PRZEGLĄD ELEKTROTECHNICZNY

OPEN ACCESS

Bartosz DOMINIKOWSKI
ORCID: 0000-0002-4762-2005
DOI: 10.15199/48.2025.11.32

A TinyML-powered classifier designed for 2D data analysis in device diagnostics

Klasyfikator oparty na TinyML służący do analizy danych 2D w diagnostyce urządzeń

dr inż. Bartosz Dominikowski, Politechnika Łódzka, Instytut Systemów Inżynierii Elektrycznej, ul. Stefanowskiego 18, 90-537 Łódź
Correspondence address: bartosz.dominikowski@p.lodz.pl

Abstract: Rozwój elektroniki umożliwia realizację złożonych zadań dzięki rosnącej mocy obliczeniowej układów cyfrowych. W artykule opisano wdrożenie algorytmu uczenia maszynowego w układzie mikroprocesorowym. Proponowany system można zastosować w diagnostyce pracy urządzenia. Inteligentny algorytm przypisuje etykiety sektorom powierzchni roboczej urządzenia w zależności od jego stanu pracy. Autor pracy przedstawił cały proces projektowania tego algorytmu.

Keywords: machine learning, artificial intelligence, TinyML, neural network

Streszczenie: The development of electronics enables the execution of complex tasks thanks to the increasing computational power of digital systems. The article describes the implementation of a machine learning algorithm in a microprocessor-based system. The proposed system can be applied to device operation diagnostics. The intelligent algorithm assigns labels to sectors of the device's working surface depending on its operating state. The author presents the entire design process of this algorithm.

Słowa kluczowe: uczenie maszynowe, sztuczna inteligencja, TinyML, sieć neuronowa

Introduction

Measurement systems are increasingly supported by artificial intelligence-based algorithms, resulting in advanced diagnostic or control and measurement systems. In automation in industrial applications, e.g. the Industrial Internet of Things (*IIot*), monitoring the operating status of individual device elements (Condition Based Monitoring, *CbM*) allows for insight into the operating status of the entire system. Such support affects: increasing the efficiency of the device, minimizing downtime, reducing repair costs and ensuring safety against the occurrence of a major failure. A system like this works well in industry 4.0, which drives the development of intelligent factories supported by digital technology. Owing to their capability to process and analyze large datasets, artificial intelligence (*AI*) and machine learning (*ML*) algorithms make it possible to perform diagnostic and other application-specific tasks. The use of *ML*-based algorithms improves the working of diagnostic system by indicating the best directions of action. Machine learning *ML* is an area of artificial intelligence concerning algorithms that can modify their parameters based on data and perform tasks without explicit instructions. *TinyML* technology is a field that deals with running machine learning models on devices with limited computational resources, such as microcontrollers. Dedicated libraries are often used in such systems, e.g. TensorFlow Lite for Microcontrollers (*TFLM*, a port for TensorFlow), which was designed to work without an operating system, with minimal memory and energy consumption (a lightweight platform for *ML*). Properly programmed classic *AI* algorithms (e.g. fuzzy logic, decision trees) work correctly on microcontrollers. However, a model based on machine learning must be optimized to work properly in an embedded system. The author of the article describes the entire process of implementing an algorithm based on *ML* machine learning and its implementation in a microcontroller system.

The main focus of this article is to present the structure and implementation of an intelligent algorithm based on machine learning *ML* in a microprocessor system used for diagnostic the correctness of the device operation. This algorithm divides the workspace of the supervised system into given sectors in a quantity dependent on the number of labels defined by the user. The usefulness of the proposed algorithm becomes especially apparent when dealing with a large number of sectors of varying shapes, which are difficult to reconstruct using traditional *if/elseif/else* conditional statements.

Literature review

Currently, the machine learning is widely published and described in scientific papers. The implementation of machine learning algorithms in embedded systems is thoroughly presented in [1]. This book describes in detail the methods of implementing algorithms based on *TinyML* technology in a microcontroller system. The paper [2] presents an overview of the current state of development of *TinyML* along with a set of tools for its operation. The authors of this paper also present the key factors enabling the improvement of *TinyML*-based systems. The papers [1, 2] are valuable contributions to the development of the proposed algorithm. The paper [3] presents an overview of the use of artificial neural networks for short-term power forecasting and energy production in photovoltaic systems. This article confirms the effectiveness of using algorithms based on neural networks, which is important in the proposed algorithm. Reference [4] presents research and development focused on the practical applications *TinyML*. This paper helped in the implementation of the algorithm by paying attention to the available computational power and limited memory. The paper [5] presents a set of *TinyML* algorithms developed to improve the management of Big Data in large-scale *IoT* systems. The paper [6] presents a detailed review of available *TinyML* frameworks for integrating *ML* algorithms into *MCUs*. The review of frameworks allowed the author of this paper to select the appropriate one. In [7] the authors present the benefits and applications of *TinyML* and provide general information based on the current literature. Next, the authors of this paper present the TensorFlow Lite framework, which supports *TinyML*, as well as the analytical steps aimed at creating a machine learning model. The descriptions of platforms, frameworks and libraries that are compatible with *TinyML* technology were used to build the proposed algorithm. In [8] the *TinyML* technique used on microcontrollers is presented, focusing on optimization. This study stands out due to the quantization technique used as an optimization method in the proposed algorithm. In [9] the authors used a systematic literature review based on the PRISMA methodology and analyzed three approaches to implement *TinyML*: *ML*-oriented, *HW*-oriented and co-design. The mentioned papers significantly helped the author in conducting the research due to the valuable information provided regarding microcontrollers and *TiniML* implementation. Machine-learning (*ML*) techniques can be used to optimize the numerical algorithms described in [10, 11]. These references provide demanding and detailed methodologies together with practical guidance for tackling complex computational problems.

Operating point of the diagnosed system

The proposed system in diagnostics application divides the surface of consideration into five parts based on the labels provided by the user. The number of sectors may be larger and depends on the degree of complexity of the tested device or controlled system. The intelligent algorithm can be used in monitoring the operation of an electric motor. The division of the surface of considerations for the above example is given below:

Example 1: Division into sectors of the surface of consideration

1. Left-bottom (*LB*) – supply voltage too low;
2. Right-bottom (*RB*) – Engine without load, e.g. idling;
3. Left-Top (*LT*) – Short circuit, in the power supply circuit;
4. Right Top (*RT*) – Full load operation, maximum performance;
5. Center (*C*) – Nominal supply and load. (*X*-axis – supply voltage (V_{in1}), *Y*-axis – load current (V_{in2})).

An example of a normalized surface is shown in Figure 5.

Structure of the measurement system

The system processing the input information concerning the diagnostics of the tested system is a *PSoC 5LP* microprocessor. It has the ability to design an electronic circuit (analog and digital) composed of ready-made integrated components in a programmatic manner (hardware design). This measuring circuit consists of a analog multiplexer, operational amplifier, *ADC* and *DAC* converter. Figure 1 shows the circuit processing the input signals from the tested device (V_{in1}, V_{in2}).

The analog multiplexer allows the input signals (V_{in1}, V_{in2}) to be connected to the input of one *ADC* sigma-delta converter ($\Sigma\Delta$) in the measurement system. This system introduces additional time needed

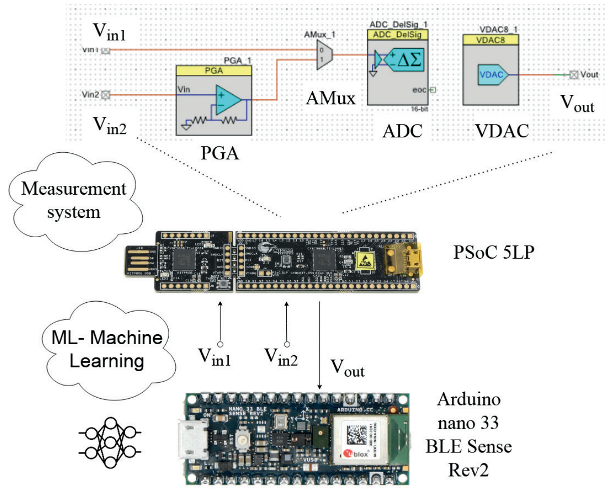

Fig. 1. Processing of the input signal in the *PSoC* microprocessor in the measurement system

to switch signals from measurement channels. In systems requiring fast response, it can cause significant measurement errors, especially when many measurement signals are used. The *PGA* (Programmable Gain Amplifier) amplifier used in the measurement system allows for programmable amplification of the input signal or its transmission with a constant gain of 1 V/V. The V_{in1} and V_{in2} signals are processed in the *ADC–DAC* circuit. At the output of the digital-to-analog converter (*DAC*), voltage signals $V_{out1}(t_1)$ and $V_{out1}(t_2)$ $(t_2>t_1)$ $(V_{out}=[V_{out1}(t_1)\ V_{out1}(t_2)])$ are generated, which are transferred on the output terminal of the measurement system. The measurement system transmits these signals to the input of a second microcontroller, where an intelligent algorithm is implemented. In the measuring system there was no need to amplify the signal, therefore the relationship $V_{in1}=V_{out1}(t_1)$ and $V_{in2}=V_{out1}(t_2)$ occurs in the processing circuit.

Construction of a neural network in a diagnostic system

The author's model implemented in the microcontroller system defines different work areas in the operating space of the tested device. The mapped work space of the device is divided into sectors that define its state by using a label containing the name of the area: nominal operation, no supply voltage, etc. The author of the intelligent model uses the TensorFlow library and the integrated high-level Keras framework as a tool for designing, training, evaluating and implementing machine learning models. The intelligent algorithm was implemented in an interpreted language such as Python. Model training was carried out in the Google Colab environment operating in the cloud. In a later phase of the design, the model was converted to TensorFlow Lite (*TFLite*) format and examined with the Netron. This tool is used to visualize neural network models. The structure of the proposed algorithm can be presented in the form of the following diagram:

Algorithm: Embedded-ready neural classifier with decision boundaries in *2D* (model.py)

1. Importing the appropriate libraries;
2. Entering input data (points) and class labels;
3. Building a neural network model ;
4. Compile the model by setting the optimizer, loss function, and metrics;
5. Train the model;
6. Convert the model to TensorFlow Lite (*TFLite*) format;
7. Convert the model to *C* header format (*.h*).

The above process outlines a complete workflow from data preparation and model training to conversion for deployment on embedded systems. This enables creating an efficient neural network classifier optimized for resource-constrained devices, allowing real-time diagnostics and decision-making directly on the device.

Figure 2 shows the sequential diagram of the proposed algorithm.

The diagram from Figure 2 allows to visualize the order and way of exchanging messages between the algorithm objects in time, which makes it easier to understand its logic.

The trained model's structure is illustrated using the Netron tool in Figure 3, with additional details shown in Figure 4.

A useful technique for optimizing the algorithm is the application of quantization. During training of the model based on machine learning, calculations require high accuracy, that's why weights and biases are saved as 32-bit floating-point numbers. Quantization allows the reduction of numerical precision and saves the values in 8-bit integer format, which results in a fourfold reduction in size [1]. In the proposed algorithm, this process is performed by using the Post-Training Quantization (*PTQ*) technique. The advantage of the quantization technique is a small loss of accuracy.

The developed algorithm for multi-class classification of measurement points (V_{in1}, V_{in2}) in *2D* space uses supervised machine learning with a multilayer perceptron (*MLP*) neural network. The detailed structure of the neural network is shown in Figure 4. It represents the proposed algorithm and consists of an input layer, two hidden layers, and an output layer. The graphical representation of connections in a neural network model makes it easier to understand its architecture and data flow between layers, which is a valuable addition to the text description. The neural network

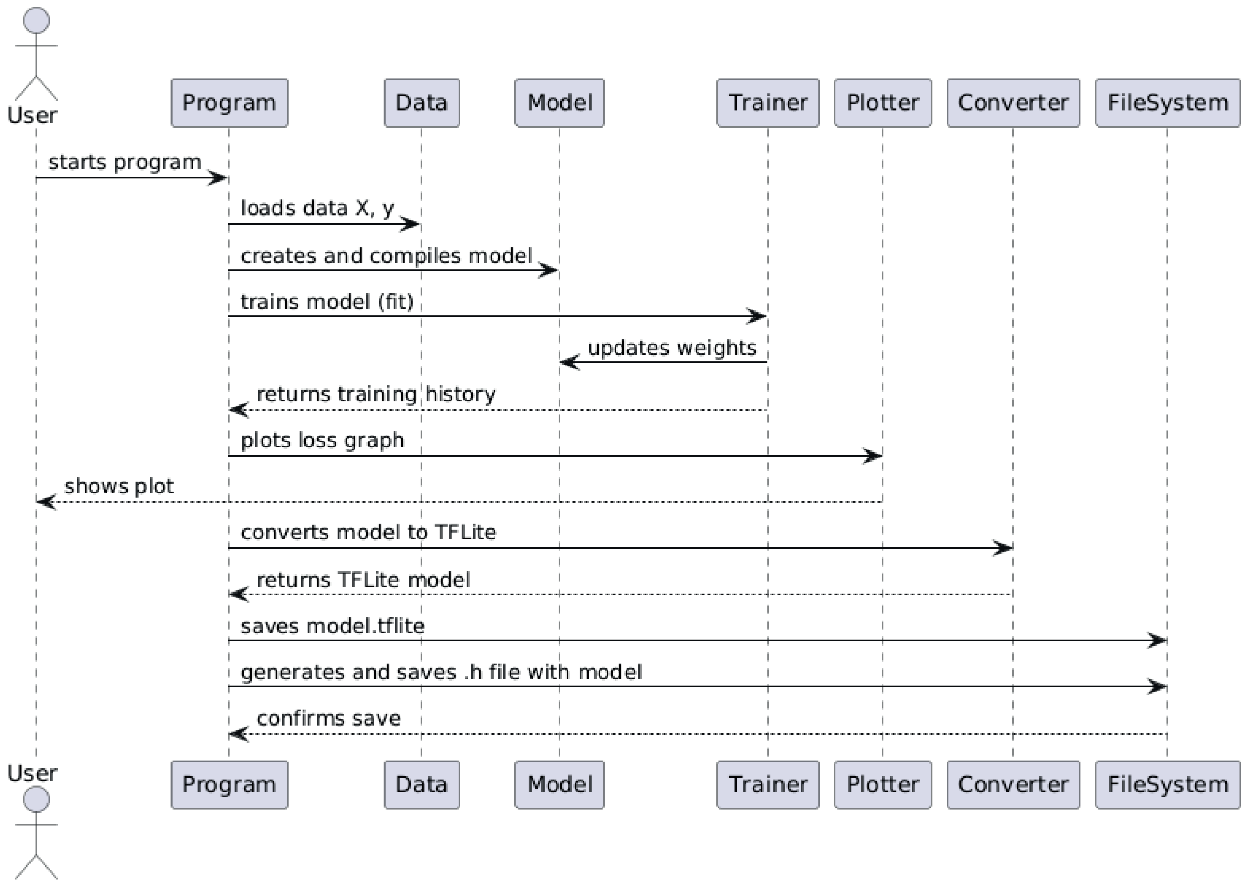

Fig. 2. Sequential diagram of the measurement point classification algorithm

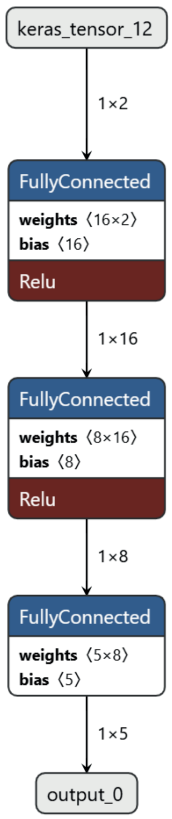

Fig. 3. Neural network model of the algorithm (The value shown next to the arrow indicates the size of the data processed at each stage, in the format (number of samples × number of features per sample))

consists of 2 inputs (V_{in1}, V_{in2}), two hidden layers (16 and 8 neurons) and output with 5 neurons (classes). The model contains a total of 229 trainable parameters (weights and biases). The proposed network is of dense type and uses the *ReLU* (Rectified Linear Unit) activation function in the input and hidden layers. The 2×16×8×5 network architecture is optimal for achieving good classification performance and fast response time. A larger network (e.g. 2×16×32×5) did not provide significant improvement but increased computational complexity.

Surface of the 2D point classification algorithm

The author of the article for the analyzed case (Example 1: Division into sectors of the plane of consideration) divided the considered space of the tested system into an appropriate number of 5 classes. A specific label is assigned to each sector: *LB* – class 2, *RB* – class 0, *LT* – class 1, *RT* – class 4, *C* – class 3 (Figure 5). To verify the operating state of the tested device, two input signals were proposed: the monitored current (V_{in2} – *Y* axis) and voltage (V_{in1} – *X* axis) both measured at its input terminal. The current signal of the tested device was converted to the voltage

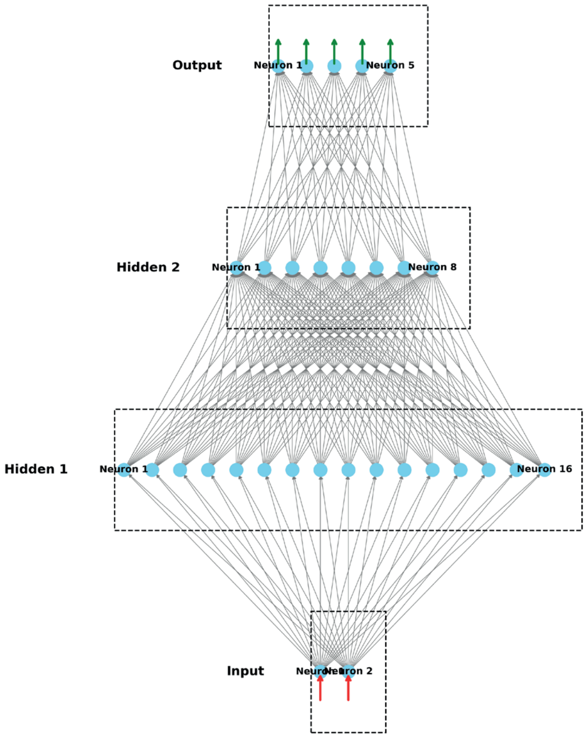

Fig. 4. Neural network model implementing the proposed algorithm

V_{in2} using an appropriate transducer (e.g. electronically assisted current transformer). The division into sectors of the device's operating space was implemented by a trained neural network. The working surface of this device has been normalized and is shown in Figure 5.

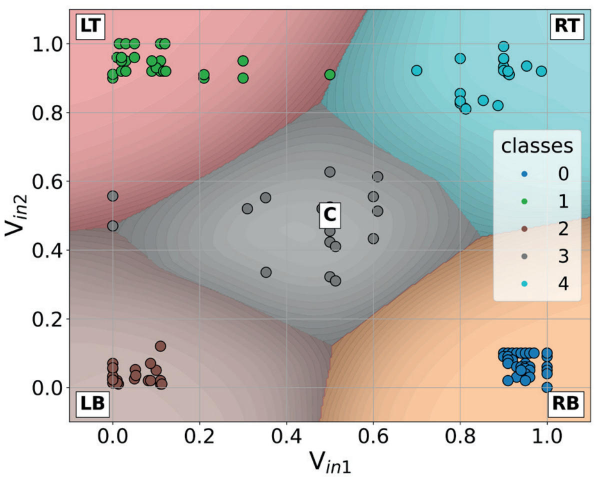

Fig. 5. The consideration space of the monitored system divided into sectors (classes 0 to 4) by the proposed algorithm

The input data to the neural network are *2D* points represented as a *1D* tensor in the form of a vector *input* = *[V_{in1}, V_{in2}]*. The neural network expects an input data of the shape *input_shape=(2,)*. The correctness of the neural network model, designed to classify the measurement points of the tested device (V_{in1}, V_{in2}) was experimentally verified both in the programming environment and through observation of results generated by the Arduino Nano 33 BLE Sense rev2 system.

Neural network model in the Arduino system

In order to implement the trained neural network model on the microcontroller, an algorithm was developed in the *C* programming language. This program reflects the architecture of the model and allows for real-time processing of input data. The structure of the proposed algorithm written in *C* language in a microcontroller system can be written according to the following scheme:

Algorithm: Embedded-ready neural classifier with decision boundaries *2D* in Arduino system (model.ino)

1. Including TensorFlow Lite Micro libraries and model header (*.h*);
2. Defining constants and variables, including the tensor buffer;
3. Initialization of the error reporting system;
4. Loading the *tflite* model converted to *C*;
5. Checking the model version (compatibility with the *TFLite* scheme);
6. Registration of operators used by the model (memory optimization);
7. Create TensorFlow Lite Micro interpreter and allocate memory;
8. Getting pointers to the input and output tensors;
9. In the *loop()*, prediction is performed for for the next input vectors;
10. Analysis of the output results and identification of the class with the highest value;
11. Displaying results.

At the start of the Arduino program, the required TensorFlow Lite for Microcontrollers libraries are included, as well as the header file (.h) from (point 1), which contains the trained model in the form of a byte array. Next, the constants and variables needed to run the algorithm are defined (point 2). The program initializes the error reporting system (point 3) and loads the model from memory (point 4), then checks its compliance with the expected schema version (point 5). In the next step, the operators used by the model are registered (point 6). This allows for limiting the application size and optimizing the embedded system's memory. After creating the TensorFlow Lite Micro interpreter and allocating the working memory buffer (step 7), pointers to the input and output tensors are retrieved (step 8). These pointers are used to perform inference (prediction) in the main program loop (*loop()*) (step 9), based on the currently provided input data. Finally, the classification result is analyzed (point 10) and the class number with the highest value is presented via the serial port of the Arduino microprocessor system (point 11).

Microprocessor in the intelligent system based on machine learning

The electronics market offers efficient and energy-efficient microcontrollers that enable the implementation of algorithms based on machine learning. The Arduino Nano 33 BLE Sense Rev2 microcontroller supports TensorFlow Lite for Microcontrollers (*TFLM*). This system was used in the study of the operation of machine learning algorithms and is characterized by good parameters, such as 1 MB flash memory, 256 KB RAM and a floating-point unit (*FPU*). The proposed system is based on an ARM Cortex-M microcontroller, known for its low power consumption, which makes it an ideal solution for this application.

Software Tools

The author of the work used the Python programming language to build the algorithm. The converted algorithm was compiled and uploaded to the microcontroller by using the *Arduino IDE* program. The structure of the neural network was generated by using the *Netron* program. The *PSoC Creator 4.4* software was used to build the input signal processing circuit in the measurement module.

The results

The proposed algorithm was verified by providing real signals to the input of the measurement module and observing the output signals using the serial port monitor of the microcontroller where the classifier is running (Figure 6).

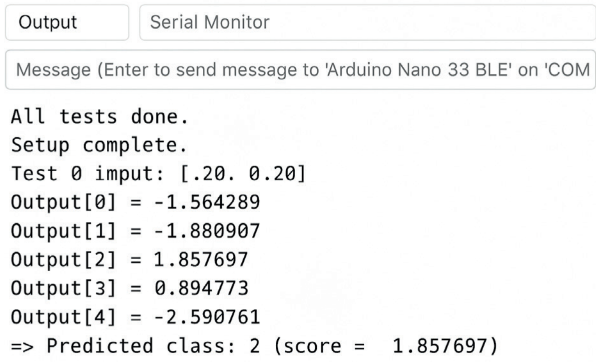

Fig. 6. Arduino serial port window

The author of the present paper checked all the areas of individual sectors of the surface, as shown in Figure 5. The obtained results confirm the correct operation of the algorithm.

Conclusions and further development

The presented approach confirms that machine-learning-based diagnostic algorithms can be successfully deployed on resource-constrained microcontroller platforms, enabling real-time analysis and decision making directly at the edge. The solution reduces communication overhead, shortens reaction times, and lowers maintenance costs by allowing immediate detection of incorrect operating states of the tested device without relying on a central server.

Further development may focus on several directions. More advanced optimization and quantization techniques could be applied to decrease memory usage while preserving classification accuracy. Hardware acceleration through the use of dedicated neural processing units could enable even faster inference and lower power consumption. An example of hardware support includes processors with NPUs (Neural Processing Units) capable of performing 0.5 TOPS (Tera Operations Per Second). Infineon's *PSoC* Edge devices are modern microcontrollers designed for applications requiring advanced *AI/ML* processing at the edge of the network (edge computing). They are particularly suitable for applications such as smart *IoT* devices, speech recognition, image analysis, presence detection, and human-machine interfaces. These devices are a focus for future research aimed at optimizing performance. Expanding the dataset and exploring adaptive or online learning methods would allow the system to update its model as operating conditions change. In addition, integrating secure wireless update mechanisms and standardized industrial communication protocols would facilitate large-scale deployment of the proposed system in Industry 4.0 environments and beyond.

Received: 23.10.2025, **Accepted:** 07.11.2025, **Published:** 24.11.2025

REFERENCES

[1] Pete Warden, Daniel Situnayak, TinyML. Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers, O'Reilly Media, Inc., (2019)

[2] Partha Pratim Ray, A review on TinyML: State-of-the-art and prospects, Journal of King Saud University – Computer and Information Sciences, 34 (2022), 1595–1623

[3] Piotrowski, P. Analiza zastosowań sztucznych sieci neuronowych do krótkoterminowego prognozowania mocy oraz produkcji energii elektrycznej w systemach fotowoltaicznych, Przegląd Elektrotechniczny, 91 (2015), 162–165

[4] Lin J., Zhu L., Chen W.M., Wang W.C., Han S., Tiny Machine Learning: Progress and Futures. IEEE Circuits Syst. Mag., 23 (2023), 8–34

[5] Karras A., Giannaros A., Karras C., Theodorakopoulos L., Mammassis C.S., Krimpas G.A., Sioutas S. Kowalski J., TinyML Algorithms for Big Data Management in Large-Scale IoT Systems, MDPI, Future Internet, 16, 42 (2024)

[6] Sanchez-Iborra, R., Skarmeta, A.F., TinyML-Enabled Frugal Smart Objects: Challenges and Opportunities. IEEE Circuits Syst. Mag., 20 (2020), 4–18

[7] Schizas N., Karras A., Karras C., Sioutas S., TinyML for Ultra-Low Power AI and Large Scale IoT Deployments: A Systematic Review, MDPI, Future Internet, 14, 12 (2022)

[8] Immonen R., Hämäläinen T., Tiny Machine Learning for Resource-Constrained Microcontrollers, Hindawi, J. Sens. 2022, (2022)

[9] Capogrosso L., Cunico F., Cheng D.S., Fummi F., Cristani M. A., Machine Learning-Oriented Survey on Tiny Machine Learning, IEEE Access, 12 (2024)

[10] Pawłowski, S., Mączka, M. Optimisation of QCL Structures Modelling by Polynomial Approximation. Materials (2022), 15, 5715. https://doi.org/10.3390/ma15165715

[11] Mączka, M. Hałdaś, G. Calculations of transport parameters in semiconductor superlattices based on the Greens' functions method in different Hamiltonian representations, Bulletin of the Polish Academy of Sciences Technical Sciences, (2019), 67, 3. https://doi.org/10.24425/bpasts.2019.129661; ISSN 2300-1917