

This paper presents the implementation on the edge of algorithms for the processing of information extracted from tactile sensing arrays mounted on a Baxter robot. In particular, we addressed two hardness classification problems: (1) 5-class hardness classification of objects having the same shape, and (2) 5-class hardness classification of objects having a different shape. We designed pre-processing and ML algorithms that cope with the deployment on a resource-constrained device. We deployed the algorithms on edge electronics by achieving real-time performance, low energy consumption, and high classification accuracy in both problems. To the best of the authors' knowledge, this is the first paper addressing the embedded implementation of the whole data processing pipeline for the classification of objects' hardness through tactile sensing arrays mounted on a robot. The main contributions of this paper may be summarized as follows:

- we designed shallow ML algorithms for the deployment on resource-constrained devices;
- we designed energy efficient and low time latency pre-processing techniques to extract features from tactile signals and normalize data;
- we deployed pre-processing techniques and ML models, written in C language, on an edge device, namely STM32 Nucleo 745ZI-Q board;
- we employed a tactile sensing system, made of sensing arrays and embedded electronics, for two hardness classification problems, achieving high accuracy, real-time inference, low memory footprint, and low energy consumption.

The remainder of this paper is organized as follows: Section 2 reports related works, Section 3 the materials and methods, Section 4 the experimental setup, Section 5 the results and discussion, and Section 6 concludes the paper.

2. Related works

The research on object properties from tactile data in robotics ranges from the design of new sensors to the formulation of intelligent algorithms for extracting information effectively and efficiently. The authors in [34] presented an implementation of a smart tactile sensing system based on an embedded convolutional neural network (CNN) approach addressing a 22-class object recognition on a robot. Input data size was reduced as a way to optimize the proposed model, using three different input sizes. However, hardware implementation was carried on different powerful edge platforms which contain GPUs and/or powerful CPUs, where they compared the performance, time inference, and power consumption for each model on each hardware platform. Moreover, Amin et al. in [26], proposed the use of a learning strategy based on a loss function [42] that leads to finding the best configuration of the prediction model balancing the generalization performance and the computational cost of the whole elaboration system. To validate their work, the authors integrated a tactile sensing system on a Baxter robot to collect and classify data from five daily-life objects, using four different algorithms, while keeping track of the computational cost in terms of FLOPs. As a result, the SVM model achieved the best balance between accuracy and computational cost. However, when the computational cost was not relevant, the fully connected neural network achieved the best performance.

In [43], a piezoresistive Tekscan tactile sensing system was integrated into a robotic finger and pressure map images were collected upon grasping four objects of different shapes. The authors developed a novel algorithm to extract features from the pressure maps, which were used to train an artificial neural network (ANN) to classify the shape of objects. The model achieved a high success rate of 90%, but it was not implemented on constrained devices for real-time classification. Tao et al. [44] proposed a tactile sensing system based on piezoelectric sensors to detect the surface roughness of fruits and vegetables using a support vector machine (SVM) algorithm with a radial basis function kernel. In addition, deep learning (DL) algorithms namely CNN was employed for feature extraction from spatially distributed tactile sensors. The model achieved a 91.07% average classification accuracy while testing four different contact shapes.

A CNN-based model was trained in [45] to process the tactile information in order to enable successful in-grasp manipulation with untrained daily objects. As a result, CNN effectively handled the tactile information from uSkin sensors. Furthermore, tactile data from 241 distributed tactile skin sensors were used to train feed-forward and deep neural networks to generate a controlled in-hand manipulation of objects of different sizes and shapes [46]. In [47], a tiny CNN architecture was implemented on a Cortex-M micro-controller to classify touch modalities applied on an E-skin. The CNN model was optimized through layer fusion and buffer reuse strategies to speed up the inference on the edge devices. As a result, the CNN model achieved a real-time classification with low energy consumption and higher classification accuracy compared to other ML algorithms that were employed to classify the same dataset. In [19], the authors proposed a low-cost Arduino-based implementation for a real-time and highly accurate tactile texture classification of data from multiple tactile sensors using a random forest classifier. Twelve texture classes were used for the data collection and classification. A memory-efficient feature extraction method was proposed in order to achieve real-time processing of data compared to other time consuming feature extraction methods such as the Fourier transform. In [20], a tactile sensing system integrated into a robotic fingertip was proposed for object textures recognition. Tactile data generated from the PVDF piezoelectric film were used to feed different ML algorithms including SVM, KNN, and ANN. The results of the SVM model were promising giving classification rates higher than 90%.

In state-of-the-art, some works presented ML-based solutions for retrieving objects' hardness information from tactile sensing systems mounted on robots. One of many ways to create objects of different hardness is using a 3D printer with different filling percentages [48]. This method was utilized in [35] to print nine objects of different hardness levels by applying different combinations between three different filling percentages and two types of printing materials. An FPGA-based tactile system was mounted on a Cartesian robot, and tactile data were collected upon squeezing the objects. Two algorithms were considered for classification: K-medoids and KNN. Both algorithms achieved good classification accuracy reaching up to 86.7%. In [17], the authors proposed a CNN and recurrent neural network (RNN) to estimate the hardness of objects with different shapes using a GelSight tactile sensor. The sensor was integrated into a robot's fingertip and data was recorded upon grasping the object. The network was able to predict well the hardness of silicone samples with similar shape in the dataset, regardless of the loading conditions; whereas for objects with rigid surfaces, the model was not capable of estimating their hardness well. In [49], a decision tree and Naive Bayes methods were used to classify objects of different hardness. Data were collected from grasping objects using a two-fingered robotic gripper equipped with a tactile sensing system. Models were deployed into the PIC32 micro-controller to develop a real-time implementation for hardness classification. The decision tree outperformed the Naive Bayes model achieving 90% classification accuracy. In [50], piezoresistive tactile sensors were integrated into a robotic gripper to explore the material properties by squeezing the objects. Tactile information was represented over a time sequence of images that encode the pressure applied over the taxels. Each tactile image is an array of 64 values (8 *x* 8). They extracted as features the mean and standard deviation for each tactile image (one frame) to reduce the dimensionality of the data and use it to train a K-nearest neighbor (KNN) classifier to discriminate rigid and non-rigid objects [50,51]. Zhang et al. [13] integrated a tactile sensing system into a two-finger robotic gripper for the hardness classification of fruits and vegetables. The compression test technique was adopted to measure and label the hardness of the objects. Tactile sensors were mounted on both clamps of the gripper, and data were collected by grasping each object multiple times. Similar to [50], the mean and standard deviation were extracted to represent the features of the tactile signals. Two models, i.e. support vector machine (SVM) and K-nearest neighbor (KNN), were trained to solve a four-class hardness classification problem. As a result, the models achieved a high classification accuracy (94.37%) using simple features. The authors in [15] proposed a two stages processing unit for the elaboration and hardness classification of tactile data. A tactile sensing system was mounted to Baxter robot to collect data while grasping three objects of different hardness. For solving 3-class hardness classification problem, five different single-layer feed-forward neural networks (SLFNNs), namely fully connected neural network (FC) and extreme learning machine (ELM) [52], were implemented. In the end, the SLFNNs provided a good generalization performance (96.3%) keeping low the computational cost (<13 KFLOPs).

Only a few studies in the state-of-the-art have focused on hardness classification for robotic applications without addressing computational cost problem, although such processing algorithms could be computationally expensive. However, none of them have targeted embedded implementation of signal processing and classification of models on resource-constrained devices. Therefore, having a real-time smart tactile sensing system necessitates embedded implementation near the sensor location. In this context, a pre-processing strategy and ML algorithms were proposed to extract high-level information regarding object hardness from tactile data. This is challenging since processing models should have low memory footprint, latency, and energy consumption. Moreover, a comprehensive study was conducted to assess the performance of the proposed models on a commercial micro-controller Unit (MCU) STM32 board, in terms of accuracy, memory, latency, and energy consumption to achieve a real-time tactile sensing system for hardness classification.

3. Materials and methods

The aim of this paper is to build a real-time robotic tactile sensing system for hardness classification based on embedded ML implementation. Fig. 1 reports the diagram of the system

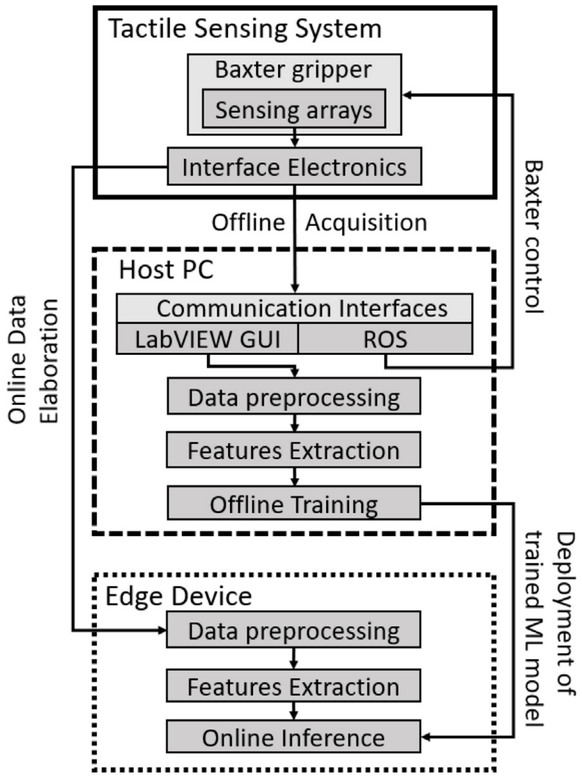


Fig. 1. Diagram of the information flow between system blocks and tasks implemented by each block.

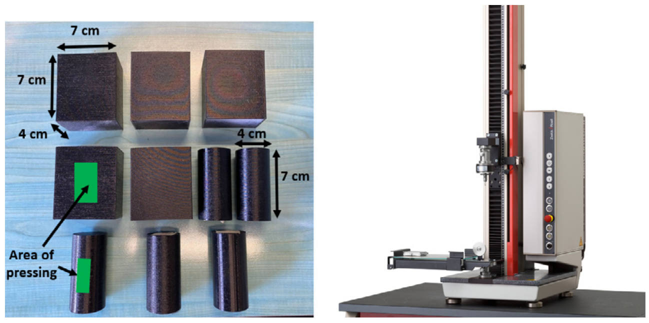
workflow divided into three blocks, highlighting the tasks implemented in each block. The first block represents the tactile sensing system made of sensor arrays, mounted on the clamps of the Baxter robot, and interface electronics (IE) to collect and filter the sampled signals. In the second block, a host PC controls the robot to perform the grasp through the robotic operating system and, by a LabVIEW GUI, retrieves and saves the data from the IE. Therefore, data are preprocessed, the features are extracted, and ML algorithms are trained offline. The trained models are then deployed on an edge device (third block) to provide online inference. Besides the models, the edge device hosts the data processing and features extraction. During the online inference, the host PC is no longer employed, except for the Baxter robot programming to perform the grasp actions. The system was tested on a use case addressing two classification problems: (i) 5-class hardness classification of 3D printed objects having the same shape, (ii) 5-class hardness classification of 3D printed objects having a different shape.

3.1. Materials

3.1.1. Objects selection and hardness level determination

Ten objects were designed and 3D printed (shown in Fig. 2(a)). Objects were made of two shapes: cubic and cylindrical (five objects for each shape). The size of the cubes is 7 x 7 x 4 cm. For the cylinders, the diameter and height are 4 cm and 7 cm, respectively. The objects were printed using different filling percentages, i.e. 3% – 5% – 7% – 10% – 12% for each shape, in order to obtain different hardness [48,53]. Filaflex material, i.e. Thermoplastic Polyurethane (TPU) that presents a large elasticity, was used to print the objects.

The literature presents different techniques to measure the hardness of an object. One of them is called the indentation technique [54], but it suffers from damage on the object's surface



a 3D Printed Objects **b** Zwick/Roell model Z0.5

Fig. 2. (a) 3D printed object samples in cubic and cylindrical shapes; (b) Materials testing machine used to apply compression tests to determine the hardness level of objects.

Table 1
Labels of printed objects based on the hardness value.

Samples	Hardness value	Hardness level
Cube 3%	4.13	0
Cylinder 3%	4.62	0
Cube 5%	8.52	1
Cylinder 7%	8.22	1
Cube 7%	12.21	2
Cylinder 10%	12.07	2
Cube 10%	27.19	3
Cube 12%	33.18	4

because it is subjected to a very large load. Another way to express the hardness value of an object is by measuring the maximum force obtained when compressing an object to a pre-defined displacement [13,55]. The compression technique was adopted in our experiments by using the Zwick/Roell machine model Z0.5 (shown in Fig. 2(b)) to measure the hardness value of the printed objects and provide effective labeling for their classification. For the experiments, the maximum displacement and velocity are set to 2 mm and 25 mm/sec, respectively, and pressure is applied at the center of the objects. For each object, 170 trials were performed.

The loading force-displacement curves corresponding to the compression tests are shown in Fig. 3. Figs. 3(a) and 3(b) present the hardness values of the cube and cylindrical objects, respectively, where each color represents an object of a certain filling percentage. The figures display two trials for each object. For objects of the same shape, differences in hardness values are noticed. In addition, some similarities in hardness are observed in Fig. 3(c) between objects of different shapes. The hardness values of the cubic objects with the following filling percentages: 3%, 5%, and 7%, are similar to the filling percentages 3%, 7%, and 10% of the cylindrical objects, respectively. Therefore, we selected all the cubic objects for the study and the three cylindrical objects that match the cubes' hardness.

The average hardness value of each object is computed over multiple compression trials. Finally, five categorical hardness levels were extracted: 0, 1, 2, 3, and 4. Table 1 reports the hardness measurements for the eight objects and the associated label.

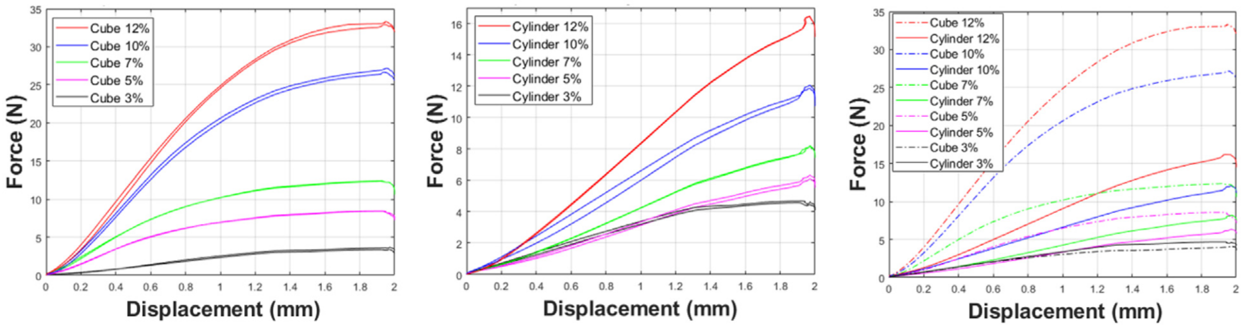
3.1.2. Tactile sensing arrays and acquisition system

The primary task was to integrate the tactile sensors into the gripper of a Baxter robot and collect tactile information from the objects' grasp. Two sensor arrays were employed, consisting of P(VDF-TrFE) piezoelectric sensing patches. This technology has already been exploited in previous works that dealt with robotic applications [14,15,26]. Different from the literature, the sensing patches in this work are smaller and have a higher spatial resolution: each patch consists of 8 sensors of 1 mm diameter each and 0.6 cm center-to-center pitch. Fig. 4 shows the structure of a sensing patch. In addition, these sensors have high-frequency bandwidth that ranges from 0.5 Hz to 1 kHz. For the experiments, the two sensing patches are shielded using a special conductive tape (Model tesa 60,262, tesa) in order to remove external changes by carrying them to the ground. Also, protective and substrate layers are added to the top and bottom sides of each patch, respectively. The readout circuit, tactile signal conditioning, and data acquisition are done by a low-power interface electronics (IE) equipped with ARM-Cortex M0 micro-controller and a DDC232 analog-to-digital converter with a sampling frequency of 2 KSamples/sec. The system, made of tactile patches and IE, is integrated into a Baxter robot (Fig. 5). The gripper's clamps were customized to fit the patches' size, while the IE was placed on the robotic arm near the sensors and fastened with an elastic fabric strap. Eventually, a host PC was used to control the displacement and grasping time of the gripper using the robotic operating system (ROS), while a LabVIEW GUI was designed for collecting, visualizing, and saving tactile data received from the IE in real-time.

3.2. Methods

3.2.1. Data collection and pre-processing of tactile information

We programmed the IE to acquire tactile data from 16 channels simultaneously and filter the data from each channel using a two-sample Moving Average filter (MA): $y[i] = \alpha\tilde{x}[i] + (1-\alpha)y[i-1]$; where α is a user-defined parameter and $\tilde{x}$ is the tactile signal. The Baxter robot was programmed to perform grasp-release actions on the objects by setting the gripper velocity



a Cubes Compression Curves **b** Cylinders Compression Curves **c** Cubes and Cylinders Compression Curves

Fig. 3. Loading force-displacement compression curves of cubes and cylinders.

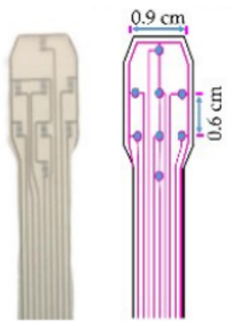


Fig. 4. Sensing patch structure.

and displacement of the manipulator to 5 mm/s and 2 mm, respectively. Tactile data were collected from the 16 channels while applying 170 grasp-release actions on each object. During each grasp-release action, the gripper closed the clamps for 1.2 s, then opened them for 2 s.

The piezoelectric PVDF sensors produce charges when subjected to mechanical stimulation. Therefore, as a response to grasping, the charge across a sensor decreases forming a negative peak, then the sensor's response return to its initial state as the gripper holds the object. When an object is released, the charge increases creating a positive peak. An example of sensors' response to a grasp-release action is shown in Fig. 6. In this figure, we present the response of 8 sensors within one sensing patch to the grasp-release event. All sensors in contact with an object exhibit the same behavior but with different amplitudes.

However, out of the signals we extracted only the grasp peaks to reduce the number of samples on which to compute the features for the next training procedure. Using Matlab, we implemented an automated pre-processing technique to extract N consecutive samples simultaneously from each of the 16 channels at the instance of grasp. The technique is based on a threshold allowing an easy implementation on the IE. In this way, the threshold acts as a trigger for the grasp signals, avoiding continuously analyzing the whole raw data even though a grasp action did not happen. Since each sensor can present a different offset due to fabrication, a threshold must be set for each one of them. As a result, the acquired data for each grasp action is a 2D tensor and can be formulated as $\mathcal{X} \in \mathbb{R}^{16\times N}$, where 16 is the number of sensors and N is the number of samples.

Table 2
Details of the six datasets built following the pre-processing of tactile signals generated upon grasping objects of different shapes and hardness.

Dataset name	Nb. of classes	Nb. of samples for each class	Total Nb. of samples
Cube_80	5	170	850
Cylinder_80	3	170	510
Cube_40	5	170	850
Cylinder_40	3	170	510
Merged_40	5	170	1360
Merged_80	5	170	1360

In particular, two values for N, i.e. $N = 40$ and $N = 80$, were chosen to investigate the effect of having a different number of samples on the classification accuracy and computational efficiency. Fig. 7 illustrates the pre-processing technique on one tactile signal from one channel.

3.2.2. Datasets

After the preprocessing of tactile information for all grasps performed on the eight objects, we obtained the datasets presented in Table 2. Cube_80 and Cylinder_80 correspond to the grasp peaks with $N = 80$ samples gathered from the cubes and cylinders, respectively. Similarly, Cube_40 and Cylinder_40 are obtained by the grasp peaks with $N = 40$ samples. Moreover, we built two datasets, namely Merged_80 and Merged_40, that were obtained by merging Cube_80 with Cylinder_80 and Cube_40 with Cylinder_40, respectively.

3.2.3. Feature extraction

To find a good representation of the tactile data and reduce its dimension, we extracted the mean (μ) and standard deviation (σ) from each grasp peak and use them as features. As a result, each grasp action $\mathcal{X}$ was transformed into a one-dimensional array $\boldsymbol{x} \in \mathbb{R}^{32\times 1}$ as sketched in Fig. 8. This array represents one sample of the datasets that will be used to train hardness classification models. In the following, the datasets presented in Table 2 from which the features were extracted will be named Cube_80_feat, Cylinder_80_feat, Cube_40_feat, Cylinder_40_feat, Merged_40_feat, Merged_80_feat, respectively.

3.2.4. Classifiers

For classifying the hardness level, we proposed three machine learning algorithms namely a Single-layer Feed-Forward Neural

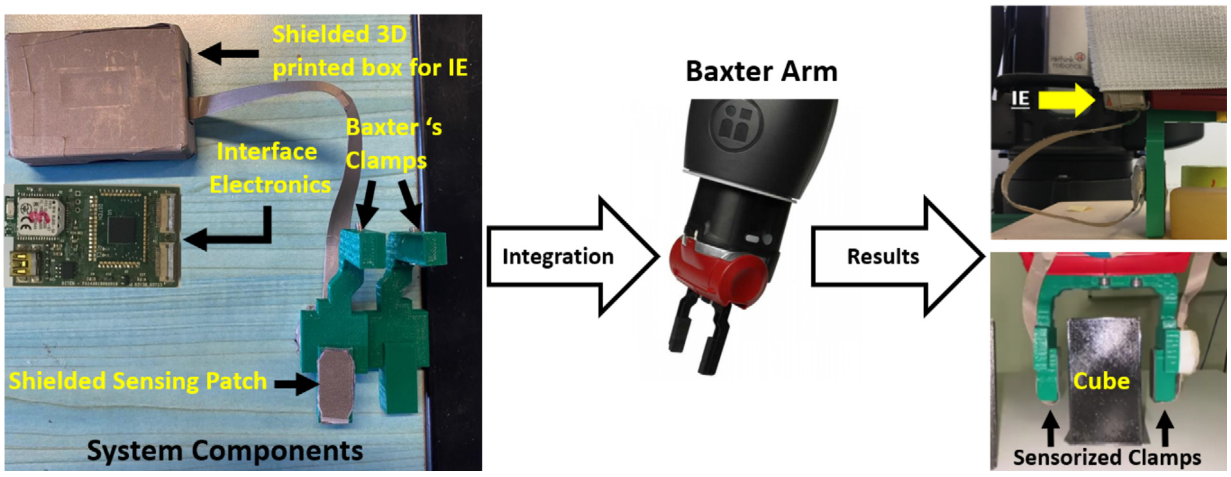


Fig. 5. Integration of the sensing patches and IE on the Baxter robot. On the left, the tactile sensing system made of the interface electronics and the sensing patch mounted on the 3D printed clamps. On the right, the outcome of the integration on the Baxter arm.

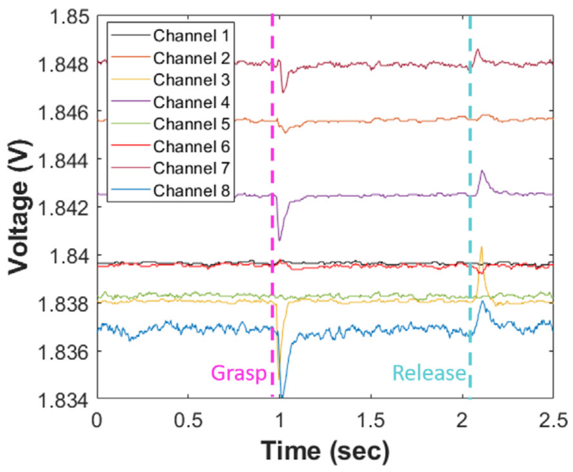


Fig. 6. Example of the response of eight piezoelectric sensors containing grasp and release events. Purple and cyan lines highlight the beginning of grasp and release events, respectively.

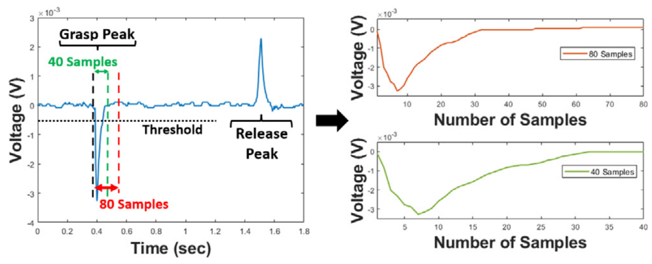


Fig. 7. Example of signal pre-processing on one channel. On the left, grasp samples ($N = 40$ and $N = 80$) are extracted when the signal crosses the threshold. On the right, in orange the grasp signal with $N = 80$ sample, and in green the grasp signal with $N = 40$.

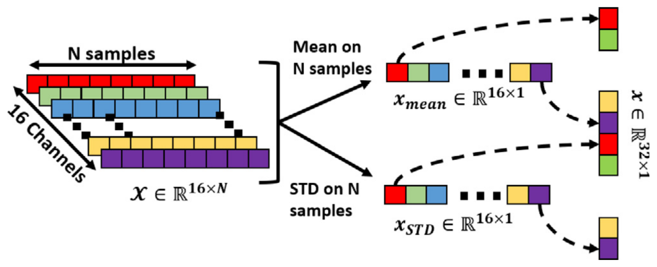

N samples
16 Channels
$x \in \mathbb{R}^{16\times N}$
Mean on N samples
STD on N samples
$x_{mean} \in \mathbb{R}^{16\times 1}$
$x_{STD} \in \mathbb{R}^{16\times 1}$
$x \in \mathbb{R}^{32\times 1}$

Fig. 8. Features extraction computing the mean and standard deviation from a data tensor of shape $16 \times N$, then reshaping the extracted features into a 1-D tensor to feed the ML algorithms.

Network (SLFNN), a Support Vector Machine (SVM), and a shallow Convolution Neural Network (CNN). These algorithms proved a good trade-off between generalization accuracy and computational cost in the classification of tactile data [15,26,34,56–58]. The SLFNN and SVM were trained using the datasets after the feature extraction. On the other hand, the CNN was trained directly on the tactile data obtained after the preprocessing phase. These algorithms were trained to solve two classification problems: (i) hardness level on objects with the same shape, and (ii) hardness level on objects with a different shape. For case (i), the cube datasets were employed, i.e. Cube_40 and Cube_80 for the CNN, Cube_40_feat and Cube_80_feat for SLFNN and SVM. While for case (ii) the merged datasets were adopted, i.e. Merged_40 and Merged_80 for CNN, Merged_40_feat and Merged_80_feat for SLFNN and SVM. In the following, the algorithms are briefly described.

Support vector machine

The Support Vector Machine (SVM) classifier is a supervised ML algorithm that computes the hyperplane that maximizes the margin to the nearest samples of the two classes (i.e., the support vectors). SVM is capable of capturing complex relationships between data points. We have implemented only the linear kernel due to its efficiency in terms of memory and computation [59]. The inference of one input datum $\boldsymbol{z}$ follows the One-vs-One (OvO) strategy that splits the multi-class classification into one binary classification problem per each pair of hardness levels, solving (1):

$$y = sign(\boldsymbol{w} \cdot \boldsymbol{z} + b) \tag{1}$$

where $\boldsymbol{w}$ and b are the support vector and bias, respectively. The eventual label of $\boldsymbol{z}$ is assigned according to the majority of votes among the predicted classes.

Single-layer Feed-Forward Neural Network

Single-layer Feed-Forward Neural Networks (SLFNNs) are fully connected networks that are trained through the backpropagation technique. These models mimic the structure of the human brain's neural network such that all neurons in one layer are connected to the neurons in the next layer. It has proven to be efficient in terms of computational efficiency and modeling multi-class hardness classification problems [15,26]. The prediction function of an SLFNN is:

$$f(\boldsymbol{z}) = softmax\left(\sum_{i=1}^{Neu} \beta_{i,j}\phi(\boldsymbol{z}\cdot\boldsymbol{w}_i + b_i) + B_{i,j} \text{ with } j = 1,\ldots,5\right) \tag{2}$$

where $\boldsymbol{z}$ the tested datum, Neu is the number of hidden neurons, $\beta_{i,j}$ and $B_{i,j}$ are the weights and biases between the hidden and output layer, respectively, and ϕ is the ReLU activation function. $\boldsymbol{w}$ and b_i are the weights and biases between the input and hidden layers, and *softmax* is the Softmax function to predict the label. The output layer contains 5 neurons, as the number of hardness classes, indexed by j in the equation.

Convolutional neural network

A Convolutional Neural Network (CNN) is one of the most popular deep neural networks used in a multitude of applications. CNN is composed of different building blocks such as convolutional, pooling, and fully connected layers. Unlike the SLFNN and SVM which use hand-crafted features, CNNs are able to combine feature extraction and classification into a single learning body. Moreover, CNNs proved their effectiveness when applied to tactile data decoding [34,56]. In this work, the tactile signals are used to train a shallow 1-D CNN. The inputs of this network are the grasp peaks collected in the datasets presented in Table 2. The CNN consists of several functional blocks composed of one 1-D convolutional with ReLU activation, a dropout, and an average pooling layer. The blocks are stacked sequentially based on the number of convolutions chosen by the designer. In this work, the number of blocks was set through the *filter* parameter listed below. Stacked at the bottom of the functional blocks, one 10 neurons dense layer with the ReLU activation and a dense layer with the softmax activation function provide the classification. The 10 neurons dense layer receives the features extracted by the functional block and flattened by means of a *Flatten* layer. Other deep learning models are much less efficient for resource-constrained implementation, requiring a much higher number of computations due to their complexity as recently demonstrated in [47,56,60].

3.2.5. Embedded implementation

In order to test our models on an embedded device, we implemented using C language on an STM32 microcontroller [61] a set of functions that provide the fundamental layer functionalities of the algorithms. The implementation relies on the C standard libraries and is portable to any device that supports C code. Besides the classifiers, the feature extraction and normalization stages were implemented on the edge as well. In the following, the parameters of feature extraction, normalization, and algorithms that were saved on the STM32 microcontroller are listed. To evaluate the performance of the deployed models, we computed the accuracy of the test sets, the memory footprint of the pre-processing and classification algorithms, the inference time, and the energy consumption.

Feature extraction and normalization memory footprint

Before the online inference, we performed the same feature extraction and normalization steps on the input data as for the offline training phase (see Section 3.2.3). The features are computed online when SLFNN and SVM classifiers are employed, thus they do not require any storage of parameters. Afterward, the normalization is computing on the features and on the channels in the case of CNN. As previously mentioned, the MinMax scaler was adopted to normalize data in the range [0, 1] as follows:

$$x_{norm} = \frac{x - xmin}{xmax - xmin}. \tag{3}$$

where *xmax* and *xmin* are the maximum and minimum values of features or channels, respectively. Thus, *xmin* and *xmax* computed on the training data for each feature and channel were stored on the device for the online normalization of data. As a result, 32 *xmin* and *xmax* values were saved for SLFNN and SVM, while 16 values were for CNN.

SVM device memory footprint

The $\boldsymbol{w}$ and b parameters of each trained SVM binary (1) classifier were stored in the device. Since the OvO training strategy was exploited and the number of hardness classes is 5, the number of binary classifiers is 10 resulting in 10 biases b saved in the memory of the STM32. Moreover, since an input datum is represented by 32 features, 320 values for $\boldsymbol{w}$ parameters were deployed on the device. In the case of linear SVM, the number of parameters (and bytes) is equal for each hardness classification problem addressed in this paper.

SLFNN device memory footprint

According to (2), Neu β weights and B biases were saved on the device for each class. Moreover, $32 * Neu$ $\boldsymbol{w}$ weights and Neu b biases were stored as well. The amount of parameters depends on the best configuration of neurons Neu found during the training procedure. Formalizing, the number of parameters for SLFNN can be computed as $N_{params} = Neu * (nfeat + 1 + nclasses) + nclasses$, where $nfeat$ is the number of features and $nclasses$ the number of classes.

CNN device memory footprint

The number of parameters stored in the device for the CNN depends on the number of functional blocks, the number of filters in each block, and the kernel size. These hyper-parameters were chosen during the training procedure. The number of parameters for each functional block can be computed as $nchannels * ker\_size * filt + filt$, where $nchannels$ represents the number of output filters of the previous block (in the case of the first block $nchannels$ corresponds to the channels of the input, i.e. 16), ker_size the kernel size, and $filt$ the number of output filters. Additionally, $Neu * (nfeat + 1 + nclasses) + nclasses$ parameters for the fully connected layer stacked to the functional blocks were saved on the device as well, where $nfeat$ represents the number of features extracted by the last functional block.

Inference time

The inference time is defined as the time interval that the edge device requires to output a classification result after it receives an input datum. The STM32 hardware abstraction layer (HAL) provides the HAL_GetTick() function that measures the elapsed time in milliseconds [47]. Thus, it can be used to compute the inference time on the microcontroller by calculating the difference between a new input datum read and the previous one.

Energy consumption

The energy consumption was computed based on the following equation:

$$E = P \cdot t \tag{4}$$

where P is the power consumption retrieved by multiplying the voltage with the average Current provided by STM32CubeIDE software [62], and t is the inference time.

Optimization using memory caching

Memory Caching optimization technique was adopted to decrease the inference time. It is based on cache memory which is a small and fast temporary storage area close to the CPU that can be found in computers/microcontrollers. It allows the processor to retrieve data faster than accessing the DRAM memory, thus providing a more efficient, easy, and near-instant data retrieval solution. In general, there are three cache levels (i.e., L1, L2, and L3) that identify increasing storage capacity and distance from the CPU. The STM32H7 series devices contain only an L1-cache which is small but extremely fast. It provides an optional L1-cache for the data (D-cache) and the instructions (I-cache), with up to 16 Kbytes per type [63]. L1-cache stores a set of instructions and data near the CPU to prevent frequently used instruction/data from being fetched multiple times from the DRAM by the CPU. Moreover, the bus accesses to the subsystem memory, which takes more than one CPU cycle to execute, are different from the CPU pipeline instruction stream execution [64]. Therefore, cache memory is meant to speed up the read/write operations by having the data locally available thus accessing them in only one clock cycle, providing a huge raise in performance especially if the model is small enough to fit entirely in the L1 cache. Therefore, the CPU I-cache and D-cache were enabled for a performance improvement in terms of latency [65].

4. Experimental setups

4.1. Algorithms hyper-parameters

The training procedure involved model selection, i.e. the tuning of the classifier architecture hyper-parameters. That procedure explored a grid of candidates for each one of the algorithms. The best candidates were selected by evaluating the accuracy of the validation set during the training phase.

The hyper-parameter for the SVM was the regularizer λ that has been chosen as the norm L2 in the range $\lambda = [10^i, with\ i = -4, -3, \dots, 4]$ during the training procedure.

The hyper-parameters for the SLFNN were:

- hidden neurons $Neu = [10 * i, with\ i = 1, 2, \dots, 10]$;
- L2 regularizer $\lambda = [10^i, with\ i = -4, -3, \dots, 4]$.

The hyper-parameters for the CNN were:

- number of convolutional layers from 2 to 4 (the filter candidates were: (8, 8), (16, 16), (16, 32), (4, 8, 16), (8, 8, 8), (8, 16, 32), (4, 8, 16, 32));
- kernel size $Ks = \{8, 12, 16\}$;
- dropout percentage 20%.

Table 3
Training and test splits for the two classification problems.

Classif.	Class	Train samples		Test samples	
problems		Cubes	Cylin	Cubes	Cylin
Cubes	1	120	–	50	–
	2	120	–	50	–
	3	120	–	50	–
	4	120	–	50	–
	5	120	–	50	–
Merged	1	80	40	90	130
	2	80	40	90	130
	3	80	40	90	130
	4	120	–	50	–
	5	120	–	50	–

The filters represented the number of kernels applied to the input features in each functional block. For example, setting $f = (4, 8, 16)$ implied the use of three functional blocks, the application of 4 kernels to the input tensor, the dropout, the average pooling, and the doubling in the number of kernels at each next functional block.

4.2. Training strategy

To compare the performance of the algorithms on the two hardness classification problems described in Section 3.2.4, all the models were trained on 4 datasets: Cube_40_feat, Cube_80_feat, Merged_40_feat, Merged_80_feat for SLFNN and SVM, while Cube_40, Cube_80, Merged_40, Merged_80 for CNN. The datasets were randomly split into training, validation, and testing sets. More precisely, in the classification problem of the cubic objects, the datasets contain 850 data that were split as follows: 70% of data for training (i.e., 120 data per class), and 30% for testing (i.e., 50 data per class). Whereas, in the case of the merged datasets, which contain a total of 1360 samples, the data splitting was done as follows: we randomly extracted 40 samples from each of the three-cylinder classes that match the cube ones (Section 3.1.1), and added them to the cube training set. As a consequence, we removed 40 random cubes from the training set and considered them in the test set. The remaining 130 cylinder samples per class were added to the cube test set. As a result, the merged training set consists of 120 cylinders (40 per class) and 480 cubes (80 for the three matching hardness levels and 120 for the other two), while the merged test set consists of 390 cylinders (130 per class) and 370 cubes (90 for the three matching hardness levels and 50 for the other two). Table 3 summarizes the training and test splits for both classification problems with respect to each hardness class. Before the training of SLFNN and SVM, the features of the datasets were normalized in the range [0, 1] using the MinMax Scaler. In the case of CNN, the data were normalized along the channels in the range [0, 1]. A grid search technique was employed to find the best configuration of hyperparameters for all the models. The best model was chosen by evaluating the accuracy of the validation split randomly extracted from the training set (20% of data were used for validation). For the SLFNN and CNN other parameters were set as follows: Adam optimizer with a learning rate $lr = 10^{-3}$, batch size $bs = 64$, and number of epochs $ep = 100$. Moreover, a patience $p = 8$ was set to implement an early stop criterion on the validation accuracy. The three algorithms are built using Python with the Keras library. At the end of the training, the best model for each algorithm was deployed on the STM32 microcontroller for the evaluation of the performance on the edge.

Table 4
SVM accuracy.

Datasets	Cubes	Cylinders	Overall
Cube_80_feat	100	–	–
Cube_40_feat	99.6	–	–
Merged_80_feat	99.2	98.21	98.59
Merged_40_feat	98.4	94.87	96.26

Table 5
SLFNN accuracy.

Datasets	Cubes	Cylinders	Overall
Cube_80_feat	100	–	–
Cube_40_feat	99.6	–	–
Merged_80_feat	100	98.71	99.38
Merged_40_feat	100	95.90	97.50

5. Results and discussion

We assessed the performance of the proposed hardness classification tactile sensing system on the commercial Microcontroller Unit (MCU) STM32 NUCLEO H745ZI-Q board in terms of accuracy, memory, latency, and energy. This board hosts an ARM Cortex-M7 core running at 480 MHz, with 2 MB flash memory and 1 MB SRAM.

5.1. Accuracy

As described in Section 3.2.4, the linear SVM, SLFNN, and the 1D-CNN algorithms were trained on four different datasets, obtaining four models for each algorithm.

Table 4 shows the classification accuracy of the SVM models on the test sets. The first column reports the datasets, the second the accuracy on the cubes object, the third the accuracy on the cylinders, and the last the overall accuracy. The accuracy of the cylinders and the overall one were computed only in the case of the merged datasets since they contain both objects. As a result of reducing the number of samples used to represent a grasp signal from $N = 80$ to $N = 40$, the classification accuracy slightly decreases in both problems. Moreover, for the merged datasets the accuracy on the cubes is higher than on the cylinders, probably because only a small subset of cylinders was employed to train the models.

Table 5 presents the classification accuracy of the SLFNN models. The models achieve a 100% accuracy on the cubes except for Cube_40_feat (99.6%). Concerning the merged datasets, the SLFNN presents a similar trend to the SVM: the cubes' accuracy is higher than the cylinders' one. Moreover, by reducing the number of samples to represent a grasp signal, the drop in accuracy is small. In general, the SLFNN outperforms the SVM when the merged datasets were employed, meaning that it is more suitable to classify the hardness levels when objects of different shapes are grasped. The best SLFNN models are shown in Fig. 9. The models trained with the datasets containing only the cubes reach out for a simpler solution, i.e. with a low number of neurons in the hidden layer. On the opposite, the models trained with the merged datasets required a greater number of neurons, thus increasing the inference time, the memory footprint, and the energy consumption.

The classification accuracy of CNN models is reported in Table 6. The accuracy on only cube samples (i.e., 100% and 99.6%) is the same of SVM and SLFNN architectures. On the other hand, CNN achieves the best overall accuracies on the merged datasets, presenting the highest performance in the classification of the cylindrical objects. As for the other classifiers, the classification accuracy of Cube_40 and Merged_40 is lower than Cube_80 and

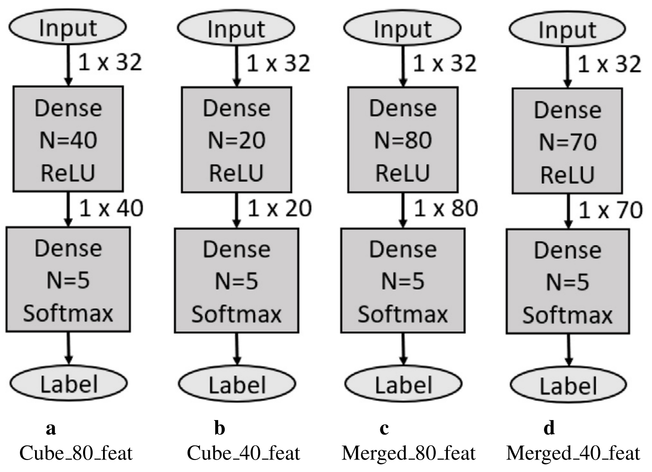


Fig. 9. Best SLFNN models: (a) Cube_80_feat, ()b) Cube_40_feat, c) Merged_80_feat, and d) Merged_40_feat.

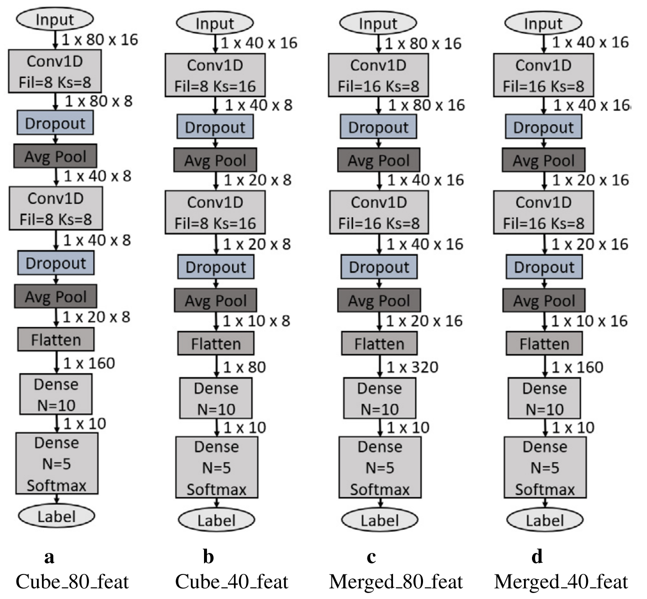


Fig. 10. Best CNN models: (a) Cube_80_feat, (b) Cube_40_feat, (c) Merged_80_feat, and (d) Merged_40_feat.

Merged_80, respectively. However, the drop in accuracy for the merged dataset is lower with respect to the other two classifiers.

The best CNN models are presented in Fig. 10. All the CNN best models consist of two functional blocks (a 1-D convolutional with ReLU activation, a dropout, and an average pooling layer) and two fully connected layers, where the first one has the ReLU activation function and the second the Softmax activation function providing the classification. The models for the cubes classification present 8 filters in both convolutional layers, while the models that classify both shapes have 16 filters in both layers, thus presenting a higher number of parameters, that in turn leads to a higher inference time, memory occupation, and a energy consumption.

To summarize, for models trained with only the cubes, all algorithms achieved the same classification accuracy, i.e. 100% for Cube_80(_feat) and 99.2% for Cube_40(_feat). Whereas, for the models trained with both objects, CNNs outperform SLFNN and SVM by achieving higher classification accuracy on cylinders. Moreover, in all cases, it is observed that the accuracy decreases

Table 6
CNN accuracy.

Datasets	Cubes	Cylinders	Overall
Cube_80	100	–	–
Cube_40	99.6	–	–
Merged_80	100	99.23	99.53
Merged_40	99.2	97.17	98.13

Table 7
SVM memory footprint on the edge device.

Datasets	Pre-proc	Classifiers	Total
Cube_80_feat			
Cube_40_feat	64	330	394
Merged_80_feat	(256)	(1320)	(1576)
Merged_40_feat			

Table 8
SLFNN memory footprint on the edge device.

Datasets	Pre-proc	Classifiers	Total
Cube_80_feat	64 (256)	1525 (6100)	1599 (6396)
Cube_40_feat	64 (256)	765 (3060)	829 (3316)
Merged_80_feat	64 (256)	3045 (12180)	3109 (12436)
Merged_40_feat	64 (256)	2665 (10660)	2729 (10916)

when the number of samples for the grasp signals halves from $N = 80$ to $N = 40$. The high classification accuracy is probably due to the number of sensors exploited in this study: in a previous work [15], 96.3% accuracy was achieved by a SLFNN classifying 3-class objects' hardness using 300 neurons and four tactile sensors.

5.2. Memory footprint

As described in Section 3.2.5, to normalize the input samples during the online pre-processing stage, *xmin* and *xmax* parameters in (3) must be deployed on the edge device for each feature in case of SVM and SLFNN, or for each CNN input channel. As mentioned, the linear SVM classifier trained with the OvO strategy and with 5 classes requires 320 $\boldsymbol{w}$ and 10 biases being stored on the device. Table 7 reports the memory footprint of the SVM classifiers. The first column shows the training datasets, and the second, third, and last columns report the number of parameters stored in the memory of the edge device and the number of bytes between the brackets for the pre-processing, the classifier, and the total amount, respectively, for each one of the training datasets. The parameters were represented as 32-bit floating point numbers.

Table 8 shows the memory footprint of the SLFNNs, maintaining the same format of Table 7. The pre-processing phase for SLFNN models is the same one used for SVM models, thus requiring the same memory size: On the other hand, the number of parameters in the classifiers depends on the number of hidden neurons as described in Section 3.2.5. According to the table, the size of the model trained with Cube_40_feat is approximately half of the one trained with Cube_80. In contrast, the size of the model trained with Merged_40_feat is only 12% smaller compared to the one trained with Merged_80_feat. It is straightforward that the size of cubes-based models is significantly lower than the size of the merged-based models since the lasts require more hidden neurons to solve a more complex problem.

Similar to the SLFNN, Table 9 reports the memory footprint for the 1-D CNN models. The MinMax normalization was performed on the tactile data across the 16 channels, hence the total number of *xmin* and *xmax* values is 32. Decreasing the input size resulted in an increase in the size of the model trained with cubes of about 22%. This is possibly due to the higher number of filters needed

Table 9
CNN memory footprint on the edge device.

Datasets	Pre-proc	Classifiers	Total
Cube_80	32 (128)	3217 (12868)	3249 (12996)
Cube_40	32 (128)	3953 (15812)	3985 (15940)
Merged_80	32 (128)	7393 (29572)	7425 (29700)
Merged_40	32 (128)	5793 (23172)	5825 (23300)

Table 10
SVM inference time including pre-processing.

Datasets	Pre-proc (ms)	Classifiers (ms)	Total (ms)
Cube_80_feat	0.138	0.03	0.168
Cube_40_feat	0.074	0.03	0.104
Merged_80_feat	0.138	0.03	0.168
Merged_40_feat	0.074	0.03	0.104

Table 11
SLFNN inference time including pre-processing.

Datasets	Pre-proc (ms)	Classifiers (ms)	Total (ms)
Cube_80_feat	0.138	0.126	0.264
Cube_40_feat	0.074	0.063	0.137
Merged_80_feat	0.138	0.226	0.364
Merged_40_feat	0.074	0.171	0.245

to extract more features from a smaller input. On the opposite, the number of parameters in the model trained with *Merged_40* decreased by about 22%.

In summary, the linear SVM models outperform the SLFNN and 1D-CNN models in terms of memory requirements for both classification problems. The CNN, which achieved the best performance in classifying the hardness of objects with different shapes, presents the worst solution in terms of memory occupation for all four datasets. Straightforwardly, the SLFNN could represent an alternative choice since it attained a higher accuracy than the SVM, with a lower memory allocation than the CNN.

5.3. Inference time

For the SVM algorithm, Table 10 reports the inference time including the pre-processing. The pre-processing corresponds to the time measured for feature extraction and data normalization. The classification time for all SVM models is equal and requires only 0.03 ms, while the pre-processing affects most of the inference time and depends on the input size. In fact, when the size of the input data is reduced by half, the pre-processing time decreases from 0.138 ms to 0.074 ms. In general, the inference time is lower than 1 ms, accomplishing real-time performance.

In the case of SLFNN (Table 11), the pre-processing time is the same as the SVM. Unlike the SVM, the classification time is not negligible and it differs from one model to another due to the different number of neurons. The inference time in cubes and merged classification problems decreased by approximately 48% and 33%, respectively, when reducing the input size from $N = 80$ to $N = 40$. In the case of merged datasets, the inference time is lower than 1 ms as SVM, thus SLFNN represents a valuable classifier outperforming the SVM in terms of accuracy.

Finally, the inference time of CNN models is shown in Table 12. Unlike SVM and SLFNN, the pre-processing time for CNN input samples corresponds only to the MinMax normalization. However, pre-processing time for both input sizes is negligible with respect to the whole inference. Nevertheless, reducing the input size played an important role in decreasing the inference time by about 50% both for cubes and merged classification problems. In general, the inference time of the CNN models is above 4 ms and 7 ms in the case of cubes and merged datasets, respectively.

Table 12
CNN inference time including pre-processing.

Datasets	Pre-proc (ms)	Classifiers (ms)	Total (ms)
Cube_80_feat	0.318	8.513	8.831
Cube_40_feat	0.019	4.429	4.448
Merged_80_feat	0.318	14.723	15.041
Merged_40_feat	0.019	7.279	7.298

Table 13
Energy consumption per inference.

Datasets	SVM (μJ)	SLFNN (μJ)	CNN (μJ)
Cube_80(_feat)	12.51	19.65	657.82
Cube_40(_feat)	7.74	10.21	331.33
Merged_80(_feat)	12.51	27.04	1120.40
Merged_40(_feat)	7.74	18.25	543.63

Even if these times cope with a real-time application on the STM32 Nucleo, it is worth noting that deploying the CNN in a more resource-constrained device with a less-performing CPU, the real-time inference could be not guaranteed.

To summarize, the SVM models achieved the lowest inference time for both hardness classification problems. Moreover, by reducing the data size to $N = 40$, the inference time decreased by 38% in SVM, 48% (cubes datasets) and 33% (merged datasets) in SLFNN, about 50% for both datasets in CNN. For the hardness classification problem of objects with different shapes, the SVM models are more than 2x faster than the SLFNN models, and up to 70x faster than CNN models. Nevertheless, the total time latency of the SVM and SLFNN models is substantially low (< 1 ms) compared to the CNN models (up to 15 ms). Nonetheless, all algorithms met real-time requirements on the STM32 Nucleo.

5.4. Energy consumption

Table 13 shows the energy consumption of the three models with respect to the datasets. As expected the energy consumption (4) for the SVM models is the lowest due to the fast inference time. SLFNN shows a slight increase in energy consumption compared to the SVM due to higher latency. While the CNN models require at least an order of magnitude higher energy consumption with respect to the other models. Again, this result is expected because of the higher inference time of CNN models compared to the others (Table 12). Nevertheless, reducing the input size from $N = 80$ to $N = 40$ has also led to a reduction in energy consumption for all models.

5.5. Optimization

Table 14 shows the improvements in time latency performance reached by means of the optimization technique discussed in by means of the cache memory. The table shows for each algorithm the inference time and the difference with the inference without the optimization expressed in ms and as a percentage. For the cube classification problem, SLFNN presents a similar inference time to the SVM, while for the merged datasets it has a gap of about 0.03 ms. It is worth noting that the inference time for both classifiers is always below 0.1 ms except for SLFNN classifying the Merged_80_feat dataset. Even CNN presents a remarkable improvement in the inference time while using the cache memory: for the classification of the cubes, CNN presents an improvement of more than 70% with an inference time lower than 2 ms, while for the merged datasets the inference time is reduced by more than 65% achieving about 4 ms inference in case of Merged_80 and 2 ms with Merged_40. In general, the memory caching technique avoids the CPU to accesses many times to the

Table 14
SVM, SLFNN, and CNN inference time with memory caching optimization.

Datasets	SVM		SLFNN		CNN	
	Inference (ms)	Difference (ms)	Inference (ms)	Difference (ms)	Inference (ms)	Difference (ms)
Cube_80(_feat)	0.077	−0.091 (−54%)	0.089	−0.175 (−67%)	1.772	−7.059 (−80%)
Cube_40(_feat)	0.045	−0.059 (−57%)	0.047	−0.090 (−66%)	1.227	−3.221 (−72%)
Merged_80(_feat)	0.077	−0.091 (−54%)	0.111	−0.253 (−70%)	4.105	−10.936 (−73%)
Merged_40(_feat)	0.045	−0.059 (−57%)	0.073	−0.172 (−70%)	1.996	−5.302 (−73%)

Table 15
Energy consumption of the models with memory caching optimization.

Datasets	SVM		SLFNN		CNN	
	Energy (μJ)	Difference (μJ)	Energy (μJ)	Difference (μJ)	Energy (μJ)	Difference (μJ)
Cube_80(_feat)	5.74	−6.77 (−54%)	6.63	−13.02 (−67%)	132.00	−525.82 (−80%)
Cube_40(_feat)	3.35	−4.39 (−57%)	3.50	−6.71 (−66%)	91.40	−239.93 (−72%)
Merged_80(_feat)	5.74	−6.77 (−54%)	8.27	−18.77 (−70%)	305.78	−814.62 (−73%)
Merged_40(_feat)	3.35	−4.39 (−57%)	5.74	−12.51 (−70%)	148.68	−394.95 (−73%)

Table 16
SLFNN and CNN inference time after the deployment with X-Cube-AI tool.

Datasets	SLFNN			CNN		
	Inference (ms)	Diff Not Optim (ms)	Diff Mem Optim (ms)	Inference (ms)	Diff Not Optim (ms)	Diff Mem Optim (ms)
Cube_80(_feat)	0.09	−0.174 (−65%)	+0.001 (+1%)	1.568	−7.263 (−82%)	−0.204 (−11%)
Cube_40(_feat)	0.051	−0.086 (−62%)	+0.004 (+9%)	1.205	−3.243 (−73%)	−0.022 (−2%)
Merged_80(_feat)	0.109	−0.255 (−70%)	−0.002 (−2%)	3.083	−11.958 (−80%)	−1.022 (−25%)
Merged_40(_feat)	0.072	−0.173 (−71%)	−0.001 (−1%)	1.453	−5.845 (−80%)	−0.543 (−27%)

DRAM, thus saving many clock cycles during the inference of a tactile datum. Since time and energy are directly proportional (4), the improvement in latency performance results in a significant reduction in energy consumption as can be seen in Table 15. The percentages of energy consumption reduction are similar to the inference time ones.

For the sake of comparison, the models were optimized and deployed by means of the X-Cube-AI tool provided by STM [66]. The deployment of the SVM models is not supported by the tool, thus only the results concerning the SLFNN and CNN are presented. Table 16 shows the inference time results and has the same structure as Table 14. The table presents not only the difference in the inference time with the not optimized models but also with the optimized ones by adopting the memory caching technique. Concerning the SLFNN models, models supported by X-Cube-AI are up to faster than the non-optimized C written models, providing up to 70% inference time reduction. However, compared to optimized models written in C language, X-Cube-AI models achieved slightly higher inference time. Whereas in the case of CNN, the inference time of CNN models deployed with X-Cube-AI is significantly lower than the non-optimized C written models for both classification problems (in between 73% and 82% inference time reduction). and slightly lower than CNN models optimized with memory caching in the case of merged datasets especially Cube_40 (only 2% inference time reduction). On the other hand, the X-Cube-AI library is only supported by STM devices, thus not providing a fair comparison in case of deployment on other edge devices.

5.6. Conclusive remarks

The experiments revealed that we are able to achieve high classification accuracy for both hardness classification problems using any of the proposed algorithms. However, CNN models achieved the best accuracy for hardness classification on the merged datasets. Whereas, in terms of memory, latency, and energy consumption, SVM models outperformed significantly CNN models, and slightly SLFNN models. Furthermore, reducing the size of the input data has resulted in remarkable improvements in the efficiency of models in terms of memory requirements (up to 46.7% reduction), latency (up to 50%), and energy (up to 50%), with only slight reduction in accuracy (0.4% on the cubes and a maximum of 2.3% in Merged models). In order to further improve the models' implementation, we adopted the memory caching technique. As a result, we achieved an additional reduction in latency and energy consumption. It is important to note that the achieved results enable the use of these models on extremely resource-constrained devices such as the Cortex-M0 family [67].

6. Conclusion

In this work, we presented the implementation on the edge a real-time tactile sensing system for hardness classification based on machine and deep learning algorithms. We developed and implemented in plain C a set of functions that provide the fundamental layer functionalities of the linear SVM, SLFNN, and 1-D CNN models, along with the pre-processing to extract the features and normalize the data. Furthermore, the implementation does not rely on any of the existing libraries and therefore it is deployable to any device that supports C code.

To evaluate our work we mounted the tactile sensing system onto a Baxter robot and collected data by grasping objects of different hardness and shape. Two classification problems were addressed: 5 levels of hardness classified on the same objects' shape, and 5 levels of hardness classified on two different objects' shape. Pre-processing techniques were employed for extracting the features and normalizing the data. The models and pre-processing were implemented on STM32 NUCLEO H745ZI-Q board which hosts an ARM Cortex-M7, where we assessed the performance of the system in terms of accuracy, memory footprint, time latency, and energy consumption. All the models presented a high classification accuracy of close to 100% in the first problem and above 96% in the second problem. Reducing the input size has led to a drastic drop in inference time and energy consumption with a slight deterioration of the accuracy in all the models. We also showed that inference time and energy consumption can be further improved using a memory caching optimization strategy with a reduction of the two quantities of up to 80% for the CNN. Eventually, SVM models have proved to be the best models in terms of memory requirements, time latency, and energy consumption, whereas in terms of accuracy the CNN achieved the highest values. On the other hand, SLFNN provided a trade-off between accuracy on one side and latency time, energy consumption, and memory requirements on another, by achieving slightly lower accuracy than CNN models and slightly higher latency time, energy consumption, and memory requirements than SVM models. Eventually, this work demonstrated the feasibility of integrating fast, small, accurate, and energy-efficient machine learning models on a resource-constrained device, in order to provide a real-time robotic tactile sensing system for object hardness classification. This also paves the way for developing embedded tactile sensing systems for a variety of robotic and prosthetic applications.

In future works, we aim to develop an automatic calibration algorithm for the sensors due to the observed variation in responses among different sensing patches and even within the same patch. This variability is attributed to the fabrication process of the piezoelectric sensors. By implementing an automatic calibration algorithm, we will be able to address these differences in sensor responses, especially when using different patches.

Furthermore, during the testing, we noticed a significant decline in hardness classification accuracy when evaluating our models on cylindrical objects, which were originally trained on cubic objects. However, we were able to mitigate this issue to some extent by including a few cylindrical samples in the training set. Nevertheless, in-corporating additional shapes such as spheres, cones, and so on, as well as accommodating different hardness levels, would require a considerable amount of time and effort. To overcome these challenges, we propose the use of a soft gripper instead of a rigid one in future experiments. Employing a soft gripper could help alleviate problems related to changes in object shape, thereby improving the classification performance across different shapes. Moreover, we are considering the development of semi-supervised or unsupervised strategies to identify different hardness levels including al-so new shapes. By leveraging these strategies, we aim to reduce the number of samples required for retraining the model to classify new hardness levels. In the case of unsupervised classification, the models would not need to be retrained, which could save both time and effort.

In summary, we plan to address the sensor response variation through an automatic calibration setup, mitigate the impact of object shape changes by using a soft gripper, and explore semi-supervised or unsupervised strategies for identifying different hardness levels including also new shapes. These measures will contribute to improving the generalization performance of our acquisition system.

CRediT authorship contribution statement

Youssef Amin: Conceptualization, Methodology, Software, Investigation, Validation, Data curation, Writing – original draft, Visualization. **Christian Gianoglio:** Software, Validation, Investigation, Writing – review & editing. **Maurizio Valle:** Resources, Writing – review & editing, Supervision, Funding acquisition.

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

The dataset and software code are publicly available at: https://github.com/YoussifAmin/Object_Hardness_Classification.

Acknowledgments

The authors acknowledge partial financial support from:

- TACTIle feedback enriched virtual interaction through virtual realITY and beyond (TACTILITY) project: EU H2020, Italy, Topic ICT-25-2018-2020, RIA, Proposal ID 856718.
- AI-Powered Manipulation System for Advanced Robotic Service, Manufacturing and Prosthetics (IntelliMan) project: EU H2022, Italy, Grant agreement ID 101070136.

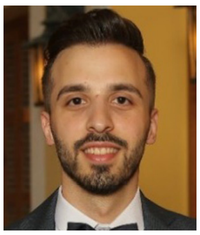
Data availability

The dataset and software code are publicly available at: https://github.com/YoussifAmin/Object_Hardness_Classification

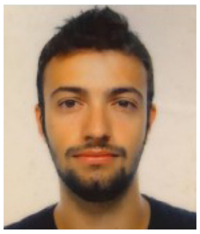
References

[1] B.A. Jenkins, E.A. Lumpkin, Developing a sense of touch, Development 144 (22) (2017) 4078–4090.
[2] M.A. Srinivasan, R.H. LaMotte, Tactual discrimination of softness, J. Neurophysiol. 73 (1) (1995) 88–101.
[3] Y. Wu, Y. Liu, Y. Zhou, Q. Man, C. Hu, W. Asghar, F. Li, Z. Yu, J. Shang, G. Liu, M. Liao, R.-W. Li, A skin-inspired tactile sensor for smart prosthetics, Science Robotics 3 (22) (2018) eaat0429.
[4] Y. Abbass, M. Saleh, S. Dosen, M. Valle, Embedded electrotactile feedback system for hand prostheses using matrix electrode and electronic skin, IEEE Trans. Biomed. Circuits Syst. 15 (5) (2021) 912–925.
[5] R.A. Romeo, C. Lauretti, C. Gentile, E. Guglielmelli, L. Zollo, Method for automatic slippage detection with tactile sensors embedded in prosthetic hands, IEEE Trans. Med. Robot. Bionics 3 (2) (2021) 485–497.
[6] M. Borghetti, E. Sardini, M. Serpelloni, Sensorized glove for measuring hand finger flexion for rehabilitation purposes, IEEE Trans. Instrum. Meas. 62 (12) (2013) 3308–3314.
[7] B.-S. Lin, I.-J. Lee, J.-L. Chen, Novel assembled sensorized glove platform for comprehensive hand function assessment by using inertial sensors and force sensing resistors, IEEE Sens. J. 20 (6) (2020) 3379–3389.
[8] J.M. Butt, H. Wang, R. Pathan, Design, fabrication, and analysis of a sensorized soft robotic gripper, in: 2018 IEEE 8th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems, CYBER, 2018, pp. 169–174.
[9] Y. Wang, X. Wu, D. Mei, L. Zhu, J. Chen, Flexible tactile sensor array for distributed tactile sensing and slip detection in robotic hand grasping, Sensors Actuators A 297 (2019) 111512.
[10] S. Kim, H. Shin, K. Song, Y. Cha, Flexible piezoelectric sensor array for touch sensing of robot hand, in: 2019 16th International Conference on Ubiquitous Robots, UR, 2019, pp. 21–25.
[11] A. Schmitz, Y. Bansho, K. Noda, H. Iwata, T. Ogata, S. Sugano, Tactile object recognition using deep learning and dropout, in: 2014 IEEE-RAS International Conference on Humanoid Robots, IEEE, 2014, pp. 1044–1050.
[12] X. Qian, E. Li, J. Zhang, S.-N. Zhao, Q.-E. Wu, H. Zhang, W. Wang, Y. Wu, Hardness recognition of robotic forearm based on semi-supervised generative adversarial networks, Front. Neurorobot. 13 (2019) 73.
[13] Z. Zhang, J. Zhou, Z. Yan, K. Wang, J. Mao, Z. Jiang, Hardness recognition of fruits and vegetables based on tactile array information of manipulator, Comput. Electron. Agric. 181 (2021) 105959.
[14] Y. Amin, C. Gianoglio, M. Valle, A novel tactile sensing system for robotic tactile perception of object properties, in: AISEM Annual Conference on Sensors and Microsystems, Springer, 2023, pp. 182–187.
[15] Y. Amin, C. Gianoglio, M. Valle, Computationally light algorithms for tactile sensing signals elaboration and classification, in: 2021 28th IEEE International Conference on Electronics, Circuits, and Systems, ICECS, IEEE, 2021, pp. 1–6.
[16] X. Huang, R. Muthusamy, E. Hassan, Z. Niu, L. Seneviratne, D. Gan, Y. Zweiri, Neuromorphic vision based contact-level classification in robotic grasping applications, Sensors 20 (17) (2020).
[17] W. Yuan, C. Zhu, A. Owens, M.A. Srinivasan, E.H. Adelson, Shape-independent hardness estimation using deep learning and a gelsight tactile sensor, in: 2017 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2017, pp. 951–958.

[18] S. Chun, J.-S. Kim, Y. Yoo, Y. Choi, S.J. Jung, D. Jang, G. Lee, K.-I. Song, K.S. Nam, I. Youn, et al., An artificial neural tactile sensing system, Nat. Electron. 4 (6) (2021) 429–438.
[19] O. Kursun, A. Patooghy, An embedded system for collection and real-time classification of a tactile dataset, IEEE Access 8 (2020) 97462–97473.
[20] A. Drimus, M.B.r. Petersen, A. Bilberg, Object texture recognition by dynamic tactile sensing using active exploration, in: 2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication, IEEE, 2012, pp. 277–283.
[21] Z. Su, K. Hausman, Y. Chebotar, A. Molchanov, G.E. Loeb, G.S. Sukhatme, S. Schaal, Force estimation and slip detection/classification for grip control using a biomimetic tactile sensor, in: 2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids), IEEE, 2015, pp. 297–303.
[22] A. Shrestha, A. Mahmood, Review of deep learning algorithms and architectures, IEEE Access 7 (2019) 53040–53065.
[23] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
[24] L. Zou, C. Ge, Z.J. Wang, E. Cretu, X. Li, Novel tactile sensor technology and smart tactile sensing systems: A review, Sensors 17 (11) (2017).
[25] S. Luo, J. Bimbo, R. Dahiya, H. Liu, Robotic tactile perception of object properties: A review, Mechatronics 48 (2017) 54–67.
[26] Y. Amin, C. Gianoglio, M. Valle, Towards a trade-off between accuracy and computational cost for embedded systems: A tactile sensing system for object classification, in: International Conference on System-Integrated Intelligence, Springer, 2023, pp. 148–159.
[27] Google cloud, 2022, https://cloud.google.com/tpu, (Online; Accessed: 20-May-2022).
[28] R.S. Johansson, J.R. Flanagan, Coding and use of tactile signals from the fingertips in object manipulation tasks, Nat. Rev. Neurosci. 10 (5) (2009) 345–359.
[29] G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, K. Huang, Toward an intelligent edge: Wireless communication meets machine learning, IEEE Commun. Mag. 58 (1) (2020) 19–25.
[30] M.T. Yazici, S. Basurra, M.M. Gaber, Edge machine learning: Enabling smart internet of things applications, Big Data Cognit. Comput. 2 (3) (2018).
[31] H. Mao, S. Yao, T. Tang, B. Li, J. Yao, Y. Wang, Towards real-time object detection on embedded systems, IEEE Trans. Emerg. Top. Comput. 6 (3) (2016) 417–431.
[32] M. Rasouli, Y. Chen, A. Basu, S.L. Kukreja, N.V. Thakor, An extreme learning machine-based neuromorphic tactile sensing system for texture recognition, IEEE Trans. Biomed. Circuits Syst. 12 (2) (2018) 313–325.
[33] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, W. Shi, Edge computing for autonomous driving: Opportunities and challenges, Proc. IEEE 107 (8) (2019) 1697–1716.
[34] M. Alameh, Y. Abbass, A. Ibrahim, M. Valle, Smart tactile sensing systems based on embedded CNN implementations, Micromachines 11 (1) (2020) 103.
[35] R. Lora-Rivera, J.A. Luna-Cortés, A. de Guzmán-Manzano, P. Ruiz-Barroso, J. Castellanos-Ramos, Ó. Oballe-Peinado, F. Vidal-Verdú, Object stiffness recognition with descriptors given by an FPGA-based tactile sensor, in: 2020 IEEE 29th International Symposium on Industrial Electronics, ISIE, IEEE, 2020, pp. 561–566.
[36] D. Shadrin, A. Menshchikov, A. Somov, G. Bornemann, J. Hauslage, M. Fedorov, Enabling precision agriculture through embedded sensing with artificial intelligence, IEEE Trans. Instrum. Meas. 69 (7) (2020) 4103–4113.
[37] E. Ragusa, C. Gianoglio, S. Dosen, P. Gastaldo, Hardware-aware affordance detection for application in portable embedded systems, IEEE Access 9 (2021) 123178–123193.
[38] E. Ragusa, C. Gianoglio, R. Zunino, P. Gastaldo, Random-based networks with dropout for embedded systems, Neural Comput. Appl. 33 (2021) 6511–6526.
[39] E. Ragusa, T. Apicella, C. Gianoglio, R. Zunino, P. Gastaldo, Design and deployment of an image polarity detector with visual attention, Cogn. Comput. 14 (1) (2022) 261–273.
[40] X. Wang, F. Geiger, V. Niculescu, M. Magno, L. Benini, Leveraging tactile sensors for low latency embedded smart hands for prosthetic and robotic applications, IEEE Trans. Instrum. Meas. 71 (2022) 1–14.
[41] L. Dutta, S. Bharali, Tinyml meets iot: A comprehensive survey, Internet of Things 16 (2021) 100461.
[42] C. Gianoglio, E. Ragusa, P. Gastaldo, M. Valle, A novel learning strategy for the trade-off between accuracy and computational cost: A touch modalities classification case study, IEEE Sens. J. 22 (1) (2021) 659–670.
[43] H. Liu, J. Greco, X. Song, J. Bimbo, L. Seneviratne, K. Althoefer, Tactile image based contact shape recognition using neural network, in: 2012 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, MFI, IEEE, 2012, pp. 138–143.
[44] Y. Tao, J. Zhou, Y. Meng, N. Zhang, X. Yang, Design and experiment of tactile sensors for testing surface roughness of fruits and vegetable, Trans. CSAM 46 (11) (2015) 16–21.
[45] S. Funabashi, T. Isobe, S. Ogasa, T. Ogata, A. Schmitz, T.P. Tomo, S. Sugano, Stable in-grasp manipulation with a low-cost robot hand by using 3-axis tactile sensors with a CNN, in: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2020, pp. 9166–9173.
[46] S. Funabashi, A. Schmitz, T. Sato, S. Somlor, S. Sugano, Versatile in-hand manipulation of objects with different sizes and shapes using neural networks, in: 2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids), 2018, pp. 1–9.
[47] F. Sakr, H. Younes, J. Doyle, F. Bellotti, A. De Gloria, R. Berta, A tiny CNN for embedded electronic skin systems, in: International Conference on System-Integrated Intelligence, Springer, 2023, pp. 564–573.
[48] I. Bogrekci, P. Demircioglu, H.S. Sucuoglu, O. TURHANLAR, The effect of the infill type and density on hardness of 3D printed parts, Int. J. 3d Print. Technol. Digit. Ind. 3 (3) (2019) 212–219.
[49] I. Bandyopadhyaya, D. Babu, A. Kumar, J. Roychowdhury, Tactile sensing based softness classification using machine learning, in: 2014 IEEE International Advance Computing Conference, IACC, 2014, pp. 1231–1236.
[50] A. Drimus, G. Kootstra, A. Bilberg, D. Kragic, Design of a flexible tactile sensor for classification of rigid and deformable objects, Robot. Auton. Syst. 62 (1) (2014) 3–15.
[51] A. Drimus, G. Kootstra, A. Bilberg, D. Kragic, Classification of rigid and deformable objects using a novel tactile sensor, in: 2011 15th International Conference on Advanced Robotics, ICAR, IEEE, 2011, pp. 427–434.
[52] E. Ragusa, P. Gastaldo, R. Zunino, E. Cambria, Balancing computational complexity and generalization ability: a novel design for ELM, Neurocomputing 401 (2020) 405–417.
[53] Q. Ma, M. Rejab, A.P. Kumar, H. Fu, N.M. Kumar, J. Tang, Effect of infill pattern, density and material type of 3D printed cubic structure under quasi-static loading, Proc. Inst. Mech. Eng. C 235 (19) (2021) 4254–4272.
[54] S.M. Walley, Historical origins of indentation hardness testing, Mater. Sci. Technol. 28 (9–10) (2012) 1028–1044.
[55] S. Azhari, T. Setoguchi, I. Sasaki, A. Nakagawa, K. Ikeda, A. Azhari, I.H. Hasan, M.N. Hamidon, N. Fukunaga, T. Shibata, et al., Toward automated tomato harvesting system: Integration of haptic based piezoresistive nanocomposite and machine learning, IEEE Sens. J. 21 (24) (2021) 27810–27817.
[56] C. Gianoglio, E. Ragusa, R. Zunino, M. Valle, 1-d convolutional neural networks for touch modalities classification, in: 2021 28th IEEE International Conference on Electronics, Circuits, and Systems, ICECS, IEEE, 2021, pp. 1–6.
[57] J.M. Gandarias, A.J. Garcia-Cerezo, J.M. Gomez-de Gabriel, CNN-based methods for object recognition with high-resolution tactile sensors, IEEE Sens. J. 19 (16) (2019) 6872–6882.
[58] C. Gianoglio, E. Ragusa, P. Gastaldo, M. Valle, Trade-off between accuracy and computational cost with neural architecture search: A novel strategy for tactile sensing design, IEEE Sens. Lett. 7 (5) (2023) 1–4.
[59] F. Sakr, F. Bellotti, R. Berta, A. De Gloria, Machine learning on mainstream microcontrollers, Sensors 20 (9) (2020) 2638.
[60] H. Al Haj Ali, C. Gianoglio, A. Ibrahim, M. Valle, Resource-constrained implementation of deep learning algorithms for dynamic touch modality classification, in: International Conference on System-Integrated Intelligence, Springer, 2023, pp. 105–115.
[61] STMicroelectronics, STM32-bit Arm Cortex MCUs, 2022, https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html, (Online; Accessed: 20-October-2022).
[62] STMicroelectronics, STM32CubeIDE, 2022, https://www.st.com/en/development-tools/stm32cubeide.html, (Online; Accessed: 20-October-2022).
[63] Arm developer, 2022, https://developer.arm.com/documentation/ddi0489/f/memory-system/l1-caches, (Online; Accessed: 20-October-2022).
[64] STMicroelectronics, L1–cache on STM32H7 series, 2022, https://www.st.com/resource/en/application_note/an4839-level-1-cache-on-stm32f7-series-and-stm32h7-series-stmicroelectronics.pdf, (Online; Accessed: 20-October-2022).
[65] F. Sakr, R. Berta, J. Doyle, H. Younes, A. De Gloria, F. Bellotti, Memory efficient binary convolutional neural networks on microcontrollers, in: 2022 IEEE International Conference on Edge Computing and Communications, EDGE, 2022, pp. 169–177.
[66] X-CUBE-AI, 2022, https://www.st.com/en/embedded-software/x-cube-ai.html, (Online; Accessed: 20-June-2021).
[67] STMicroelectronics, STM32f0 series, 2022, https://www.st.com/en/microcontrollers-microprocessors/stm32f0x0-value-line.html, (Online; Accessed: 20-October-2022).



Youssef Amin received the B.S. degree in Electronics engineering / Biomedical engineering and the M.Sc. degree in electronics engineering / biomedical engineering from the Faculty of engineering, Lebanese International University, Beirut, Lebanon, in 2017 and 2019, respectively. He is currently working toward the Ph.D. degree in science and technology for electronic and telecommunication engineering with the Department of Naval, Electrical, Electronic, and Telecommunications Engineering, University of Genoa, Genoa, Italy. His main research interests include signal processing, machine learning, embedded electronics, integrated sensing systems.



Christian Gianoglio received the master's degree cum laude in electronic engineering and the Ph.D. degree in electrical engineering from the University of Genoa, Italy, in 2015 and 2018, respectively. He is currently a Researcher Fellow at DITEN, University of Genoa. His main research areas include machine learning for resource-constrained devices, machine learning for tactile applications, and pattern recognition for quality assessment of insulation systems in electrical apparatuses.

Maurizio Valle received the M.S. degree in electronic engineering and the Ph.D. degree in electronics and computer science from the University of Genoa, Italy. From December 2019, he was a Full Professor of Electronics with DITEN, University of Genoa, where he leads the Connected Objects, Smart Materials, Integrated Circuits (COSMIC Laboratory). He has been and is in charge of many research contracts and projects funded at local, national and European levels. His research interests include bio-medical circuits and systems, electronic/artificial sensitive skin, tactile sensing systems for prosthetics and robotics, neuromorphic touch sensors, electronic, and microelectronic systems.