Embedded Machine Learning for Machine Condition Monitoring

Michael Grethler[1](✉), Marin B. Marinov[2], and Vesa Klumpp[3]

[1] Institute for Information Management in Engineering (IMI), Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
michael.grethler@kit.edu
[2] Department of Electronics, Technical University of Sofia, 8, Kliment Ohridski Blvd., 1756 Sofia, Bulgaria
[3] Knowtion GmbH, An der RaumFabrik 33c, 76227 Karlsruhe, Germany

Abstract. With the application of a new generation of information technology in the field of manufacturing and the deep integration of computer technology and manufacturing, industrial production is moving towards intellectualization and networking . Because the current production system cannot fully exploit the value of industrial data and the existence of information islands in the production process, this paper presents a study on the development, testing, and evaluation of a machine learning process that can be run on low-cost standard microcontrollers with limited computing and memory resources. This paper first analyzes the basic idea of whether it is possible to develop software for intelligent sensors whose algorithms run on microcontrollers. At the same time, it is considered whether the training and the adaptation of the model parameters can be done on the microcontroller to enable an online adaptation of the machine to be monitored. The goal is a closed system that does not need a backend and the storage of large amounts of data is not necessary.

Keywords: Embedded machine learning · Microcontroller · Intelligent sensor · Industrie40 · IoT · AI

1 Introduction

1.1 Motivation

Modern sensor technologies are at the basis of many applications, such as monitoring the state of structures [1, 2] Internet-of-Things (IoT), diagnostics of machines [3, 4], human body monitoring [5], health [6, 7], localization of sites [8, 9], ecological monitoring [10], electronic circuits analysis [11], etc. The application of sensor technologies is often related to the processing of large volumes of data, which creates problems with storage, transmission, and security [12, 13].

Modern research shows that by the end of 2020, embedded systems provide more than 10% of all digital data [14]. Sharing sensor systems and using cloud and edge computing

D. Perakovic and L. Knapcikova (Eds.): FABULOUS 2021, LNICST 382, pp. 217–228, 2021.
https://doi.org/10.1007/978-3-030-78459-1_16

technologies is an alternative for achieving efficient processing of large volumes of data and high transmission speeds.

This paper discusses the capabilities of embedded sensor systems of limited resources to undertake machine learning tasks.

The design, implementation, and integration of a machine learning algorithm based on a conventional sensor development kit of the Bosch company are also presented [15]. The algorithm that is implemented uses sensor data obtained in real-time. To apply the machine learning algorithms to the sensor board, an appropriate data structuring is made and the calculation sequence is optimized.

The process of machine learning and classification is demonstrated by real-time experiments.

The chosen hardware platform and machine learning software offer many advantages, such as:

a) enabling the sensor system to be continuously trained;
b) providing an opportunity for real-time analysis of the condition of the machine;
c) scalability so that the algorithms also work with additionally mounted sensors;
d) reduction of the data volume from the sensors and improved security through the communication of parameters instead of large volumes of raw data.

1.2 Artificial Intelligence in Embedded Systems

Until now, implementing artificial intelligence (AI) in embedded systems has been a lengthy process requiring the expertise of data scientists, months or years of development time, complex, large datasets, lots of computing power, and often expensive specialized hardware. The study was concerned with the development, testing, and evaluation of a machine learning procedure that can be executed on low-cost standard microcontrollers with limited computing and memory resources. The entire learning process was thus to be executed directly on the microcontroller. The task here was to implement data compression directly on the sensor for automated condition monitoring of machines with low communication bandwidths. The idea here was to develop software for intelligent sensors that output (only) the status of the machine and not the raw data. The respective algorithms for this are to run on a microcontroller. The solution approach here goes further than the calculation of a Fourier transform, as it is often used in vibration analysis, special machine learning methods are used. The training and adaptation of the model parameters are also to be performed at runtime on the microcontroller, thus enabling online adaptation to the particular machine to be monitored.

The learning itself is to be done completely locally on the microcontroller. This provides a closed system that does not require a backend for learning. The learning process is done online, iteratively in a kind of streaming pipeline. A short data vector of a few 100 to 1000 data points is ingested and after pre-processing and feature extraction (consisting of whitening, pooling, principal component analysis, and frequency analysis), a model update is performed based on the features. This is performed iteratively several times, each time with new data, so there is no need to store large amounts of data.

The study was conducted as part of the funding measure “Industrie 4.0 test environments - mobilizing SMEs for Industrie 4.0”. The unique selling point of the funding

measure was the short-term and focused collaboration between SMEs and Industrie 4.0 test environments at research institutions. The project focused on the development, testing, and further development of digitized processes and products as well as networked business models.

The Industrie 4.0 Collaboration Lab of the Institute for Information Management in Engineering (IMI) at the Karlsruhe Institute of Technology was selected as the test environment. The selection was based primarily on the following:

– The availability of machines and qualified personnel to operate them,
– The main focus of the Institute IMI on automation and data management,
– The equipment of the laboratories.

2 Materials and Methods

2.1 Embedded Machine Learning

Machine learning (ML) is a category of artificial intelligence that uses algorithms and statistical methods to efficiently perform a variety of tasks on embedded processors without the need for precise instructions [16, 17].

Deep learning is a common name for a family of ML methods based on neural networks. Examples are Convolutional Neural Networks, Recurrent Neural Networks, Deep Neural Networks (DNN). These methods are currently widely used in areas such as computer vision, speech recognition, and natural language processing [18, 19].

To be able to solve various tasks through machine learning, it is necessary to obtain results from a trained ML model of the embedded system. There are two main approaches for solving this problem: cloud computing and edge/distributed computing. With the cloud approach, all data collected from embedded devices are sent to the cloud over the Internet so that remote servers can execute ML algorithms, and then the results are sent back. With the second approach - edge computing, ML models are pushed to the periphery of the network [20].

Embedded machine learning refers to the second approach, i.e. placement of the ML model in the embedded device and local execution of calculations. The training phase can be done locally or externally. Besides, the main focus of Embedded ML is the use of deep learning training methods. Therefore, for the successful implementation of Embedded ML, it is necessary to choose the optimal configuration of hardware architecture and algorithms to achieve a balance between performance, accuracy, and cost. When designing a built-in solution for ML, a number of questions should be carefully answered, the most important of which are [20]:

- What type of neural network model provides the best trade-off for the accuracy and performance of that particular application?
- What type of processor should be used? (e.g. with a reduced instruction set (RISC) or with a complete instruction set (CISC)?
- How much RAM is needed?

2.2 Machine Learning on Any M-Cortex-Based MCU

"Twenty-five billion objects will be AI-smart by 2025 - that's 25 percent of all edge devices sold by then," Rubino quotes from a recent IDC study. To reach this enormous number, he says, it will be necessary to make the development of integrated AI algorithms much easier, faster, and more affordable than it is today. The problem: Until now, ML, or creating and training neural networks (NNs), has required a lot of computing power. A lot of computing power. That's why these tasks typically run on specialized, power-hungry processors in large data centers, such as in a cloud environment. Outsourcing precisely such tasks to microcontrollers in edge devices, which are trimmed for cost and energy efficiency and therefore have only (very) limited computing power, seems daring at first glance, to say the least. During the study, research on the state of the art was conducted at regular intervals. The product "Bob Assistant" of the French manufacturer Cartesiam became known. It is a solution in the area of anomaly detection using IoT sensors, which are directly attached to devices to be monitored. But Cartesiam promises exactly that with the recently introduced NanoEdge AI Studio integrated development environment (IDE) for AI and ML applications. However, the company approaches the problem from a different angle. Instead of developing and optimizing an NN model from scratch for each new task, embedded developers will be able to select one from a huge library of pre-built models that best fit their application. But that's not all: According to Rubino, the models are capable of adapting themselves to the application better and better.

2.3 Mounting the Sensors

This section describes the work carried out and the results obtained. (Each subchapter corresponds to a work package.) Deviations from the planned implementation are shown. Selection of sensors. Two potential candidates were investigated as IoT sensors: Bosch XDK 110 (XDK) and the Analog Devices Evaluation Board EV-COG-AD4050LZ with the sensor extension board EV-GEAR- MEMS1Z (ADI). The selection of the sensor technology was based on different criteria. ADI was less suitable because it is a development board primarily for hardware manufacturers and does not have its power supply using an accumulator. Furthermore, ADI does not have a gyroscope, which is especially important for the detection of dynamic movements, and protective housing, which would have to be designed and manufactured. ADI has a slightly more powerful microcontroller with a floating-point unit (Cortex M4F, compared to Cortex M3 in XDK), but the main memory configuration is identical for both sensors. XDK has an accumulator as well as these accelerometers and gyroscope and also provides wireless communication technologies such as WLAN and Bluetooth Low Energy (BLE), which allows the wireless installation of the sensors. Furthermore, XDK offers a slot for micro SD cards, which can be used for recording raw data.

For these reasons, XDK was chosen.

Machine selection and sensor installation. The following machines were selected for testing:

- Milling machine consisting of CNC portal machine EAS(Y) 300 and spindle set Jäger 42-2 W38 FS.

- AUBO i5 robot arm.

From preliminary work by Knowtion and IMI, a mounting location for the XDK on the milling machine had already been selected and evaluated from several possibilities. Furthermore, a mounting point for an XDK was available here on the underside of the spindle. This allows the sensor to be mounted as close as possible to the location of the chip removal, see Fig. 1. It has been found that fine chips are deposited on the housing of the sensor. However, these did not influence the functionality of the XDK.

Fig. 1. Bosch XDK on the milling machine. Deposits on the sensor caused by the machining process can be seen here.

Based on the 6 axes of the robot arm, investigations were carried out into the possible mountability of the XDK. After an initial attachment of the XDK to the flange, the configuration was changed. To increase the observability of the joint and to use the flange for tools, two brackets were produced which allow XDKs to be attached to the rigid ends of joint 6 and joint 3. The brackets are shown in Fig. 2. All joints of the robot arm are shown in Fig. 3.

2.4 Recording of Sensor Data

This was due to the changed configuration of the attachment of the XDKs to the robot arm and the subsequent recording of data for the milling machine.

First, the existing software was adapted for XDK to store raw data on an SD card. The raw data was displayed in CSV format.

When recording the data for the robot arm, acceleration data, and rotation rate data of several different trajectories were recorded. Three trajectories were defined (Trajectory A, Trajectory B, and Trajectory C), which differed slightly in individual motion sections. Trajectory A was defined as “normal case”, trajectories B and C were created as copies

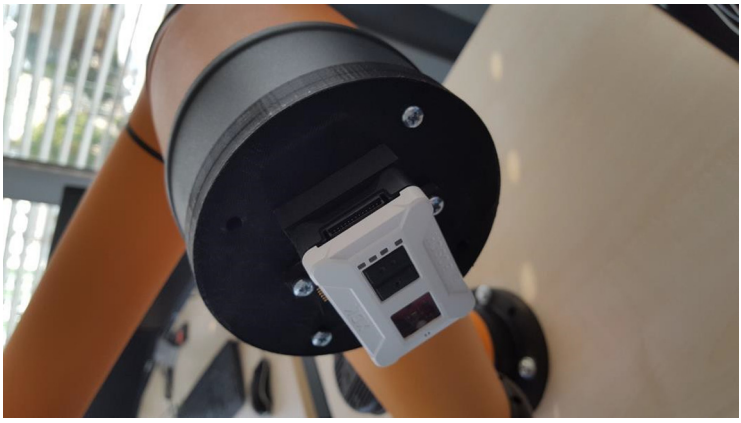

Fig. 2. Bosch XDK sensor on joint 3 of the AUBO i5.

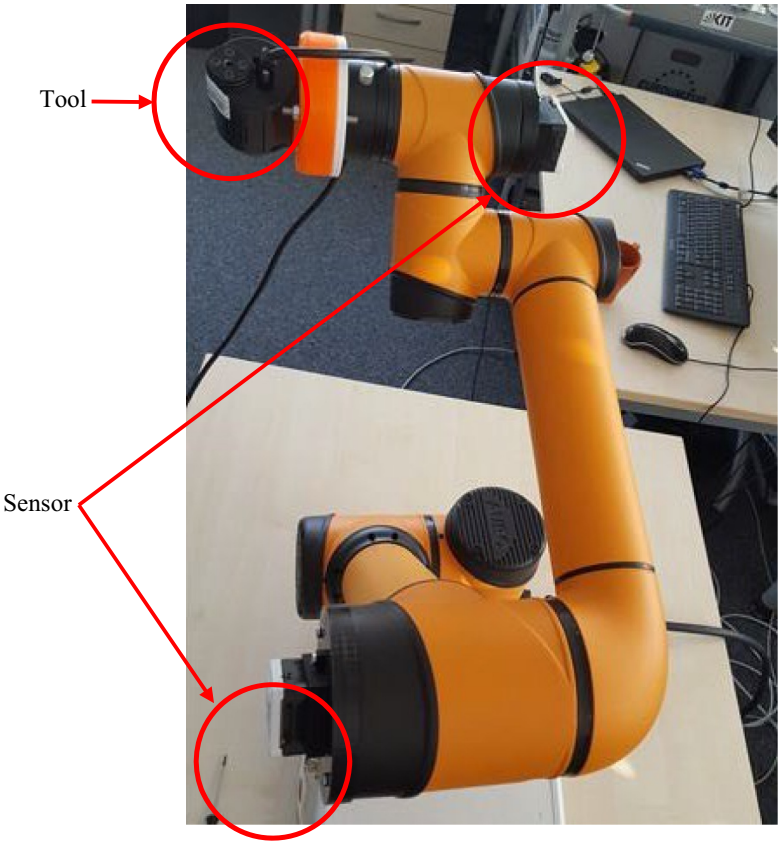

Fig. 3. Installation of Bosch XDK sensors on the AUBO i5. XDKs are mounted on axes 3 and 6.

and slightly modified (either single points of the trajectory or the speed of movement) to be used as an anomaly. The recording of the data to an SD card was done in three steps with several repetitions of each trajectory.

For the data recording of the milling machine, a CNC program was first designed, which performs a simple trajectory as a milling movement. For comparability, the same program was always used. For the selection of the tools different tools were used in each case:

- 6 mm ball cutter
- 14 mm end mill
- 6 mm end mill new
- 6 mm end mill used (slight defect on the cutting edge).

The spindle was operated at speeds of 15,000 rpm and 35,000 rpm. Data sets with several repetitions of the tool and speed combinations were recorded.

2.5 Algorithm and Software Development

The development of the algorithms was done in Python to solve the mathematical and algorithmic problem sufficiently well and to extend the existing algorithms. Algorithmic approaches were developed and evaluated based on the acquired measurement data in a rapid prototyping environment.

The procedure was iterative, first, the algorithms were revised or developed and then the performance of the algorithms is checked with the measured data.

Contrary to the plan, it turned out relatively early on that differentiating between different machine states is less helpful. Thus, the approach was taken to have only one "normal state" and mark deviations from it as an anomaly.

The processing chain was extended to be able to process the acceleration and gyroscope data with as little memory consumption as possible. Here, an approach was chosen that uses as little memory as possible and does not require multiple data.

As a result, several processing stages with the following functionalities have been developed:

In preprocessing, multidimensional sensor data are processed to a feature vector. This vector still contains the same necessary information but is more memory efficient. First, the sensor data are normalized and scaled appropriately. Furthermore, eigenvectors are calculated and frequency analysis is performed.

In training, the sensor data are converted into feature vectors by preprocessing. Subsequently, a recursive calculation of further quantities is performed to unify several feature vectors by smoothing. The result is processed into a clustering algorithm and leads to a model update. Here, adjustments of the algorithms could lead to a significantly faster convergence rate. This is necessary to be able to perform a model adjustment as easy and time-saving as possible in case of misclassification. For this purpose, approaches were developed and investigated that allow an iterative update of the model.

Anomaly detection was implemented by deriving feature vectors from the raw data and comparing them to the clusters. A challenge here was the delimitation of the affiliation of a data point to a cluster. Here, different approaches to the distance measure were developed and evaluated.

In contrast to the planned procedure, the processing effort was not reduced. The focus of the optimizations was to achieve the highest possible accuracy in classification and the most efficient use of memory.

To determine the quality of results and performance, an evaluation of each development iteration was carried out. For this purpose, a Python script was developed, which applies the procedures to the existing data sets and allows a systematic comparison of the results. The algorithms were trained with a part of the measured data and the result was checked against the non-training data.

Implementation. Following the algorithm development, the implementation was done in C. Several requirements were of special interest:

The portability of the developed source code should be kept as high as possible. This was achieved by using standard C language constructs without external libraries. This should make the source code as compatible as possible with different development platforms and it should be easy to make the algorithm executable on other microcontrollers and their build environments.

The optimization of the memory consumption was of great interest, because with lower memory consumption per unit of information more information can be stored in the main memory of the microcontroller and thus the trained model can capture more complex relationships. For this purpose, a representation of real numbers reduced to 16 bits was used, which still achieves useful results for the application despite the loss of accuracy. Thus feature vectors and cluster parameters could be represented more memory efficient.

To be as close as possible to the Python source code from the process step of algorithm development, comparisons of the results of both implementations were carried out. This was done as an automated test to easily perform regression tests during implementation and optimization.

It was not necessary to optimize the program code concerning computing resources, since all program parts could be processed within fractions of a second.

The result of the implementation was software running on the XDK, which processes acceleration and gyroscope data and can train and process a model for anomaly detection with low memory requirements.

2.6 Integration in a Test Environment

The software for XDK has been extended so that the calculated data can be made available via Bluetooth Low Energy (BLE). BLE has the advantage that no additional infrastructure is required and does not need to be integrated into a local network. This allows a higher flexibility in use. The data was provided via a Generic Attribute Profile (GATT) as a separate Bluetooth profile, which includes the anomaly indicator and a measure of model complexity.

As a user interface, an Android app was developed. This app communicates via BLE with the XDKs and allows the user to view the status of the learned models. Furthermore, the app allows the user to continue the training of the model. This updates the model in case of misclassification.

The app essentially consists of an activity, which graphically displays the anomaly indicator and the model complexity as time series.

3 Experimental Results

The operation and evaluation of the system were carried out under testbed conditions and feedback on the system was obtained.

3.1 Test Operation

The test operation showed the following essential results:

- During operation, users had difficulties mapping the anonymous machine states reported by the software to the actual process in the machine. Therefore, the consideration was limited to anomaly detection only without considering individual machines or operating states.
- There had been program errors that caused invalid results due to an incorrect numerical calculation. As a result, either no results or wrong results were displayed.
- Outliers in the measured data and not considered disturbances, such as impacts, had a significant influence on the calculated result. However, these short-term disturbances did not affect the machine.
- A faulty classification had to be reported immediately by the user as long as the faulty classification was still displayed and the behavior of the machine and sensor data did not change. Here the wish was to be able to consider history to be able to make corrections on the model afterward.
- The transferability of the model from one XDK to another XDK was also a listed wish. This was especially relevant when replacing a sensor.
- Depending on the application scenario, the developed procedure requires an independent orientation or alignment of the XDK during installation. This occurred when the XDK was reinstalled in reverse orientation.
- It was possible to train a model relatively well with one run. However, two to three runs are advantageous in training to take into account the dispersion of data.
- In addition, raw data were recorded to investigate certain misclassifications, e.g. the correct differentiation of tools in the milling machine. An evaluation is shown in Fig. 4.

3.2 Adjustments

The adjustments were mainly related to the algorithmic part of the system to increase the accuracy and robustness of the results. Here, programming errors in the prototypical

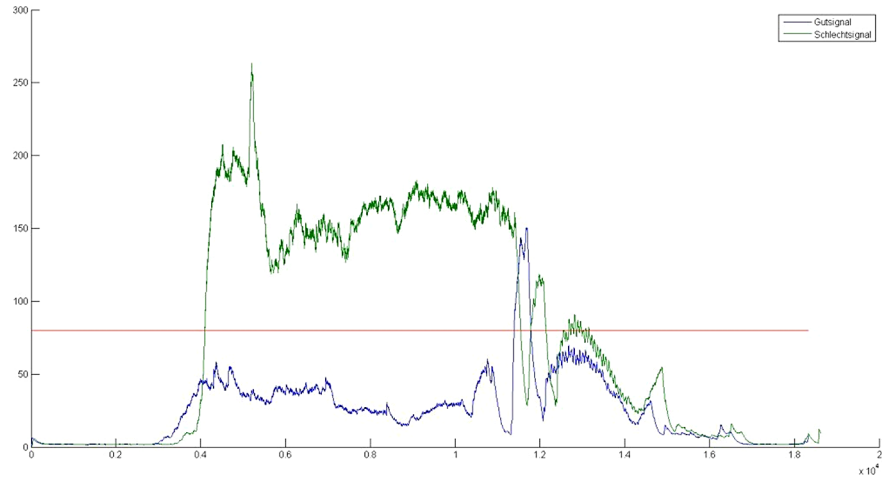

Fig. 4. Differentiation between good signal (blue; no error) and bad signal (green; error present) (Color figure online).

implementation were fixed and an adaptation of the quantification of internal quantities in a 16-bit format was performed. Furthermore, parameters and the measure of quality regarding cluster membership were adjusted. Here, the sensitivity or the question of whether a measurement series can be assigned to a state or not was investigated further. The result of the adaptation of the algorithms was that the selectivity could be improved with an increasing number of existing training data.

4 Conclusions and Future Work

Embedded machine learning involves several limitations and the need to compromise. There is no common best practice and the best way to successfully implement it is to make optimal use of available resources. The present work presents some challenges in the implementation of machine learning on embedded devices.

In general, most of the challenges associated with the use of machine learning in embedded devices can be overcome through appropriate software and hardware optimizations. That is why this approach is increasingly used.

Today, there are a variety of software solutions for machine learning, but these are usually only available for the PC and use Python as the programming language [21]. A solution that allows neural networks to be executed and trained on embedded systems, such as microcontrollers, is not currently available. However, it may be helpful to perform training directly on the embedded system - for example, if an implanted sensor is to calibrate itself. The vision is for sensor-related AI that can be integrated directly into a sensor system. The artificial neural network also for deep learning.

With “Artificial Intelligence for Embedded Systems”, the development team has realized this idea and developed a machine learning library in the C programming language that can be run on micro-controllers, but other platforms such as PC, Raspberry PI, or Android are also considered. The technology library currently includes a fully

configurable artificial neural network (KNN) that can generate deep networks for deep learning if the interest is there. The use case used here was optimized specifically for embedded systems. The use case was optimized specifically for embedded systems. “We kept the source code to a minimum so that the KNN can be learned directly on the microcontroller or sensor, i.e. the embedded system. In addition, the source code is universal; it can be compiled for almost any platform. Since the same algorithms are always used, a KNN created on a PC, for example, is easily portable to a microcontroller . This was previously not possible with commercially available software solutions.

Acknowledgments. This research is partly supported by the Bulgarian National Science Fund in the scope of the project “Exploration the application of statistics and machine learning in electronics” under contract number КП-06-Н42/1.

References

1. Lynch, J.P.: A summary review of wireless sensors and sensor networks for structural health monitoring. Shock Vibr. Digest **38**(2), 91–128 (2006)
2. Yick, J., Mukherjee, B., Ghosal, D.: Wireless sensor network survey. Comput. Netw. **52**(12), 2292–2330 (2008)
3. Jardine, A.K.S., Lin, D., Banjevic, D.: A review on machinery diagnostics and prognostics implementing condition-based maintenance. Mech. Syst. Sig. Process. **20**(7), 1483–1510 (2006)
4. Aldrich, C., Auret, L.: Unsupervised Process Monitoring and Fault Diagnosis with Machine Learning Methods. In: Advances in Computer Vision and Pattern Recognition. Springer, Berlin (2013). https://doi.org/10.1007/978-1-4471-5185-2
5. Liang, M., Zhuang, Y., Cai, C.: Face classification using sparse reconstruction embedding. In: Proceedings of International Conference on Computational Intelligence and Software Engineering (CiSE), Wuhan, China, December 2010
6. Patel, S., et al.: A review of wearable sensors and systems in rehabilitation. J. Neuroeng. Rehab. 1–17 (2012)
7. Rajan, D., Spanias, A., Ranganath, S., Banavar, M., Spanias, P.: Health monitoring laboratories by interfacing physiological sensors to mobile android devices. In: Proceedings of IEEE FIE (2013)
8. Zhang, X., et al.: Performance comparison of localization techniques for sequential WSN discovery. In: Proceeding of IEEE SSPD, September 2012
9. Zhang, X., Tepedelenlioglu, C., Banavar, M., Spanias, A.: Distributed location detection in wireless sensor networks. In: Proceedings of IEEE Asilomar Conference on Signals, Systems and Computers, November 2013
10. Hino, M., Benami, E., Brooks, N.: Machine learning for environmental monitoring. Nat. Sustain. **1**, 583–588 (2018). https://doi.org/10.1038/s41893-018-0142-9
11. Ivanova, M.: Methodology for analysis of gm-c filters based on statistical, fuzzy logic, and machine learning approach. In: EEET 2020: Proceedings of the 2020 3rd International Conference on Electronics and Electrical Engineering Technology, September 2020
12. Hu, H., Wen, Y., Chua, T.-S., Li, X.: Toward scalable systems for big data analytics: a technology tutorial. IEEE Access, 652–687 (2014)
13. Xia, W., Mita, Y., Shibata, T.: A nearest neighbor classifier employing critical boundary vectors for efficient on-chip template reduction. IEEE Trans. Neural Netw. Learn. Syst. **27**(5), 1094–1107 (2016)

14. A'râbi, M.-A., Schwarz, V.: General constraints in embedded machine learning. In: SCA, Freiburg, July 2019
15. Cross-Domain Development Kit XDK110 Platform for Application Development. Bosch Connected Devices and GmbH (2016)
16. Bishop, C.M.: Pattern Recognition and Machine Learning. Information Science and Statistics, Springer, Heidelberg (2006)
17. Andrade, L., Prost-Boucle, A., Pétrot, F.: Overview of the state of the art in embedded machine learning. In: 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden (2018)
18. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**, 436–444 (2015)
19. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS 2012) (2012)
20. Arm, E.M.: Learning Design for Dummies. Wiley, Hoboken (2019)
21. Lowhead, J.: Learning GeoSpatial Analysis with Python, 2nd edn. Packt, Birmingham (2015)
22. Zekveld, M., Hancke, G.P.: Vibration condition monitoring using machine learning. In: IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society, Washington, DC, pp. 4742–4747 (2018). https://doi.org/10.1109/IECON.2018.8591167
23. Paolanti, M., Romeo, L., Felicetti, A., Mancini, A., Frontoni, E., Loncarski, J.: Machine learning approach for predictive maintenance in industry 4.0. In: 2018 14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA), Oulu, pp. 1–6 (2018). https://doi.org/10.1109/MESA.2018.8449150
24. Alsheikh, M.A., Lin, S., Niyato, D., Tan, H.: Machine learning in wireless sensor networks: algorithms, strategies, and applications. IEEE Commun. Surv. Tutor. **16**(4), 1996–2018 (2014). Fourth quarter. https://doi.org/10.1109/COMST.2014.2320099
25. Wilkinson, M., Darnell, B., Delft, T.V., Harman, K.: Comparison of methods for wind turbine condition monitoring with SCADA data. IET Renew. Pow. Gener. **8**(4), 390–397 (2014). https://doi.org/10.1049/iet-rpg.2013.0318
26. Hodge, V.J., O'Keefe, S., Weeks, M., Moulds, A.: Wireless sensor networks for condition monitoring in the railway industry: a survey. IEEE Trans. Intell. Transp. Syst. **16**(3), 1088–1106 (2015). https://doi.org/10.1109/TITS.2014.2366512