Received September 1, 2019, accepted October 5, 2019, date of current version December 23, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2953019

Predictive Maintenance of Induction Motors Using Ultra-Low Power Wireless Sensors and Compressed Recurrent Neural Networks

MICHAŁ MARKIEWICZ[1,2], MACIEJ WIELGOSZ[3], MIKOŁAJ BOCHEŃSKI[2], WALDEMAR TABACZYŃSKI[2], TOMASZ KONIECZNY[2], AND LILIANA KOWALCZYK[2]

[1]Faculty of Mathematics and Computer Science, Jagiellonian University, 30-348 Cracow, Poland
[2]Centre for Advanced Materials and Technologies CEZAMAT PW Sp. z o.o., 02-822 Warsaw, Poland
[3]Faculty of Computer Science, Electronics and Telecommunications, AGH University of Science and Technology, 30-059 Cracow, Poland

Corresponding author: Michał Markiewicz (markiewicz@ii.uj.edu.pl)

This work was supported by ECSEL JU (Electronic Component Systems for European Leadership Joint Undertaking) under Grant 692519 – PRIME (Ultra-Low Power Technologies and Memory Architectures for IoT).

ABSTRACT In real-world applications – to minimize the impact of failures – machinery is often monitored by various sensors. Their role comes down to acquiring data and sending it to a more powerful entity, such as an embedded computer or cloud server. There have been attempts to reduce the computational effort related to data processing in order to use edge computing for predictive maintenance. The aim of this paper is to push the boundaries even further by proposing a novel architecture, in which processing is moved to the sensors themselves thanks to decrease of computational complexity given by the usage of compressed recurrent neural networks. A sensor processes data locally, and then wirelessly sends only a single packet with the probability that the machine is working incorrectly. We show that local processing of the data on ultra-low power wireless sensors gives comparable outcomes in terms of accuracy but much better results in terms of energy consumption that transferring of the raw data. The proposed ultra-low power hardware and firmware architecture makes it possible to use sensors powered by harvested energy while maintaining high confidentiality levels of the failure prediction previously offered by more powerful mains-powered computational platforms.

INDEX TERMS IoT, Internet of Things, smart sensors, predictive maintenance, compressed recurrent neural networks, RNN, edge computing, induction motors, bearing faults, vibration signature, electric motors.

I. INTRODUCTION

Around 50% of all energy generated in the world is consumed by electric induction motors [1]. This leads to increased interest in the topic of fault prediction and detection [2]. Typically, automatic monitoring of a motor's condition starts with the acquisition of raw data, which is then fed into an artificial intelligence system comprising of genetic algorithms, fuzzy logic, artificial neural networks, expert systems or other signal processing techniques [3]–[9]. As the cost of condition monitoring systems impedes their implementation, there are attempts to develop low-cost solutions. One approach is to decrease the amount of computational power required to process the data, as well as the bandwidth needed for data transmission. Such a solution involves using low-energy IoT devices capable of processing raw data locally, without the need to transfer said data [10]. In this article we propose a novel, low-cost system that is capable of detecting selected types of induction motor faults by utilizing compressed recurrent neural networks. The energy usage of said system is so low that it is possible to power the smart sensor using energy harvested from a small photovoltaic module or thermoelectric generator. The article is organized as follows:

- In section II, we enumerate different types of faults that occur in induction motors, then focus on the most common type of failure: bearing faults, and then discuss the applicability of vibration analysis to bearing condition monitoring.
- In section III, we describe artificial intelligence techniques that can be used to predict bearing faults.

The associate editor coordinating the review of this manuscript and approving it for publication was Min Jia.

- In section IV, we introduce the dataset on which is used to train the neural networks designated to work on low power devices,
- In section V, we describe techniques that can be used to minimize the power consumption and memory footprint of said techniques.
- In section VI, we propose ultra-low power architecture as a computational platform for running proposed artificial intelligence algorithm,
- Finally, in section VII, we show the results of our experiments including tests of wireless sensor with ultra-low power consumption used for predictive maintenance of induction motors,
- In section VIII, we discuss the obtained results.

II. STATE OF THE ART

Research shows that 75% of all induction motor failures are caused by stator or bearing faults, with about 40% of those being bearing faults [11] (41% – according to IEEE, 42% – according to Electric Power Research Institute EPRI). It follows that rolling elements are critical parts of motors.

Bearing faults can be divided into two categories: localized and distributed [12]. The former group contains cracks, pits and spalls defects on the rolling surfaces mainly caused by fatigue cracks below the surface which may happen after overloading or shock loading the bearings when a small piece of metal is removed due to this shock, resulting in a small pit or spall. Defects from the latter group are mainly caused by a manufacturing error, improper installation or abrasive wear [13]. A varying contact force between rolling elements and raceways due to defects results in an increased vibration level. Despite the fact that vibrations in bearings always occur (because of a finite number of rolling elements to carry the load, which – during rotation – change their position in the load zone, causing a periodical variation of the total stiffness of the bearing assembly [14]), their measurements and analysis are very often used in condition monitoring of bearings [15]. In the next section we describe the techniques used for that.

A. DIAGNOSIS METHODS

In order to perform automatic fault detection, the raw signal is often preprocessed in a procedure known as *feature extraction* [15]. This aims to reduce the amount of data that a diagnostic system would otherwise have to deal with, and instead focus on the relevant aspects thereof, represented by these features. For example, the Fourier transform of the signal could be used as a feature. The following paragraphs provide an overview of methods which can be used with or without feature extraction.

Artificial neural networks, described in more detail in section III, are a machine learning method which has been widely explored with different configurations and features [15]. To a significant extent, the neural networks are not manually engineered, but rely on data gained by automated *learning*. While this does give good results, it also typically leads to a loss in interpretability, i.e. human ability to understand the network's reasoning process [16]. *Expert systems* are one approach to remediating this issue. In their simplest form, they are a set of rules in the form of *IF this THEN that*. This introduces clear interpretability, but may come at a loss of accuracy. *Fuzzy logic*, an extension of Boolean logic in which truth values can be anywhere between 0 and 1, is also used due to its flexibility [17]. Finally, there are *support vector machines*, which – for a given input – map it to a vector in such a way that vectors corresponding to inputs belonging to different categories (in this case: fault or lack thereof) are separated by as wide gap as possible. The use of support vector machines for bearing fault detection is extensively described in [15]. In this paper, we focus on artificial neural network techniques.

III. RECURRENT NEURAL NETWORKS

A. INTRODUCTION TO NEURAL NETWORKS

Neural networks are a popular machine learning approach to pattern recognition which does not require hand-crafting of features to analyze, and usually makes creating a model for data analysis without domain-specific knowledge much easier. They typically consist of a number of layers, each of which is fed with an input vector and produces an output vector, known as the *activation*. The input vector of each layer is some combination of the activations of previous layers and the network's inputs, with some optional additional components discussed later. The output of a neural network is simply the output of its last layer. Each layer performs some mathematical operations, like multiplication by the *weight matrix* followed by addition of the *bias vector* and the application of a (typically) point-wise function known as the *activation function*. Any differentiable function can be used as an activation function.

Neural networks can be further categorized by their architecture, i.e. the way the layers in them are interconnected. The simplest case is the feed-forward neural network, in which each of layer operates on the output of the previous one. The applications of the simples architecture are limited, in part because of its fixed input and output size, meaning that it would be unable to process a sequence of arbitrary length. This is also one of the reasons why we decided to use recurrent neural network (RNN), which additionally are specifically suited for the task of time series analysis. Conceptual comparison of selected neural network types is shown in Fig. 1.

B. INTRODUCTION TO RECURRENT NEURAL NETWORKS

In contrast to other approaches, recurrent neural networks (RRN) operate on the assumption that additional information needed to solve the given task can be extracted from temporal dependencies between observations. They operate in timesteps, with each timestep t bringing in a new observation $x^{(t)}$. Crucially, they are also stateful – they have a way of ‘‘memorizing’’ some information, in the form of a hidden

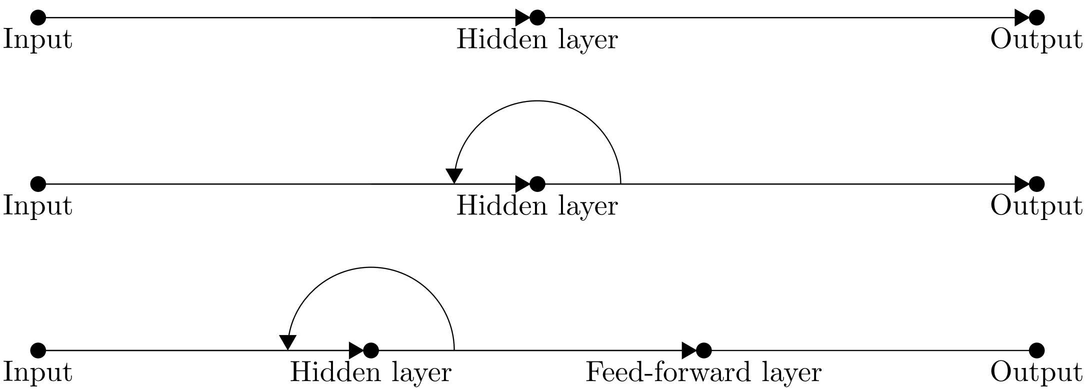


FIGURE 1. Architectures of a standard feed-forward neural network (NN) (top), RNN (recurrent NN) (middle) and mapping of outputs of RNN (bottom).

state $h^{(t)}$, to utilize later. In timestep t, the network has access to $h^{(t-1)}$ and produces $h^{(t)}$. In fact, the network operates recursively on its own output from the precious timestep – and hence the name. The simplest setup of RNN is formally described by equations (1) and (2).

$$h^{(t)} = \psi_h(W_{hx} \times x^{(t)} + W_{hh} \times h^{(t-1)} + b_h) \tag{1}$$
$$y^{(t)} = \psi_y(W_{yh} \times h^{(t)} + b_y) \tag{2}$$

ψ_h and ψ_y are activation functions, W_{hx}, W_{yh} and W_{hh} are weight matrices of the input-hidden, hidden-output and recurrent connections respectively; b_h and b_y are bias vectors. The symbol '×' denotes matrix multiplication.

C. LONG SHORT-TERM MEMORY

In this paper, we use the long short-term memory (LSTM) cell [18] as the basic building block of a recurrent neural network. It is illustrated in Fig. 2. The LSTM proves to be relatively stable, notably it is not suffering from the *vanishing or exploding gradient problem* [19] – a common issue with recurrent neural networks, that causes the strength of the looped-back signal to either decay or grow exponentially. Thanks to that, LSTM cells are highly popular in state-of-the-art models [20].

The LSTM cell is fitted with *gates*, which decide which part of its memory is passed on, and which is overwritten. The control values of gates are in range between 0 and 1 and they are multiplied by corresponding parts of the signal. There are three different gates in each LSTM cell: an input gate (which control value is denoted as $i_c^{(t)}$), an output gate ($o_c^{(t)}$) and a forget gate ($f_c^{(t)}$). The internal values $g_c^{(t)}$ and the persistent state $s_c^{(t)}$ flow in-between the gates. Input and output gates control the flow of inputs into the cell and flow of the cell's outputs into the rest of the network, respectively. The forget gate scales the internal cell state before summing it with the input through the cell's recurrent connection. Thanks to that, the cell is able to control what information is stored for further use.

There are several variants of LSTM architectures, some of which contain *peephole connections* [21]. However, they are not used in this work.

Formally, the operations of LSTM cells is governed by the following vector equations:

$$g^{(t)} = \tanh(W_{gx} \times x^{(t)} + W_{gh} \times h^{(t-1)} + b_g) \tag{3}$$
$$i^{(t)} = \sigma(W_{ix} \times x^{(t)} + W_{ih} \times h^{(t-1)} + b_i) \tag{4}$$
$$f^{(t)} = \sigma(W_{fx} \times x^{(t)} + W_{fh} \times h^{(t-1)} + b_f) \tag{5}$$
$$o^{(t)} = \sigma(W_{ox} \times x^{(t)} + W_{oh} \times h^{(t-1)} + b_o) \tag{6}$$
$$s^{(t)} = g^{(t)} \odot i^{(t)} + s^{(t-1)} \odot f^{(t)} \tag{7}$$
$$h^{(t)} = \tanh(s^{(t)}) \odot o^{(t)} \tag{8}$$

where the symbol '⊙' denotes point-wise multiplication, and '×' denotes matrix multiplication.

The cell's gates are are used to preserve relevant information while discarding everything else. For example, when the input gate is closed (has value of 0), the cell's state – and therefore also its output – is entirely unaffected by the input in the given timestep. To make it possible for the cell to *entirely* discard some information without any leaks, a hard sigmoid function σ is used, which can output 0 and 1 as given by equation (9). Thanks to that, it is possible for gates to be fully opened or fully closed.

$$\sigma(x) = \begin{cases} 0 & \text{for } x \leq t_0, \\ \dfrac{x - t_0}{t_1 - t_0} & \text{for } x \in (t_0, t_1), \\ 1 & \text{for } x \geq t_1. \end{cases} \tag{9}$$

D. CLASSIFICATION WITH LSTMS

In the proposed neural network architecture, multiple LSTM cells are used as the recurrent part of the neural network, followed by a single feed-forward layer which is fed with the outputs of these LSTM cells to produce the final output. This is illustrated in Fig. 1, and formally described by equation (10). This final layer uses the softmax activation function (10), which produces a vector whose elements are non-negative and add up to 1 – each can be interpreted as the probability that the observed input sequence belongs to

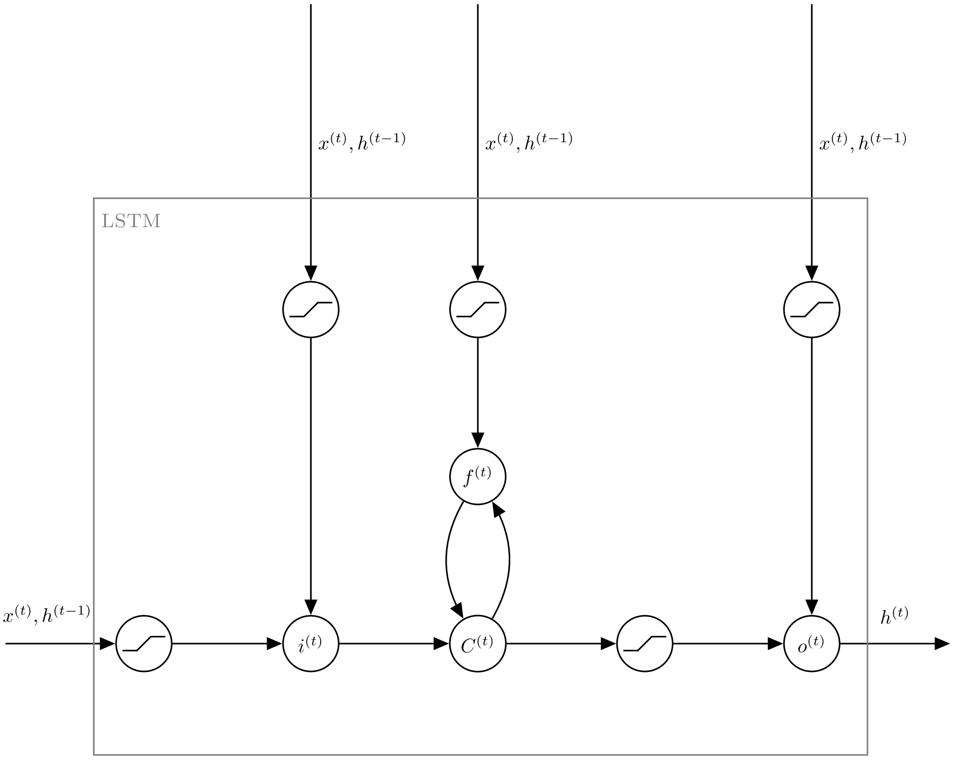


FIGURE 2. The architecture of a LSTM cell.

its corresponding class. In our case, we have two classes – 'normal functioning' and 'anomalous functioning'.

$$\begin{aligned} y^{(t)} &= \text{softmax}(W_{yh}h^{(t)} + b_y) \\ \text{softmax}(x)_i &= \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \end{aligned} \tag{10}$$

Note that the softmax function is not point-wise, it operates on an entire vector.

E. TRAINING NEURAL NETWORKS

Neural networks are typically trained using an algorithm called backpropagation [19]. To used it, a *loss function* is required – it is a measure of how well the neural network is performing, with greater values indicating poorer results. The derivatives of this function with respect to every trainable parameter in the network (weight matrix, bias vector) are calculated, and gradient descent is performed in an attempt to minimize the loss.

In recurrent neural networks a variation of backpropagation known as *backpropagation through time* [20] is used, and number of timesteps for which the gradient is back propagated is called *lookback* [18], [20].

IV. DATASET

We demonstrate the capabilities of the proposed artificial neural network system on the NASA bearing dataset [22]. This dataset consists of the results of three similar experiments carried out on electric induction motors, which were driven for long periods of time until a bearing fault occurred. In each experiment, there were four bearings, each with one or two single axis accelerometers attached. With minor exceptions, measurements were gathered over a period of one second every 10 minutes, at a sampling frequency of 20kHz. The duration of these experiments ranges from less than a week to over a month. In the spectrogram shown in Fig. 3 in the frequency domain it is clearly visible the moment when the fault occurred. Similarly, for acceleration measurements sampled with frequency of 100Hz, as shown in Fig. 4.

We labelled each data sample as coming from a machine that is normal functioning, anomalous functioning (a bearing fault), or unknown. We labelled the first half of each experiment as normal functioning and the last few days (3 in the second experiment, 10% of experiment duration in the first and the third experiments) as anomalous functioning. The samples in between we labelled as unknown. We split the dataset into three parts: one used for training the AI, the second for validating the AI's progress during training, and the third one for testing. The training set consists of about 50% of the data, with the other two having about 25% of data each.

A. TAILORING THE DATASET TO WSN LIMITATIONS

The dataset was acquired using advanced sensors. Wireless sensor nodes that have very constrained power budget so it is not possible to use high sampling frequencies for longer intervals without a risk of draining the battery. For that reason

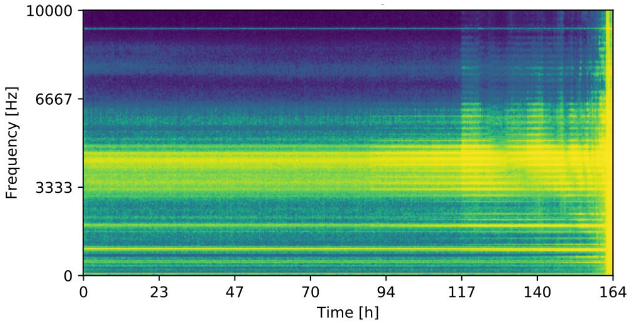


FIGURE 3. Spectrogram of accelerations from test No. 2 with original 20kHz sampling frequency. It is easy to see the moment the bearing fault occurred. Acceleration scale is not linear: 🟨 – 20g, 🟩 – 2g, 🟦 – 0.4g, 🟪 – 0g.

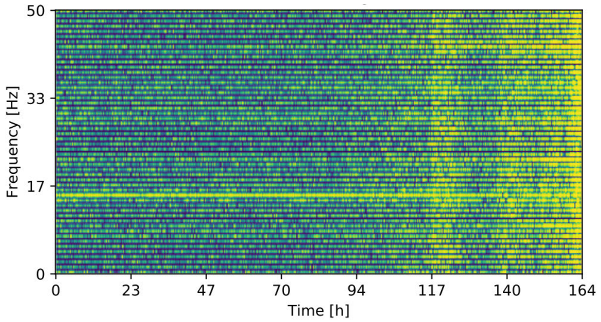


FIGURE 4. Spectrogram of the accelerations from the same experiment as shown in Fig. 3 sampled with frequency of 100Hz. The increase in vibration due to bearing faults remains visible despite much lower sampling frequency.

we downsampled the data to create learning datasets for RNN with sampling frequencies of 100Hz, 200Hz and 400Hz to explore tradeoffs between computational efficiency of processing the data and accuracy of the results.

V. NEURAL NETWORK COMPRESSION FOR WSN

Ultra low power microcontrollers used in wireless sensor nodes have very limited resources in terms of processing capabilities, available memory and time they could spent on computations without the risk of draining the battery. For that reason, a process known as *compression* may be used to trade the network's accuracy for its size, which translates into computational performance. The popular compression techniques are: *pruning*, that minimizes the number of parameters in the model, and *quantization*, which decreases the bit-length of said parameters [23].

A. PRUNING

Pruning aims to remove the parameters of the neural network which contribute the least to overall accuracy. It exploits the fact that the weights of trained neural networks typically follow a predictable pattern: most values are close to 0, with only a few outliers with a significant absolute value. Broadly speaking, it is these outliers that contribute the most to accuracy. There are many pruning strategies [24]–[28]. In the simplest one, for every layer, we set to zero all weights with are close to this value. Given weights $W = [w_0, \ldots, w_k]$ where $w_i \in \mathbb{R}$, the pruned weights are given by $W' = [w'_0, \ldots, w'_k]$ (11). The accuracy-efficiency tradeoff can be controlled by adjusting the *ratio* $\in [0, 1]$:

$$w'_i = \begin{cases} 0 & \text{if } |w_i| < \mathit{ratio} \cdot \max(|W|), \\ w_i & \text{otherwise.} \end{cases} \tag{11}$$

Pruning gives high compression ratios at a negligible loss in accuracy [26]. To benefit from it in terms of computational time it requires support of sparse matrix multiplication.

B. QUANTIZATION

Quantization reduces the bit-length of the parameters, inputs and activations of a neural network. Moreover, this process might include conversion of floating-point weights into integers, which greatly improves computational time, especially on ultra-low power devices. In our work we use linear, minmax, logarithmic and tanh quantizations which are described below.

The *linear quantization* utilizes a zero-centered range which is wide enough for all the original weights to fit into it. The possible values of the quantized weights are distributed evenly within that range; each weight is rounded to the nearest possible value. For a weight matrix W, the original floating-point value of a single weight $w \in W$, the target number of *bits* in fixed-point representation we define *linear* quantization as:

$$\begin{aligned} \operatorname{linear}(w, W) &= s_{\text{lin}}(W) \cdot c(w) \\ c(w) &= \operatorname{clamp}\left(r(w), -2^{bits-1}, 2^{bits-1} - 1\right) \\ r(w) &= \left\lfloor \frac{w}{s_{\text{linear}}} + \frac{1}{2} \right\rfloor \\ s_{\text{lin}}(W) &= 2^{1-(bits-bits_{int}(W))} \\ \mathit{int\_bits}(W) &= \lceil \log_2 \max(|W|) \rceil \\ \operatorname{clamp}(a, \mathit{min}, \mathit{max}) &= \min(\max(a, \mathit{min}), \mathit{max}) \end{aligned} \tag{12}$$

where s_{lin} is a scaling factor (the smallest possible distance between two quantized values), *int_bits* is the number of bits needed to store the integral part, and $\operatorname{clamp}(a, \mathit{min}, \mathit{max})$ is the clamping function.

Minmax quantization uses an interval which bounds are the minimal and maximal value of W:

$$\begin{aligned} \operatorname{minmax}(w, W) &= s(W) \cdot \left\lfloor \frac{w - \min(W)}{s_{\text{minmax}}} + \frac{1}{2} \right\rfloor \\ &\quad + \min(W) \\ s(W) &= \frac{\max(W) - \min(W)}{2^{bits} - 1} \end{aligned} \tag{13}$$

Hyperbolic tangent quantization preserves precision for lower absolute values:

$$\operatorname{thq}(w) = \operatorname{arctanh}\left(s_{\text{thq}} \cdot \left\lfloor \frac{\tanh(w) + 1}{s_{\text{thq}}} + \frac{1}{2} \right\rfloor - 1\right)$$

$$s_{\text{thq}} = \frac{2}{2^{bits} - 1}$$

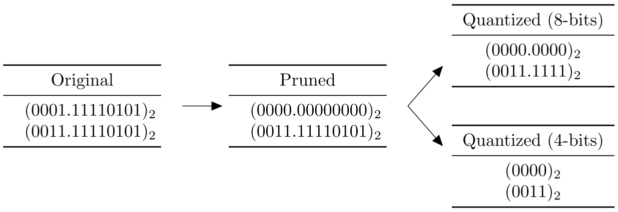


FIGURE 5. **The process of compressing neural network parameters using pruning and quantization techniques to speed up computations on low power devices such as wireless sensor nodes.**

The process described in this section that makes neural network smaller and less demanding in terms of computational power is shown in Fig. 5. In the next section we will present the ultra-low power wireless platform on which the compressed neural networks will be running.

VI. WIRELESS SENSOR NODES WITH AI

The general hardware architecture of a WSN node can be divided into four subsystems [29], [30]:

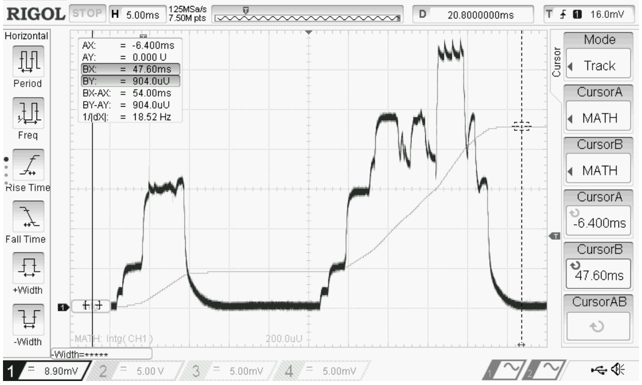
- Communication subsystem for wireless data transmission, consisting of radio transceiver and antenna,
- Computing subsystem for data processing and managing of the node, consisting of microcontroller unit MCU) with processor core, memory, timers and input-output ports,
- Sensing subsystem, for acquiring data about the environment, consisting of a set of sensors communicating with MCU,
- Power subsystem, for providing and regulating power supply voltage, consisting of energy storage, voltage regulation and optionally an energy harvester.

The most fundamental requirement in design of WSN is its energy efficiency, because replacement or recharging of batteries is difficult due to large number of nodes or limited access to the place where they are deployed [31]. Applicability of a WSN is considered adequate if its lifetime is at last several months [30]. This is one of that reasons why sensors used in nodes are mostly low-power, small-sized and factory-calibrated, it helps to unload MCU and keep its computational time at minimum.

The current paradigm of wireless sensor nodes is about gathering raw data and then transmitting it to a more powerful device for further processing. Our approach pushes the boundaries of this paradigm by performing actual processing on the sensor itself in an energy-efficient way, and transmitting only a single number reflecting probability of fault. The firmware and software architecture we used for that purpose have been developed as a part of PRIME project (Ultra-Low Power Technologies and Memory Architectures for IoT).

A. WIRELESS COMMUNICATION

There are several short-range wireless communications standards used in IoT devices including IEEE 802.15.1, IEEE 802.15.3, IEEE 802.15.4 and IEEE 802.11. Each technology makes trade-offs between bandwidth, power consumption, latency and range [32]. Multiple wireless sensor networks use IEEE 802.15.4 protocol for data communication using low-data-rate, low-power radio transmission which is best suited for battery-powered devices working in noisy, industrial environment. [33]. One of them is GreenNet, which has been initially developed by STMicroelectronics and then by Université de Grenoble and Cezamat [34]. We use it in our design at 2.4GHz frequency band. To improve energy efficiency and robustness, it heavily relies on beacon-enabled mode. This helps in minimization of the activity time of the sensor node: it only wakes up at regular time intervals (according to beacons sent by a network coordinator), spending the remaining time in deep sleep mode. This aggressive energy conservation scheme helps in achieving an average energy consumption at a level of tens of μW – despite the fact that, during normal activity, nodes consume tens of mW. Power consumption profile during activity period, i.e. node wakeup, beacon reception, data transmission and transition to sleep mode is presented in Fig. 8.



```
AX:      = -6.400ms
AY:      = 0.000 U
BX:      = 47.60ms
BY:      = 904.0uU
BX-AX:   = 54.00ms
BY-AY:   = 904.0uU
1/|dX|:  = 18.52 Hz
```

FIGURE 6. **Measurements of the power consumption of a WSN node during sensor activity. At first the node wakes up before expected time of arrival of beacon, and prepares the embedded radio for transmission. Then it goes a lower power mode for a while, just to make transition to a run mode, perform radio reception and transmission and then goes back to a deep sleep mode. Current measured as voltage drop over 5Ω shunt resistor. Input voltage: 3V.**

Having insight into the power requirements of radio transmission, we might consider when it is more energy efficient to analyze the data locally and when to transmit them as they are. Data transmission of a single packet as shown in Fig. 8 consumes about 0.5mJ of energy. Suppose that the sensor (three-axis accelerometer) is configured to have sampling frequency of 100Hz. It means that the sensor generates one hundred triplets describing acceleration along X, Y and Z axis (plus one hundred and one if temperature is also registered) every second. Even if only the 8 most significant bits of each number are taken, one second of sensing generates almost a kilobyte of data to be send. Increasing the sampling frequency to 400Hz gives about 32kB. The IEEE 802.15.4 PHY standard only supports frames of up to 127 bytes, of which up to 25 bytes (depending on the addressing scheme) is reserved. This means that transmitting a single sample would require at least 8 packets (32 for 400Hz). In the simplest scenario, when one packet is sent (no Guaranteed Time Slots (GTS) are used, and there is no collision during transmission),

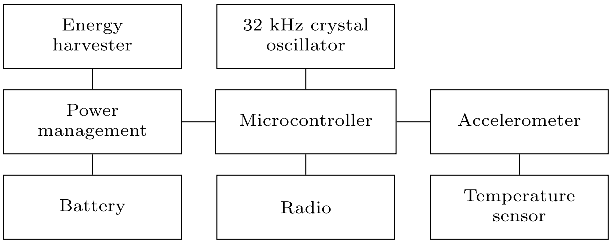


FIGURE 7. Hardware architecture of ultra low power wireless sensor nodes.

it would require 4mJ of energy (or 16mJ for 400Hz sampling frequency). Before we compare this values with the energy consumed by the proposed neural network algorithm that runs locally on a node, we will describe the hardware architecture of the used wireless sensor node.

B. HARDWARE PLATFORM

Well designed wireless node should spend about 99% of the time in sleep mode [30]. It means that overall energy budget is heavily influenced by energy consumed in sleep mode. For that reason we selected a microcontroller (MCU) that supports low-leakage deep sleep modes. The MCU of choice (MKW41Z512) is a system on a Chip (SoC) with a IEEE 802.15.4 radio transceiver, and 48 MHz ARM Cortex-M0+ processor core. It has several sleep modes, with some of them characterised by extremely low power consumption. In a very-low-power stop mode (VLPS) current drawn by the MCU is as small as 3.58μA (buck mode operation for 3V input at 25°C). In a very-low-leakage stop mode (VLLSx) current drawn by the MCU goes to 0.46μA (buck mode operation for 3V input at 25°C) [35].

The sensor used for gathering data about vibrations – ADXL363 – is an ultra-low power sensor consisting of a 3-axis microelectromechanical systems (MEMS): accelerometer and temperature sensor. It consumes less than 2μA at a 100 Hz output data rate. ADXL363 communicates with MCU via a serial port interface (SPI) and provides 12-bit output resolution for accelerometer and temperature. Its sampling frequency could be increased up to 400Hz.

The nodes have two variants of power supply. The first variant has a power subsystem which contains ultra-low power boost regulator with MPPT and charge management which was suitable for photovoltaic (PV) and thermoelectric (TEG) energy harvesting (ADP5090). The second variant of nodes has a connector which allows attaching battery or cables from a laboratory power supply. For energy consumption measurements we used the second variant to have better stability of measurements due to lack of dependency on the availability of ambient light or variations in temperature gradient. The hardware architecture is presented on Fig. 7. The nodes are presented on Fig. 10 and Fig. 11.

C. FIRMWARE

The firmware running on the sensor node is governed by a real time operating system (RTOS) for resource-constrained devices in the Internet of Things (IoT) that contains of a low-power IPv6 communication stack [36]. The main activities of the sensor could be presented in a form of infinite loop:

- Device wake-up (timer-triggered interrupt),
- Data acquisition from accelerometer,
- LSTM calculations,
- Feedforward layer calculations,
- Waiting for a beacon for network synchronization,
- Sending AI computation's results,
- Going back into deep sleep mode.

Wireless sensor nodes are not the best devices for execution of advanced algorithms due to reasons given in the previous sections. What we would like to show, is that they are good enough to do the processing of some artificial intelligence algorithms like compressed recurrent neural network (CRNN), while consuming amount of energy comparable with energy spent on maintaining wireless radio connectivity. To prove that, we have to analyze CRNN processing shown as Algorithm 1.

Algorithm 1 CRNN Data Processing Algorithm

```
 1: $s_{prev} \leftarrow 0$
 2: $h_{prev} \leftarrow 0$
 3: $a \leftarrow getMeasurementsFromAccelerometer()$
 4: **for** $it$ < MEASUREMENTS_COUNT **do**
 5:     $x \leftarrow a_x[i]$
 6:     $g \leftarrow \tanh(W_{gx} \times x + W_{gh} \times h_{prev} + b_g)$
 7:     $i \leftarrow \sigma(W_{ix} \times x + W_{ih} \times h_{prev} + b_i)$
 8:     $f \leftarrow \sigma(W_{fx} \times x + W_{fh} \times h_{prev} + b_f)$
 9:     $o \leftarrow \sigma(W_{ox} \times x + W_{oh} \times h_{prev} + b_o)$
10:     $s \leftarrow g \odot i + s_{prev} \odot f$
11:     $h \leftarrow \tanh(s) \ \odot o$
12:     $s_{prev} \leftarrow s$
13:     $h_{prev} \leftarrow h$
14:     $it \leftarrow it + 1$
15: **end for**
16: $y \leftarrow softmax(W_{yh} \times h + b_y)$
17: $enqueueResultForTransmission(y)$
```

The accelerometer is configured to collect the data for one second periods with sampling frequency of 100Hz. At first MCU reads 100 triplets from accelerometer's buffer containing of the values of acceleration along X, Y and Z axis (line 3). Then MCU iterates over the readouts (line 4), taking the value of acceleration along one of its axis (X in this case) and performing calculations which correspond to equations (3)-(6) described earlier in Section III-C. Finally, the result is processed using softmax function and enqueued for wireless transmission. Time consuming calculations of hiperbolic tangent has been avoided by using of lookup table with pre-computed values. The rest of the computations are mostly adding and multiplications of elements of matrices. And their computational complexity depends on their dimensions.

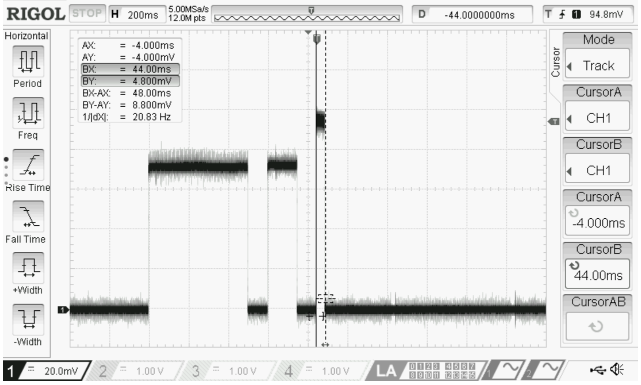


FIGURE 8. Measurements of the power consumption of a WSN node during neural network processing. The probe registers current consumption (voltage drop over 10Ω shunt resistor). Three consecutive blocks represent processing of 100 readouts from accelerometer with three neural networks of size 64, 32 and 16 LSTMs. Sources compiled with –O3 optmization flag. Input voltage: 3V.

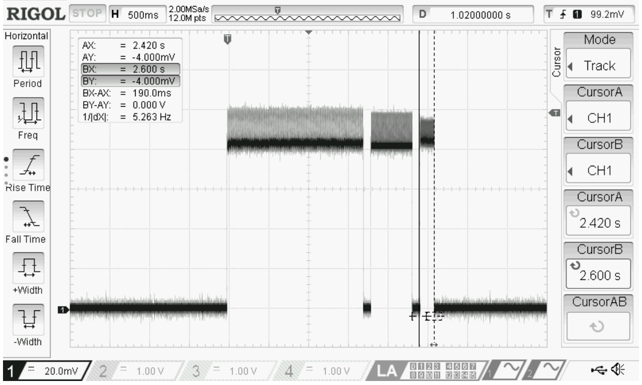


FIGURE 9. Measurements of the power consumption of a WSN node during neural network processing with a non-optimized code (sources compiled with –O0 flag). The current consumption is measured in the same way as described in Fig. 8 for three neural networks of size 64, 32 and 16 LSTMs. Without optimization processing time is more than three times slower.

To illustrate the power consumption during CRNN processing we performed several experiments for different network sizes (we altered the number of LSTMs while keeping number of samples at the same level). The measurements of power consumption for one hundred samples processed by 64, 32 and 16 LSTMs are shown in Fig. 8. Current consumption during processing of 16 LSTMs neural network turned out to be significantly higher than for networks of size 32 and 64 LSTMs. To investigate this phenomenon we compiled the same source code without any optimizations. Measurements of the power consumption shown in Fig. 9 reveal that the peak power consumption for networks of sizes 64, 32 and 16 LSTMs is the same, and similar to the power consumption of NN with 16 LSTMs compiled with optimization flag. Execution of a non-optimized code for 32 and 64 LSTMs NN shows that there are cycles of short (∼0.5ms for 64 LSTMs) periods of high power consumption (when functions like tanh are computed), and long (∼3.5ms for 64 LSTMs) periods of low energy consumption (when matrix multiplications take place). Those cycles are repeated multiple times. Different computational complexity of those two kinds of operations (tanh computation and matrix multiplication) in connection with optimizations performed by compiler, manifests in different power consumption profiles for NN of different sizes as shown in Fig. 8.

TABLE 1. Time, current and energy consumption for neural network processing on WSN node with different number of LSTM.

Number of LSTMs	T [ms]	I_{avg} [mA]	E [mJ]
64	500	6	8.19
32	150	6	2.46
16	44	8	0.52

Table 1 contains the summary of the energy consumption for different sizes of CRNN. We see that for 16 LSTMs, the energy consumption spent on computations is at the same level as energy spent on maintaining regular network connectivity of within WSN.

D. ENERGY BUDGET AND DUTY CYCLE

The rhythm of sleep and activity of a WSN node is strictly related to its network activity. IEEE 802.15.4 defines two kind of devices with different connectivity capabilities. Low-complexity and low-cost Reduced Function Device (RFD), that connect to a single more advanced Full Function Device (FFD), which in contrary can establish multiple connections to RFDs and perform network routing. The wireless network working in a beacon-enabled mode has two parameters $0 \leq SO \leq BO \leq 14$ that define duration and frequency of RFD and FFD network activity. The time between beacons – beacon interval BI is defined as:

$$BI = T_s \times 2^{BO} \tag{14}$$

for a single radio symbol lasting for $t_s = 16\mu s$ in the 2.4GHz frequency band, and duration of base super frame equals to 960 symbols, we obtain $T_s = 960 \times t_s = 15.36$ms. BI determines the periods for which RFD could remain in a sleep mode. The length of super frame duration – SD is defined as:

$$SD = T_s \times 2^{SO} \tag{15}$$

and determines the time when FFD devices have to be active to handle network activity of RFD devices connected to it. From our perspective, more important parameter is beacon interval, because we want our device to be a simple RFD and to save the energy as much as possible, delegating routing activities to other elements of the WSN.

Duty cycle, i.e. the fraction of one beacon period in which the node is active is determined by BI and time spent on data processing. Assuming for simplicity, that every time the node performs network activity it also launches neural network computations, for $BO = 7$ energy for that two activities is spent every $2^7 \times 15.36\text{ms} = 1.98$s for $BO = 8$ – for every 3.93s, and so on. As BO increases, dominating component in energy consumption equation is not neural network processing and wireless transmission but energy spend during

TABLE 2. Energy consumption for different values of BO parameter of 802.15.4 WSN. Energy spent on transmission is equal to 570μJ, energy spent on data processing is equal to 520μJ. Power consumption during sleep time equals to 3.58μA (VLPS at buck mode operation for 3V input at 25°C).

BO	BI [s]	E_{sleep} [μJ]	P_{ave} [μW]	I_{avg} [μA]
7	1.97	0.02	564.89	188.30
8	3.93	0.04	287.81	95.94
9	7.86	0.08	149.28	49.76
10	15.73	0.17	80.01	26.67
11	31.46	0.34	45.37	15.12
12	62.91	0.68	28.06	9.35
13	125.83	1.35	19.40	6.47
14	251.66	2.70	15.07	5.02

sleep time. The summary of energy expenses of a RFD is presented in Table 2. The given values apply to a normal network activity. Energy expenses for device registration within the network during installation is not taken in into account in estimations of average power consumption because it is performed only once.

E. BATTERY POWER SUPPLY

Power subsystem consisting of at least energy storage and voltage regulation module is required to give the autonomy to the node. Non-rechargeable primary batteries like popular lithium coin cell CR2032 in low pulse drain applications like the one described in this paper, will have a capacity near to its average drain [37]. For one-minute activity intervals corresponding to BO=12, the averaged current is below 10μA, which gives more than six months of operations. Using larger battery like CR2450 battery could extend this period to more than six years [37]. The nodes with detachable batteries are presented in Fig. 10.

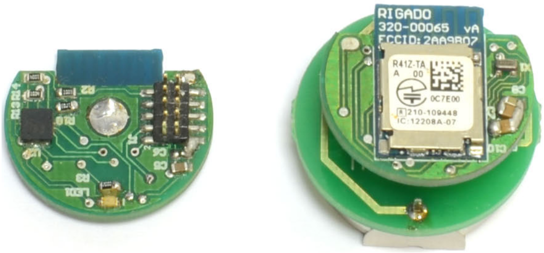


FIGURE 10. IEEE 802.15.4-compatible wireless sensor nodes with ultra-low power accelerometer and detachable power supply. On the right hand side there is a WSN node with a battery attached.

F. ENERGY HARVESTING POWER SUPPLY

Designing the node for unlimited lifetime by powering it by harvested energy seems to be more environment-friendly solution and also dramatically decreases the cost of the maintenance of WSN node. Thermoelectric generators (TEGs) seems to be a good choice, due to temperature gradient created by friction in bearings and induction motor itself. For more information regarding TEG-powered WSN please refer to [30], [38].

The other approach is to use photovoltaic modules (PVs) as a source of energy, which do not require physical contact with the source of heat as in case of TEG. We decided to use this approach. The PV of choice was AM-1801 amorphous silicon solar cell suitable for indoor applications, with typical characteristics 3.0V, 18.5μA at FL-200 lux [39]. In connection with ADP5090 ultra-low power boost regulator with maximum power point tracking (MPPT) and VL-2020 vanadium lithium rechargeable battery they are well suited for work in low-light conditions. The nodes with photovoltaic energy harvester are presented in Fig. 11.

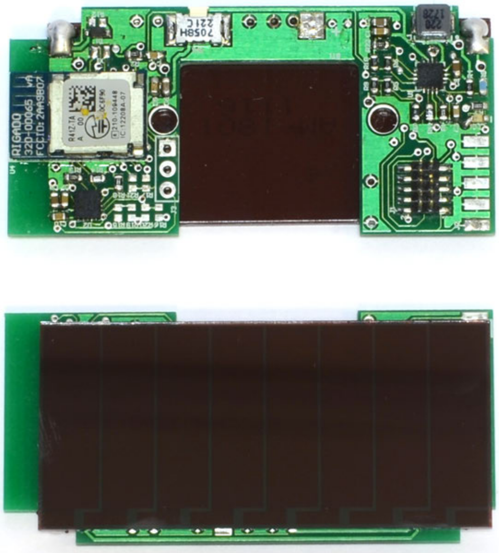


FIGURE 11. IEEE 802.15.4-compatible wireless sensor nodes with ultra-low power accelerometer and photovoltaic cell for energy harvesting.

VII. EXPERIMENTS & RESULTS

To verify the concept of detection of induction motor faults on ultra-low power wireless sensor, we had to harmonize several parameters of the system: power consumption of a node during algorithm processing with power supply provided either by battery or energy harvester; processing time – long enough for performing neural network (NN) computations, but not too long to preserve good battery condition; size of the NN – to ensure it fits into device memory, and not too small to maitain all critical information stored in it.

We will start with explanation of the adjustment of NN parameters.

A. NEURAL NETWORK ADJUSTMENTS FOR WSN

We started our experiments from defining of metrics that would help us in measuring accuracy of fault detection. Thanks to that we were be able to change the number of LSTM cells and compression parameters that directly impact computational time on WSN node and observe how it impacts the accuracy.

We defined the first metric *accuracy* – as proportion of correctly classified samples t to the sum of correctly and incorrectly classified samples $t+f$:

$$\text{accuracy} = \frac{t}{t+f}. \tag{16}$$

TABLE 3. Comparison of accuracy of recurrent neural network with 16 LSTMs, lookback of 32 with other classification methods (one-channel samples).

Algorithm	Accuracy	F1-score
RNN with 16 LSTMs, lookback of 32	91.97%	0.81
RNN with 16 LSTMs, lookback of 32, thq 8-bits	92.04%	0.81
Constant (always classify as OK)	78.80%	0.39
K-NN (K=5)	82.67%	0.31
OC-SVM (linear kernel)	67.62%	0.39
Isolation Forest	88.95%	0.77

TABLE 4. Confusion matrix for uncompressed RNN with 16 LSTMs, lookback of 32 and one-channel data input, and CRNN with the same parameters compressed using thq 8-bits quantization.

Ground truth	Classified as OK		Classified as anomaly	
	Original	Thq 8-bits	Original	Thq 8-bits
OK	74463	74586	3584	4324
Anomaly	4447	3640	17639	17583

Then we defined two auxiliary metrics *recall* and *precision*:

$$\text{recall} = \frac{tp}{tp + fn}, \tag{17}$$

$$\text{precision} = \frac{tp}{tp + fp}, \tag{18}$$

where:

- tp – true positive is the number of items correctly classified as an anomaly,
- fp – false positive is the number of items incorrectly classified as an anomaly,
- fn – false negative is the number of items incorrectly classified as non-anomaly,

and *F1-score* metric as:

$$\text{F1} = 2 \cdot \frac{\text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}}. \tag{19}$$

Initially, we trained the NN with 32 LSTMs, lookback of 64 and with information from four channels using TensorFlow [40], Keras [41] and Analysta [42]. We obtained the accuracy of detection at 96.4%. Then we decreased by half the number of LSTMs and lookback, and reduced the number of channels to one to match the power capabilities of the designed WSN node. We compared the performance of this neural network with the results given by other classification methods (presented in Table 3).

With 91.97% of accuracy we found this result acceptable, but to satisfy our curiosity we also tested several compression methods of this recurrent neural network. We found the hiperbolic tangent 8-bits quantization (thq) the most interesting. The comparisons of confusion matrices of uncompressed and thq-quantized NN is presented in Table 4.

The comparison of several compression methods is presented in Table 5. All accuracy values are reported for the testing set, which is never presented to the AI during training. They are reported for a setup in which every channel has been presented to the NN at some point during training, so the NN did have the opportunity to learn the specific dynamics of every bearing setup. However, in a different experiment in which only data from one induction motor was made available to the NN during training, we did not observe a significant drop in accuracy when running the network on data from another induction motor. It might be surprising that in some cases compressed models – even those having the size of about 25% of the original model – give better results that the uncompressed model. This is possible because the compression may serve as additional normalization step.

TABLE 5. Compression ratio and accuracy of recurrent neural networks with 32 LSTMs, lookback of 32, single channel input (one axis) and 100Hz sampling frequency.

RNN	Ratio [%]	Accuracy [%]
Not compressed	100.0	91.97
Quantized, minmax, 8-bits	25.0	92.04
Quantized, minmax, 4-bits	12.5	90.28
Quantized, minmax, 2-bits	6.2	62.06
Quantized, linear, 8-bits	25.0	92.19
Quantized, linear, 4-bits	12.5	21.20
Quantized, linear, 2-bits	6.25	39.83
Quantized, thq, 8-bits	25.0	92.04
Quantized, minmax, 8-bits; Pruning, 0.03	20.9	91.19
Quantized, linear, 8-bits; Pruning, 0.03	20.9	91.41
Quantized, minmax, 4-bits; Pruning, 0.01	11.8	90.32
Quantized, linear, 4-bits; Pruning, 0.01	11.8	21.20

TABLE 6. Dependency between sampling frequency, quantization, accuracy and size of the data. Feeding CRNN with a lot of uncompressed data does not lead to any significant improvement of accuracy.

Sampling frequency [Hz]	Quantization [bits]	Data size [bytes]	Accuracy [%]
100	32	400	91.97
100	8	100	92.04
100	4	50	90.32
200	32	800	91.97
200	8	200	92.04
200	4	100	90.32
400	32	1600	91.97
400	8	400	92.04
400	4	200	90.32

After multiple numerical experiments with different recurrent neural network parameters, we decided to use in WSN node a small and robust 8-bits hyperbolic tangent compressed neural network with 16 LSTMs and lookback of 32.

The proposed approach leads to the best performance in a scenario where the computations are preformed locally of the sensor device with limited need for data transfer outside the sensing node. Furthermore, the sampling rate should be the lowest possible in order to meet the energy budget. On the other hand, it should also model the phenomenon with expected quality. We tested several versions of compressed models. The tradeoffs between the size of input data (as a function of sampling frequency and quantization) and accuracy is presented in Table 6. Computing CRNN with a large datasets of accelerometer readouts does not lead to any significant improvement of accuracy.

B. TESTS WITH RUNNING INDUCTION MOTOR

We installed the sensors in machinery similar to one described in NASA dataset [22]. Sensors wirelessly transmitted raw data read from embedded accelerometers and temperature sensor for further analysis. Fig. 12 presents the test site.

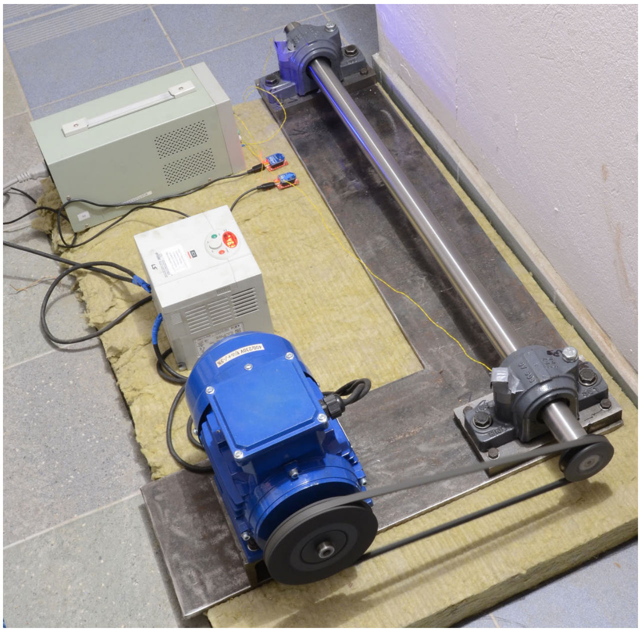


FIGURE 12. Machine with a 1.5kW three phase induction motor. The shaft is installed within a pair of spherical roller bearings (ISO: 22207 CAW33, basing load rating dynamic C: 66500N, static C_0: 76000N, limiting speed with grease lubrication: 5300 rpm). To speed up bearing fault we set the rotational speed to maximal allowed (5300 rpm) and did not install the shaft coaxially, but we slightly misaligned it.

Sensors were powered by a laboratory power supply, to ensure stable power supply. The test shown that the power consumption is at designed levels, nodes work stable and wireless communication is reliable. However, we were not able to confirm accuracy of the neural network because our machine has bearings without oil circulation system to force lubricate them which also regulates the flow and the temperatures of the lubricant. Having such a system would allow us using of a magnetic plug installed in the oil feedback pipe to collect debris from the oil as evidence of bearing degradation. Once the accumulated debris adhered to the magnetic plug exceeds a certain level and causes an electrical switch to close, we would conclude the machinery broken.

VIII. CONCLUSIONS & FUTURE WORK

In this paper we shown that ultra-low power sensors could be used as an artificial intelligence platform for running of compressed recurrent neural networks for conditional monitoring of induction motors. Sensors programmed that way offer great value for their users, because they could work without cloud backend that performs actual data analysis. We also demonstrated that the power consumption of the sensors could be reduced to the values making it possible to power them using harvested energy, minimizing a need of sensor maintenance and eliminating installation of cables. We also proved that a sensor with recurrent neural network could process the raw data, taken from the sensor as it is, without any filtering, preprocessing nor feature extraction.

In the future we would like to investigate possibilities of learning healthy condition at the beginning of machine's life – in a similar way as described in [10]. Moreover, we want to learn how accuracy of fault prediction changes when additional data is provided to the neural network like temperature or current consumption of an induction motor.

REFERENCES

[1] I. Sudhakar, S. AdiNarayana, and M. AnilPrakash, "Condition monitoring of a 3-Ø induction motor by vibration spectrum anaylsis using Fft analyser-a case study," *Mater. Today: Proc.*, vol. 4, no. 2, pp. 1099–1105, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2214785317301256, doi: .10.1016/j.matpr.2017.01.125.

[2] D. Goyal, B. S. P. Vanraj, and S. S. Dhami, "Condition monitoring parameters for fault diagnosis of fixed axis gearbox: A review," *Arch. Comput. Methods Eng.*, vol. 24, no. 3, pp. 543–556, 2017, doi: 10.1007/s11831-016-9176-1.

[3] W. Zhang, M.-P. Jia, L. Zhu, and X.-A. Yan, "Comprehensive overview on computational intelligence techniques for machinery condition monitoring and fault diagnosis," *Chin. J. Mech. Eng.*, vol. 30, no. 4, pp. 782–795, Jul. 2017, doi: 10.1007/s10033-017-0150-0.

[4] G. I. S. Palmero, J. J. Santamaria, E. M. de la Torre, and J. P. González, "Fault detection and fuzzy rule extraction in AC motors by a neuro-fuzzy ART-based system," *Eng. Appl. Artif. Intell.*, vol. 18, no. 7, pp. 867–874, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0952197605000321, doi: 10.1016/j.engappai.2005.02.005.

[5] N. Gindy and A. Al-Habaibeh, *Condition Monitoring of Cutting Tools Using Artificial Neural Networks*, London, U.K.: Macmillan, 1997. pp. 299–304, doi: 10.1007/978-1-349-14620-8_47.

[6] Y.-R. Hwang, K.-K. Jen, and Y.-T. Shen, "Application of cepstrum and neural network to bearing fault detection," *J. Mech. Sci. Technol.*, vol. 23, no. 10, p. 2730, Oct. 2009, doi: 10.1007/s12206-009-0802-9.

[7] V. Sugumaran and K. I. Ramachandran, "Effect of number of features on classification of roller bearing faults using SVM and PSVM," *Expert Syst. Appl.*, vol. 38, no. 4, pp. 4088–4096, Apr. 2011, doi: 10.1016/j.eswa.2010.09.072.

[8] R. Milne, "Artificial intelligence for online diagnosis," *IEE Proc. D, (Control Theory Appl.)*, vol. 134, no. 4, pp. 238–244, Jul. 1987. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/ip-d.1987.0040

[9] B. K. N. Rao, P. S. Pai, and T. N. Nagabhushana, "Failure diagnosis and prognosis of rolling—Element bearings using artificial neural networks: A critical overview," in *Proc. J. Phys., Conf. Ser.*, vol. 364, May 2012, Art. no. 012023, doi: 10.1088/1742-6596/364/1/012023.

[10] S. K. Bose, B. Kar, M. Roy, P. K. Gopalakrishnan, and A. Basu, "ADEPOS: Anomaly detection based power saving for predictive maintenance using edge computing," in *Proc. 24th Asia South Pacific Design Automat. Conf.*, New York, NY, USA, 2019, pp. 597–602, doi: 10.1145/3287624.3287716.

[11] A. Choudhary, D. Goyal, S. L. Shimi, and A. Akula, "Condition monitoring and fault diagnosis of induction motors: A review," *Arch. Comput. Methods Eng.*, vol. 26, no. 4, pp. 1221–1238, Sep. 2019, doi: 10.1007/s11831-018-9286-z.

[12] J. Liu and Y. Shao, "Overview of dynamic modelling and analysis of rolling element bearings with localized and distributed faults," *Nonlinear Dyn*, vol. 93, no. 4, pp. 1765–1798, Sep. 2018, doi: 10.1007/s11071-018-4314-y.

[13] C. Sunnersjö, "Rolling bearing vibrations—The effects of geometrical imperfections and wear," *J. Sound Vib.*, vol. 98, no. 4, pp. 455–474, 1985. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0022460X85902561, doi: 10.1016/0022-460X(85)90256-1.

[14] N. Tandon and A. Choudhury, "A review of vibration and acoustic measurement methods for the detection of defects in rolling element bearings," *Tribology Int.*, vol. 32, no. 8, pp. 469–480, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0301679X99000778, doi: 10.1016/S0301-679X(99)00077-8.

[15] I. El-Thalji and E. Jantunen, "A summary of fault modelling and predictive health monitoring of rolling element bearings," *Mechanical Systems and Signal Processing*, vols. 60–61, pp. 252–272, Aug. 2015, doi: 10.1016/j.ymssp.2015.02.008.

[16] Z. C. Lipton, "The mythos of model interpretability," 2016, *arXiv:1606.03490*. [Online]. Available: https://arxiv.org/abs/1606.03490

[17] L. A. Zadeh, "Fuzzy logic," *Computer*, vol. 21, no. 4, pp. 83–93, Apr. 1988.

[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[19] A. Graves, *Neural Networks*. Berlin, Germany: Springer, 2012.

[20] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," 2015, *arXiv:1506.00019*. [Online]. Available: https://arxiv.org/abs/1506.00019

[21] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017.

[22] A. Djebala, N. Ouelaa, and N. Hamzaoui, "Detection of rolling bearing defects using discrete wavelet analysis," *Meccanica*, vol. 43, no. 3, pp. 339–348, Jun. 2008, doi: 10.1007/s11012-007-9098-y.

[23] M. Wielgosz and M. Karwatowski, "Mapping neural networks to FPGA-based IoT devices for ultra-low latency processing," *Sensors*, vol. 19, no. 13, p. 2981, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/13/2981, doi: 10.3390/s19132981.

[24] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*, D. S. Touretzky, Eds. San Mateo, CA, USA: Morgan Kaufmann, 1990, pp. 598–605.

[25] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal Brain Surgeon," in *Proc. Adv. Neural Inf. Process. Syst.*, 1993, pp. 164–171.

[26] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, vol. 1. Cambridge, MA, USA, 2015, pp. 1135–1143.

[27] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, New York, NY, USA, 2016, pp. 2082–2090.

[28] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A filter level pruning method for deep neural network compression," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 5058–5066.

[29] V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava, "Energy-aware wireless microsensor networks," *IEEE Signal Process. Mag.*, vol. 19, no. 2, pp. 40–50, Mar. 2002.

[30] M. Kuorilehto, M. Kohvakka, J. Suhonen, P. Hämäläinen, M. Hännikäinen, and T. D. Hamalainen, *Ultra-Low Energy Wireless Sensor Networks in Practice: Theory, Realization and Deployment*. Hoboken, NJ, USA: Wiley, 2008.

[31] G. Lu, B. Krishnamachari, and C. S. Raghavendra, "An adaptive energy-efficient and low-latency MAC for data gathering in wireless sensor networks," in *Proc. 18th Int. Parallel Distrib. Process. Symp.*, Apr. 2004, p. 224.

[32] P. Mannion. (2017). *Comparing Low-Power Wireless Technologies*. [Online]. Available: https://www.digikey.com/en/articles/techzone/2017/oct/comparing-low-power-wireless-technologies

[33] *IEEE Standard for Low-Rate Wireless Networks*, IEEE Standard 802.15.4-2015, 2015. [Online]. Available: https://standards.ieee.org/standard/802_15_4-2015.html

[34] L.-O. Varga, G. Rom-aniello, M. Vucinic, M. Favre, A. Banciu, R. Guizzetti, C. Planat, P. Urard, M. Heusse, F. Rousseau, O. Alphand, E. Duble, and A. Duda, "Greennet: An energy-harvesting ip-enabled wireless sensor network," *IEEE Internet Things J.*, vol. 2, no. 5, pp. 412–426, Oct. 2015.

[35] *MKW41Z/31Z/21Z Data Sheet, NXP Semiconductors, 3 2018, Rev. 4*. Accessed: Jul. 6, 2019. [Online]. Available: https://www.nxp.com/docs/en/data-sheet/MKW41Z512.pdf

[36] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki—A lightweight and flexible operating system for tiny networked sensors," in *Proc. 29th Annu. IEEE Int. Conf. Local Comput. Netw.*, Washington, DC, USA, Nov. 2004, pp. 455–462, doi: 10.1109/LCN.2004.38.

[37] Energizer Brands, LLC. (2018). *Lithium Coin Handbook and Application Manual Lithium/Manganese Dioxide—Coin (Li/Mno$_2$)*. [Online]. Available: http://data.energizer.com/pdfs/lithiumcoin_appman.pdf

[38] P. Dziurdzia, "Modelling and Simulation of Thermoelectric energy harvesting processes," *Sustainable Energy Harvesting Technologies: Past, Present and Future*, Kraków, Poland: AGH Univ. Science Technology, 2011, pp. 109–116.

[39] Sanyo. (2007). *Amorphous Silicon Solar Cells*. [Online]. Available: https://media.digikey.com/pdf/Data%20Sheets/Sanyo%20Energy/Amorphous_Br.pdf

[40] *Tensorflow*. Accessed: Jul. 6, 2019. [Online]. Available: https://www.tensorflow.org/

[41] F. Chollet. (2015). *Keras*. Accessed: Jul. 6, 2019. [Online]. Available: https://keras.io/

[42] M. Wielgosz. *Analysta—Data Analysis and Anomaly Detection*. Accessed: Jul. 6, 2019. [Online]. Available: https://bitbucket.org/maciekwielgosz/anomaly_detection

MICHAŁ MARKIEWICZ received the M.Sc. degree from Jagiellonian University, Cracow, Poland, in 2006, and the Dr.-Ing. degree from Bremen University, Bremen, Germany, in 2015. He is currently an Assistant Professor with the Faculty of Mathematics and Computer Science, Jagiellonian University. His research interests include sensor networks, traffic management, and power electronics.

MACIEJ WIELGOSZ received the Ph.D. degree (Hons.) in high-performance reconfigurable computing from the AGH University of Science and Technology (AGH-UST), Krakow, Poland, in 2010. He is currently an Assistant Professor with the Department of Electronics, AGH-UST, and works in the Academic Computing Centre CYFRONET. His research interests include cognitive computing and machine learning.

MIKOŁAJ BOCHEŃSKI, photograph and biography not available at the time of publication.

WALDEMAR TABACZYŃSKI received the M.Sc. degree in mechanics and mechanical engineering from the Faculty of Mechanical Engineering and Aeronautics, Rzeszów University of Technology, in 1999. In IT industry, since 2005, he has been as a Programmer, a Project Manager, and the Head of support department. His primary research interest includes artificial intelligence.

TOMASZ KONIECZNY received the M.Sc. degree in electronics and telecommunications from the Silesian University of Technology, Gliwice, Poland, in 1994. Since 1995, he has been a Designer (Chief Designer or Project Manager) in research and development departments.

LILIANA KOWALCZYK, photograph and biography not available at the time of publication.