Efficient People Counting in Thermal Images: The Benchmark of Resource-Constrained Hardware

MATEUSZ PIECHOCKI, MAREK KRAFT, (Member, IEEE), TOMASZ PAJCHROWSKI, PRZEMYSLAW ASZKOWSKI, AND DOMINIK PIECZYNSKI
Institute of Robotics and Machine Intelligence, Poznan University of Technology, 60965 Poznan, Poland
Corresponding author: Mateusz Piechocki (mateusz.piechocki@put.poznan.pl)

This work was supported by the Polish Ministry of Science and Higher Education under Grant 0214/SBAD/0233.

ABSTRACT The monitoring of presence is a timely topic in intelligent building management systems. Nowadays, most rooms, halls, and auditoriums use a simple binary presence detector that is used to control the operation of HVAC systems. This strategy is not optimal and leads to significant amounts of energy being wasted due to inadequate control of the system. Therefore, knowing the exact person count facilitates better adjustment to current needs and cost reduction. The vision-based people-counting is a well-known area of computer vision research. In addition, with rapid development in the artificial intelligence and IoT sectors, power-limited and resource-constrained devices like single-board computers or microcontrollers are able to run even such sophisticated algorithms as neural networks. This capability not only ensures the tiny size and power effectiveness of the device but also, by definition, preserves privacy by limiting or completely eliminating the transfer of data to the cloud. In this paper, we describe the method for efficient occupancy estimation based on low-resolution thermal images. This approach uses a U-Net-like convolutional neural network that is capable of estimating the number of people in the sensor’s field of view. Although the architecture was optimized and quantized to fit the limited microcontroller’s memory, the metrics obtained by the algorithm outperform the other state-of-the-art solutions. Additionally, the algorithm was deployed on a range of embedded devices to perform a set of benchmarks. The tests carried out on embedded processors allowed the comparison of a wide range of chips and proved that people counting can be efficiently executed on resource-limited hardware while maintaining low power consumption.

INDEX TERMS Benchmark testing, deep learning, edge computing, neural networks, thermal imaging, microcontrollers, performance evaluation, signal processing, TinyML.

I. INTRODUCTION
In recent years, solutions that use artificial intelligence (AI) algorithms have been developing rapidly. Both machine learning and deep learning applications are becoming a part of daily life, taking advantage of access to a vast amount of data [1] and becoming the foundation of smart cities [2], [3], [4]. Furthermore, with the era of Industry 4.0, the Internet of Things (IoT) sector is also quickly expanding [5], [6], with 14.6 billion connected devices in 2021 [7]. Low-cost, tiny microcontrollers (MCUs) or single-board computers (SBCs) are more and more popular in both home and industrial applications [8], [9] and one of the crucial reasons is the increase in their computing power [10]. Related to the above themes is the paradigm of edge computing, whose principal concept is to store data and perform computations close to the source of data [11]. Following [12], this idea is an extension of cloud computing intended for small-scale, real-time intelligent analysis, which has to meet strict time criteria of local services. This approach is facilitated by processing data onboard without uploading them to the cloud, thereby considerably reducing latency and improving the bandwidth efficiency of the network. Therefore, applications that use edge computing have plenty of potential benefits that Sanchez-Iborra et al. [13] recognize as energy efficiency, low cost and latency, system reliability, and data security. Additionally, contemporary MCUs enable the onboard execution of more and more sophisticated algorithms, even ones such...
as neural networks. Such use cases are associated with the TinyML term, which refers to the machine learning application on inexpensive and resource-constrained devices [14], [15]. Shafique et al. [16] enumerate the TinyML spectrum of applications as healthcare, surveillance and security, smart things (always-on wake modules), industrial monitoring and control, finance and administration. The listed leading use cases are: keyword spotting, image classification, visual wake words, object detection, anomaly detection, semantic segmentation, motor control, gesture recognition, forecasting, face recognition, and activity detection [17]. Amato et al. [18] presented an efficient method to monitor car parking occupancy based on visual detection with a convolutional neural network (CNN). The application was deployed on a Raspberry Pi platform with a camera that enables the supervision of up to fifty parking spaces. T’Jonck et al. [19] proposed an accelerometer-based non-invasive way of monitoring seniors’ specific movements. Their strategy profits from the CNNs architecture attaining superior classification results. Furthermore, the neural network was deployed and successfully validated on the nRF52 development kit.

However, MCUs have several constraints that require consideration when creating a solution based on them. Dutta et al. [20] specify challenges of TinyML applications such as inconsistent power usage, memory limitations, processor power, or cost reduction in multisensor applications. Besides these aspects, Banbury et al. [21] also indicate hardware and software heterogeneity as vital factors, which preclude straightforward and direct benchmarking of MCUs, due to different architectures (diverse performance, power, and capabilities) and model deployment (hand-coding, code generation, and ML interpreters).

In this paper, we proposed and tested the occupancy counting method based on low-resolution thermal images. The algorithm utilizes the lightweight encoder-decoder neural network architecture [22] with standard layers and operations. The approach used for people counting shares similarities with other state-of-the-art solutions. Raykov et al. [23] present the first noteworthy publication. In their work, motion patterns were extracted from raw sensor data utilizing an infinite hidden Markov model (iHMM). Those patterns were then used to infer the number of occupants with statistical regression methods. In [24] Leech et al. proposed a Bayesian machine learning algorithm to estimate room occupancy using a single analog passive infrared (PIR) sensor. The method was implemented and deployed on a microcontroller unit, and with a typical size battery the aforementioned system could run continuously for more than a month. Abedi and Jazizadeh [25] introduced a deep learning method that returns binary information about the presence in the measurement area without specifying the number of people. Another approach related to occupancy counting is described by Metwaly et al. [26]. In their application, the task is defined as a classification with the classes corresponding to consecutive natural numbers. This solution has a significant drawback in the form of an upper range limit, which disallows reading the higher value from the frame than a number of classes provided during a training phase. Kraft et al. [27] introduced a method based on density estimation. This concept is borrowed from the crowd counting task [28], which is typically implemented using an RGB sensor.

The presented work extends research conducted in [27] with examinations related to algorithm optimization and quantization for edge and embedded applications using various methods, frameworks, and tools. In addition, validation, performance tests, and comparison of low-cost hardware with resource limitations are significant parts of this study as in [26] and [29]. The principal contributions of this work are as follows:

- the investigation of the influence of the shallow and plain encoder-decoder network on training and test processes,
- the comparison to state-of-the-art algorithms that show the improvement of metrics and the impact of a proposed solution,
- the implementation of the model on a range of low-cost, resource-constrained hardware devices enabling sufficiently fast analysis of thermal images,
- the comparison of performance measurement and power consumption among tested hardware.

The paper is organized as follows: Section 2 describes the Thermo Presence dataset and the method for the density map creation based on corresponding thermal images. Additionally, the chapter defines the neural network architecture with training process specifications and includes benchmark hardware characteristics. Section 3 describes the evaluation methodology and details of hardware efficiency measurements. Section 4 presents the results and a comparison to other state-of-the-art methods in terms of accuracy, processing speed and efficiency. Finally, Section 5 summarizes the work and characterizes the pathways for further development.

II. METHODOLOGY

A. DATASET

This research uses the Thermo Presence dataset published as part of a [27] research paper. Thermal images were collected using a single-board computer - Raspberry Pi 4. The measurement system setup contained an RGB camera and a thermal sensor. The camera had an IMX219 CMOS sensor and was used to control the experiment and annotate the data. Meanwhile, the core part was a MLX90640 far infrared thermal sensor with a 32 × 24 pixel thermal IR array. The dataset’s images were gathered with a 2 Hz frequency rate. According to the datasheet [30], the measured temperature ranges from -40°C to 300°C with ±1°C resolution for each pixel. The sensor was communicating with I2C interface, which is an additional advantage because it is widely available in microcontrollers.

The Thermo Presence dataset includes thermal images registered in the office space with corresponding annotations of persons’ locations. The collection, in total, consists of 13,634 labeled examples from several distinct localizations.
The number of people in each frame was in the range of 0 to 5. The recorded images were grouped into sequences and then split according to the dataset distribution file without stratifying the number of people. Detailed data distribution with a division into training, validation, and test sets is presented in Table 1.

Every frame in the dataset incorporates information about a person’s center X and Y coordinates. Like the authors of [27], the ground truth mask is constructed by setting a single pixel, corresponding to an individual location, to a maximum value. Subsequently, the occupancy density map was created using the convolution of the above-mentioned image with a 2D Gaussian mask ($\sigma = 3$). Due to the relatively constant perspective across the data collection, the standard deviation was selected empirically, based on observations, to fit the blob representing a person in thermal image as thoroughly as possible. The procedure creates a Gaussian mixture distribution with maximums in the central locations of people and the mask radius equal to $3\sigma$ that covers all significant values of the mask. The visualization of a single Gaussian mixture distribution with annotated pixel values is presented in Figure 1.

The output density maps have the exact dimensions as the thermal input images. The aggregated value of the single Gaussian output varies between 45 and 55. It depends on whether the person is in the center or on the edge of the sensor field of view. The average value, adopted as a factor in further computations, is calculated using the training data according to the formula 1.

$$\alpha = \frac{\sum_{h=1}^{n} \sum_{i=1}^{24} \sum_{j=1}^{32} y_{hij}}{\sum_{h=1}^{n} l_{h}}$$

where the numerator describes the sum of the values of the created ground truth masks, and the denominator is related to the sum of corresponding reference people counts. In the above equation, the $y_{hij}$ denotes the i-th row and the j-th column of the h-th ground truth mask from a training set, whereas $l_{h}$ is a ground truth people count for an h-th image. The $n$ defines the number of samples in the training set. For the dataset distribution adopted in the study, this value is equal to 51.35. The above-defined coefficient is used in the evaluation phase. The overall person count in the output mask is calculated by adding all elements of the predicted density map and dividing by the aforementioned $\alpha$ value.

The example visualization of the density map creation steps for the ground truth example is depicted in Figure 2.

B. NEURAL NETWORK ARCHITECTURE

The density map reconstruction, with the locations of people, requires the neural network with the encoder-decoder architecture. Moreover, the principle study idea is to deploy an end-to-end people count estimator on edge AI and TinyML devices. This approach requires shifting the main focus from model metrics to hardware-related parameters such as size, number of parameters, and inference time. Therefore, the U-Net [22], a convolutional neural network, was taken into consideration as a suitable component capable of being customized. Initial experiments have shown that even smaller models achieve satisfactory outcomes while allowing the considerable reduction of model complexity and the number of trainable parameters. The selected architecture is detailed in Figure 3. Since the input is a low-resolution, monochromatic

| Dataset     | Number of Persons in a Frame | Total |
|-------------|-----------------------------|-------|
|             | 0   | 1   | 2   | 3   | 4   | 5   |       |
| Training    | 99  | 105 | 2984| 3217| 1953| 114 | 8472  |
| Validation  | 0   | 139 | 631 | 1691| 225 | 139 | 2825  |
| Test        | 162 | 83  | 211 | 341 | 1235| 315 | 2347  |
image, the encoder part is much shallower than the original one, with a single-channel input. The neural network structure consists of convolutional, maximum pooling, upsampling, and concatenation layers, all of which can be translated to a wide range of embedded devices. The hyperparameter tuning was applied using a grid search algorithm to optimize model results. Finally, all convolution layers have a kernel size of $3 \times 3$, a stride of $1 \times 1$, and 1 pixel of padding to keep the output feature map size the same as the input. The initial number of filters $N$ in the first block of convolutions is equal to 16, and the remaining filters follow the rules depicted in Figure 3. The convolution layers are followed by the Rectified Linear Unit (ReLU) activation functions.

$$\text{ReLU}(x) = \max(0, x)$$ \hspace{1cm} (2)

The downsampled data representation is obtained by using a maximum pooling operation. The function uses a $2 \times 2$ pooling window and a $2 \times 2$ stride that changes the size of the output feature map by half on both axes. The resulting values are the maximum numbers selected from the appropriate pooling windows.

In the training phase, Adam [31] optimizer was utilized, a computationally efficient, straightforward, and relatively easy-to/configure algorithm with an initial 1e-3 learning rate. The mean absolute error (MAE) was chosen as a loss function. The aforementioned modules were implemented using the TensorFlow library. For training purposes, the NVIDIA TITAN Xp with 12 GB memory was utilized and configured with CUDA Toolkit 11.1. The batch size was set to 128 due to the small size of the input data and low model complexity. The maximum number of epochs was equal to 100. However, the early stopping callback, with patience equal to 10, was used to monitor loss function changes on validation dataset and interrupt training if no improvement was observed. Finally, the model was trained for 50 epochs, which took about 6 minutes for the above setup.

C. EVALUATION HARDWARE

With regard to potential applications in heating, ventilation, and air conditioning (HVAC) systems, inference hardware must have tiny dimensions, low weight, and an energy-efficient profile. Therefore, the following devices were chosen:

- **Raspberry Pi 4B** with a quad-core Cortex A72 processor is one of the most popular single-board computers in use. In the utilized configuration device has 2 GB of RAM and runs models in TensorFlow Lite format.
- **Coral USB Accelerator** as an Edge TPU coprocessor enables high-speed machine learning inference. It provides additional computing power to external systems and cooperates, for example, with the Raspberry Pi. The accelerator has an ARM Cortex M0+ and a custom application-specific integrated circuit (ASIC) - Edge TPU. The coprocessor can perform 4 trillion operations per second (TOPS). Using the Edge TPU Compiler, the neural network model in TensorFlow Lite format can be compiled to the form compatible with the Edge TPU.
- **Intel Neural Compute Stick 2 (NCS 2)** is a small USB module designed to accelerate deep learning algorithms in IoT applications. The NCS 2 is intended for a wide range of computer vision tasks. High performance is obtained by utilizing 16 efficient SHAVE (Streaming Hybrid Architecture Vector Engine) VLIW cores (very long instruction word) and a dedicated hardware neural accelerator. This leading part responsible for computation is Intel Movidius Myriad X Vision Processing Unit (VPU) with a 700 MHz base frequency. It is directly supported by the Intel OpenVINO toolkit, which is a
set of tools for optimizing and deploying deep learning models, especially convolutional neural networks. The NCS 2 supports exclusively floating-point operations; thus, models can be converted only to float 32 and float 16 data formats. The accelerator is a coprocessor and requires an external device with an operating system, e.g., a Raspberry Pi.

- **Arduino Nano 33 BLE Sense** is equipped with an effective and energy-efficient microcontroller nRF52480 that is built around the 32-bit ARM Cortex-M4 CPU with floating point unit, running at a 64 MHz clock rate. The chip has 256 KB of RAM and 1 MB of Flash memory. It supports neural network processing using TensorFlow Lite for Microcontrollers (TFLite Micro) optimized kernels.

- **Arduino Portenta H7** is an efficient development board designed for industrial purposes. It consists of an STM32H747 microcontroller based on ARM Pelion technology, which has two cores ARM Cortex-M7 and Cortex-M4 with respectively 480 and 240 MHz clock frequency. Both cores can work independently. The development board supports Arduino, Mbed operating system, and Micropython. Moreover, it is also possible to use TFLite Micro for the implementation of neural networks.

- **STM32 F429ZI Nucleo-144** has a 32-bit ARM Cortex M4 core, clocked at 180 MHz. The microcontroller is equipped with 2 MB of Flash and 256 KB of RAM. The chip can be configured using STM32 CubeMX and the X-CUBE-AI framework enables converting pretrained models for this platform.

- **STM32 H745ZI Nucleo-144** has two cores: ARM Cortex M7 and ARM Cortex M4 whose clocks frequencies are 480 and 240 MHz. The microcontroller has 1 MB of RAM and 2 MB of Flash memory. The platform is supported by the X-CUBE-AI expansion package, which is part of the STM32Cube AI ecosystem.

- **LOLIN32** development board with the Espressif ESP32-WROOM-32 module. The chip has 520 KB of SRAM and 4 MB of Flash memory. Inside is a dual core Tensilica LX6 microcontroller with a 240 MHz clock rate and integrated WiFi and Bluetooth BLE modules. The neural network was also implemented using the TFLite Micro framework with additional support of the ESP-NN library, which contains optimized implementations of kernel functions.

The summarized information about evaluation hardware is included in Table 2. Nevertheless, to run sophisticated and computationally intensive algorithms like neural networks on resource-constrained hardware, the model has to be previously optimized. For this case, conversion and deployment tools are utilized. The most commonly used include:

- **TensorFlow Lite** (TFLite) [32] provides a set of tools to infer machine learning on-device. TFLite achieves, by hardware acceleration and model optimization, high performance on mobile, embedded, and IoT devices. Efficiency on low-cost platforms is obtained in five key aspects: latency of the model, data privacy, no connectivity requirement, reduced size, and lowered power consumption. The TensorFlow models can be easily converted to the TensorFlow Lite FlatBuffers format. This form of the neural network is efficient, portable, and supported by several programming languages.

- **OpenVINO** [33] is an open source toolkit maintained by Intel. It contains tools to optimize and deploy AI algorithms on a wide range of Intel devices, from the cloud to the edge. The toolkit supports hardware such as vision processing units (VPUs), central processing units (CPUs), graphics processing units (GPUs), and field-programmable gate arrays (FPGA). The goal of
the package is to boost algorithm performance in applications related to either computer vision or natural language processing and automatic speech recognition. Officially supports conversion to Intermediate Representation (IR) format from several popular deep learning frameworks such as, e.g., TensorFlow, PyTorch, or ONNX. The toolkit’s principal advantage is the possibility to infer the same IR model among all supported platforms without significant code changes.

- **TensorFlow Lite Micro** (TFLite Micro) [34] is part of the TensorFlow project created to bring deep learning algorithms to microcontrollers. Designers assume that neural networks should run on devices with only a few kilobytes of memory and require neither operating system support nor dynamic memory allocation. The package supports models as a C-byte array that can be converted from TensorFlow Lite using standard system tools like the vxxd package in the Linux operating system.

- **X-CUBE-AI** [35] as an expansion pack is part of the STM32Cube AI ecosystem, which extends the STM32 CubeMX tool. The main feature of the toolkit is automatic conversion of pre-trained deep learning and machine learning models and integrating them with a project as an optimized library. The tool enables several ways of validating algorithms from the evaluation on desktop PC using the emulated environment to performance measurement on STM32 devices with pre-configurated code templates. X-CUBE-AI natively supports deep learning frameworks such as TensorFlow Keras, TensorFlow Lite, and all formats previously converted to ONNX. The package supports 8-bit model quantization and enables easy portability across various series of STM32 microcontrollers.

### III. EVALUATION

#### A. NEURAL NETWORK EVALUATION METRICS

The crucial part of the comparison with other state-of-the-art methods was selecting the set of appropriate evaluation metrics. For this purpose, the mean absolute error (MAE, equation 3) and the mean square error (MSE, equation 4) were chosen.

\[
MAE = \frac{1}{n \cdot 24 \cdot 32} \sum_{h=1}^{24} \sum_{i=1}^{32} \sum_{j=1}^{32} |y_{hij} - \hat{y}_{hij}| 
\]

\[
MSE = \frac{1}{n \cdot 24 \cdot 32} \sum_{h=1}^{32} \sum_{i=1}^{32} (y_{hij} - \hat{y}_{hij})^2 
\]

where \(n\) defines the number of samples in the test set, 24 indicate arrays height and 32 their width. Meanwhile, \(y_{hij}\), \(\hat{y}_{hij}\) denote the value related to the i-th row and the j-th column of the h-th ground truth mask and the predicted occupancy map, respectively.

The approximate number of people in a frame can be calculated, according to Section II-A, as the sum of elements in the output array divided by the constant \(\alpha\), which represents the aggregated value of a single Gaussian output. The mathematical representation is given in equation 5.

\[
count_{pred} = \frac{\sum_{i=1}^{24} \sum_{j=1}^{32} \hat{y}_{ij}}{\alpha} 
\]

where \(\hat{y}_{ij}\) corresponds to the element in the i-th row and the j-th column of the estimated density map. Furthermore, with the use of information on the predicted count, three additional metrics were adopted that are used to deal with the person count directly:

- Counting Mean Absolute Error

\[
CountingMAE = \frac{1}{n} \sum_{h=1}^{n} |count_{gt} - count_{pred}| 
\]

- Counting Mean Square Error

\[
CountingMSE = \frac{1}{n} \sum_{h=1}^{n} (count_{gt} - count_{pred})^2 
\]

- Counting Mean Relative Absolute Percentage Error

\[
CountingMRAPE = \frac{100\%}{n} \sum_{h=1}^{n} \left| \frac{count_{gt} - count_{pred}}{count_{gt}} \right| 
\]

where \(n\) specifies the number of samples in the test set, while \(count_{gt}\) and \(count_{pred}\) are, respectively, the ground truth and the estimated count (according to equation 5) for the h-th image.

Furthermore, in relation to publications with state-of-the-art solutions, standard classification metrics, such as Accuracy and F1 Score, were also calculated during the evaluation phase and are presented in comparison.

#### B. HARDWARE PERFORMANCE EVALUATION

IoT solutions require high energy efficiency due to limited power supply options and often difficult-to-reach work location. With this in mind, the devices’ current and power consumption were measured. For this purpose, a Mooshimeter multimeter with a 9 Hz measurements log rate was utilized.

Another vital aspect is the size of the model. Both microcontrollers and computing accelerators have limited memory, which precludes the use of complex neural network architectures. Therefore, memory usage optimization is a crucial aspect that can be improved during conversion principally by weights quantization. The detailed summary, including model size, depending on the used framework and data type, is given in Table 4.

In IoT applications, devices spend most of the time in a sleep mode waiting for a cyclic interrupt to trigger their action. Moreover, in most cases, there is no need for continuous prediction, and periodic operation is sufficient. In our particular case, person count does not change abruptly, and the control processes using the count as an input variable have large inertia. Thus, instead of a continual approach, two work scenarios were chosen to analyze the efficiency of
TABLE 3. The metrics obtained compared to other state-of-the-art solutions. Mean absolute error, mean square error, and their rounded version (in the scope of people count) for direct comparison with [26] and [27].

| Metric Name       | Results | Kraft et al. Results [27] | Metwaly et al. Results [26] |
|-------------------|---------|---------------------------|-----------------------------|
| MAE               | 0.106   | 0.128                     | *                           |
| MSE               | 0.033   | 0.051                     | *                           |
| Counting MAE      | 0.023   | 0.046                     | 0.316                       |
| Counting MSE      | 0.023   | 0.049                     | 0.458                       |
| Counting MRAPE [%]| 0.81    | 1.7                       | 21.03                       |
| Accuracy          | 0.978   | 0.956                     | 0.755                       |
| F1 Score          | 0.978   | 0.959                     | 0.724                       |
| No. of parameters | 46 577  | 165 441                   | 659 462                     |

TABLE 4. The size of the U-Net model depending on the format and data type.

| Model Format       | Data Type | Model Size [KB] |
|--------------------|-----------|-----------------|
| TensorFlow Keras H5| FP32      | 667             |
| TensorFlow Lite    | FP32      | 192             |
|                    | FP16      | 101             |
|                    | INT8      | 59              |
| TF Lite Micro      | INT8      | 59              |
| X-CUBE-AI          | FP32      | 383             |
|                    | INT8      | 182             |
| Edge TPU           | INT8      | 111             |
| OpenVINO IR        | FP32      | 186             |
|                    | FP16      | 93              |

benchmarked hardware. The total energy consumption was calculated for both 30-second and 5-minute measurement cycles. For this purpose, equation 9 was introduced.

\[ W_{\text{cycle}} = T_{\text{infer}} \cdot P_{\text{infer}} + (T_{\text{cycle}} - T_{\text{infer}}) \cdot P_{\text{sleep}} \]  

(9)

where \( T_{\text{infer}} \) refers to the inference time of the device, \( P_{\text{infer}} \) specifies the average power consumption during the prediction phase, \( T_{\text{cycle}} \) defines the cycle time and \( P_{\text{sleep}} \) is related to the power consumption in sleep mode in regards to the device. The Raspberry Pi, in general, does not have any selectable power modes thus energy consumption was optimized regarding to instructions described in [36].

However, to have a good comparison between chosen work cycles, the daily energy consumption for both options was calculated according to equation 10.

\[ W_{\text{daily}} = d_{\text{cycles}} \cdot W_{\text{cycle}} \]  

(10)

where \( d_{\text{cycles}} \) means number of cycles per day and \( W_{\text{cycle}} \) defines energy consumption during the one operation cycle.

IV. RESULTS

The introduced neural network achieves good results, even though the architecture is straightforward and adapted to limited memory hardware. To compare with other state-of-the-art methods proposed in [26] and [27], respectively, fully connected and encoder-decoder structures were implemented and evaluated, using the same training, validation and test splits, on the Thermo Presence dataset. However, the architecture presented in [26] as the top performer - one dense layer with 512 units - achieves very poor results and turns out to be insufficient for comparison on a more diversified dataset such as the one used in the examination. Therefore, the more complex architecture, a fully connected model with two hidden layers of 512 nodes each, was selected. Whereas, the U-Net structure described in [27] was implemented according to the diagram presented in the publication.

The complete comparison, including mean absolute error (MAE), mean square error (MSE), counting mean absolute error (Counting MAE), counting mean square error (Counting MSE), counting mean relative absolute percentage error...
### TABLE 5. Average inference time of the model and evaluation metrics for the tested hardware.

| Device                        | Data Type | Avg. Inference Time [ms] | MAE   | MSE   | Counting MAE | Counting MSE | Counting MRAPE [%] |
|-------------------------------|-----------|--------------------------|-------|-------|--------------|--------------|-------------------|
| Arduino Nano 33 BLE Sense     | INT8      | 1430.125 ± 1.143         | 0.107 | 0.034 | 0.023        | 0.024        | 0.82              |
| Arduino Portenta H7           | INT8      | 137.494 ± 0.500          | 0.107 | 0.036 | 0.023        | 0.024        | 0.82              |
| STM32 F429ZI Nucleo-144       | INT8      | 230.939 ± 0.100          | 0.120 | 0.043 | 0.036        | 0.038        | 1.14              |
| STM32 H745ZI Nucleo-144       | FP32      | 165.983 ± 0.100          | 0.106 | 0.033 | 0.023        | 0.023        | 0.81              |
|                               | INT8      | 53.260 ± 0.100           | 0.120 | 0.043 | 0.036        | 0.038        | 1.14              |
| LOLIN32                       | INT8      | 840.442 ± 0.021          | 0.106 | 0.034 | 0.023        | 0.023        | 0.81              |
| Raspberry Pi 4B               | FP32      | 7.707 ± 0.591            | 0.106 | 0.033 | 0.023        | 0.023        | 0.81              |
|                               | FP16      | 7.690 ± 0.558            | 0.106 | 0.033 | 0.023        | 0.023        | 0.81              |
| Raspberry Pi 4B +             | INT8      | 4.194 ± 0.052            | 0.120 | 0.043 | 0.038        | 0.039        | 1.18              |
| Coral USB Accelerator (std)   | INT8      | 0.605 ± 0.044            | 0.120 | 0.043 | 0.037        | 0.038        | 1.16              |
| Raspberry Pi 4B +             | INT8      | 0.570 ± 0.060            | 0.120 | 0.043 | 0.037        | 0.038        | 1.16              |
| Coral USB Accelerator (max)   | INT8      | 0.570 ± 0.060            | 0.120 | 0.043 | 0.037        | 0.038        | 1.16              |
| Raspberry Pi 4B +             | FP32      | 2.630 ± 0.159            | 0.134 | 0.043 | 0.028        | 0.029        | 0.92              |
| Intel Neural Compute Stick 2  | FP16      | 2.300 ± 0.100            | 0.134 | 0.043 | 0.028        | 0.029        | 0.91              |

### TABLE 6. Evaluated hardware current and power consumption.

| Device                        | Idle Avg. Power Consumption [mW] | Efficient Mode Avg. Power Consumption [mW] | Inference Avg. Power Consumption [mW] | Inference Max Power Consumption [mW] |
|-------------------------------|----------------------------------|-------------------------------------------|---------------------------------------|---------------------------------------|
| Arduino Nano 33 BLE Sense     | 110.93                           | 79.88                                     | 116.62                                | 140.23                                |
| Arduino Portenta H7           | 886.20                           | 514.75                                    | 881.08                                | 938.05                                |
| STM32 F429ZI Nucleo-144       | 682.98                           | 517.81                                    | 740.31                                | 813.98                                |
| STM32 H745ZI Nucleo-144       | 1020.87                          | 638.66                                    | 1136.79                               | 1315.24                               |
| LOLIN32                       | 69.90                            | 28.53                                     | 101.93                                | 127.86                                |
| Raspberry Pi 4B               | 1702.29                          | 1798.76                                   | 2994.96                               | 4230.83                                |
| Raspberry Pi 4B +             | 3116.54                          | 3223.16                                   | 4176.18                               | 5367.91                                |
| Coral USB Accelerator         |                                  |                                           |                                       |                                       |
| Raspberry Pi 4B +             | 2466.30                          | 2479.11                                   | 4447.74                               | 5407.87                                |
| Intel Neural Compute Stick 2  |                                  |                                           |                                       |                                       |
FIGURE 5. Measured current consumption of benchmarked devices, for the same supply voltage, during inference on the test set. For each graph X axis is limited and shows first 180 seconds of examination for a better view of repeatability and process execution. Note the change of the scale and range between subgraphs on Y axis.
M. Piechocki et al.: Efficient People Counting in Thermal Images

FIGURE 6. Average energy consumption [J] per cycle: (a) 30-second; (b) 5-minute. Note the change in the Y axis range between graphs.

(Counting MRAPE), classification metrics and the number of neural network parameters, is shown in Table 3. The introduced method outperformed other approaches in terms of all metrics, maintaining a notable decrease in size, number of parameters, and model complexity. Moreover, in regard to the person count metrics, the developed encoder-decoder model achieves two times better outcomes than the network proposed in [27] and over ten times exceeds the solution based on fully connected layers as measured by the proposed prediction errors.

To demonstrate the presented architecture’s accuracy of the count predictions, the confusion matrix with percentage results of the estimated occupancy counts is depicted in Figure 4. The vertical axis describes actual values, whereas horizontal dimensions refer to an estimated number of people. The diagonal contains values correctly classified, while the remaining fields refer to misclassifications with an indication between which classes it happened. Because of the classes’ imbalance in the dataset, numerical values were normalized row-wise relative to the number of ground-truth class samples and presented as a percentage score of classification. The aforementioned chart confirms the model’s ability to classify the number of people for each ground-truth class corresponding to the exact person count from the dataset.

The trained model was optimized and quantized using the tools and frameworks described in Section II-C. In most cases, the INT8 data type was the target format of model weights, inputs, and outputs. However, the Intel NCS 2 with Movidius Myriad X doesn’t support integer operations, thus, floating-point types: FP32 and FP16 were utilized. Moreover, for several devices, both integer and floating-point representations were evaluated to highlight the contrast between them in the terms of inference time and metrics value. In the next step, optimized implementations were deployed on hardware and evaluated on the dedicated test set. The above steps were repeated for each device under the same environmental conditions, including supply voltage. Only for time measurements, built-in platform functions were used.

The results of the performance and metrics evaluation for each device are presented in Table 5. Meanwhile, Table 6 shows the power consumption measured for three states:

- inference - a device was receiving input data, predicting and sending output density map through the UART or serial port,
- idle - hardware was active but did not perform any user-defined tasks or instructions,
- efficient mode - microcontrollers were put into sleep mode, whereas Raspberry Pi had optimized power consumption.

During the hardware evaluation phase, specifically the inference step, the current consumption measurements were gathered for all microcontrollers included in the benchmarks at the same supply voltage. The characteristics are shown in Figure 5. The measurement time on the x-axis is limited to 180 seconds to highlight peaks related to the prediction stage. Moreover, it is worth mentioning that the scales and ranges of the y-axis are individual for each subgraph.

According to the last part of Section III-B the efficiency indicators of the operation cycle were calculated. Figure 6 presents bar plots representing the energy required to perform
a single work cycle. The left subdiagram refers to a 30-second operation period, while the other depicts energy consumption for 5-minute single series. However, a better view of the above approaches, in terms of daily energy consumption, is presented in Figure 7. The subframes expose the required energy for one full day of operations as follow: (a) 30-second and (b) 5-minute cycles. As can be seen by comparing the graphs, the energy consumption is slightly lower for the second duration for almost all devices.

V. CONCLUSION

Accurate occupancy estimation is one of the main aspects of HVAC systems’ efficient management in Smart Buildings. The proper configuration of heating, ventilation and fans ensures not only better conditions in either offices or rooms but also allows reducing the cost of installation usage. Moreover, with the utilization of embedded devices and edge processing, the people count can be estimated onboard, reducing the use of the cloud and sending only the final number of persons. In addition, this approach protects people’s privacy in two ways: firstly, by estimating the count using low-resolution thermal images, which characteristic prevents personal data gathering. Secondly, the collected frames are processed onboard and removed from the device memory after inference. Thus, any image is recorded or streamed for further analysis.

This article presents an optimized version of the U-Net convolutional neural network for the density estimation of people’s presence. The main objective of this work was to benchmark algorithm implementation on a set of resource-limited devices. The investigated tests and benchmarks have shown that occupancy counting could be processed onboard on resource-constrained hardware with a limited power supply. The research carried out has proven that even microcontrollers based on low-energy chips, such as nRF52840 or ESP32-WROOM-32, are able to run a relatively fast density estimation algorithm whereby current consumption does not exceed 30 mA and 150 mW. On the other hand, the MCUs having an Arm Cortex-M7 core (Arduino Portenta H7 and STM32 H745ZI Nucleo-144 boards) can achieve close to 20 independent inferences per second, maintaining a current consumption of less than 270 mA and 1350 mW at the peak. Using the aforementioned 30-seconds or 5-minutes work cycles, it is possible to substantially reduce energy consumption by using the devices’ efficient mode, often known as sleep or deep sleep mode. It turns out from the conducted tests that the LOLIN32 microcontroller equipped with ESP32-WROOM-32 chip performs best, obtaining respectively 0.92 and 8.62 J per cycle and 2.64 or 2.48 kJ daily. Moreover, it is nearly three times less than second in the list microcontroller Arduino Nano 33 BLE Sense with nRF52840 unit and approximately 22 times less than the fastest and the most powerful device from benchmarked MCUs - an STM32 H745ZI on Nucleo-144 board. From another point of view, the tested Raspberry Pi with computation accelerators may be a suitable option for more complex measurement systems where the power supply is not as important as the execution time. The measurements show, that inexpensive, consumer-grade microcontrollers are capable of executing deep learning workloads with optimized neural network models.
inference time is not critical in a wide range of IoT or edge applications, such devices are a viable option to consider, which was proven by evaluating a range of microcontrollers and microprocessors using real-life data sourced from a non-trivial application. Moreover, the microcontrollers offer good cost-effectiveness both in terms of hardware price, as well as operating costs, outperforming faster, but more complex computational architectures in the evaluation.

Although the proposed approach offers accurate results and satisfactory performance, it does not circumvent all threats to the correct operation validity. Significant changes in camera placement may affect measurements, resulting in worsening the results of the algorithm at the same time. What is more, the low resolution of the thermal camera, with a relatively large number of people in the sensor field of view, may cause a saturation problem that may strongly affect the system’s capabilities. Another threat is heat sources close to the typical human temperature. Despite the fact that the authors of the dataset used additional heat sources to increase the diversity of the data as well as the robustness of the algorithm, this topic is not fully covered, and there is a possibility of false-positive results occurrence. Hence, the further development aims will include broadening the research of person detection and counting methods, e.g., using RGB sensors or testing algorithms on images containing a higher number of people. Another task concerns system installation as an HVAC perception unit and the study of its impact on the behavior of people. The aforementioned issues will be addressed in further research.

ACKNOWLEDGMENT

The code repository related to this publication is open source and available at https://github.com/PUTvision/thermal-hardware-benchmark.

REFERENCES

[1] K. Karimi and G. Atkinson. What the Internet of Things (IoT) Needs to Become a Reality. Accessed: May 31, 2022. [Online]. Available: https://www.mouser.mx/pdfdocs/INTOTHNGSWP.PDF

[2] E. Al Nuaimi, H. Al Neyadi, N. Mohamed, and J. Al-Jaroodi, “Applications of big data to smart cities,” J. Internet Services Appl., vol. 6, no. 25, pp. 1–15, 2015.

[3] Z. Allam and Z. A. Dhunny, “On big data, artificial intelligence and smart cities,” Cities, vol. 89, pp. 80–91, Jun. 2019.

[4] Q. Chen, W. Wang, F. Wu, S. De, R. Wang, B. Zhang, and X. Huang, “A survey on an emerging area: Deep learning for smart city data,” IEEE Trans. Emerg. Topics Comput. Intell., vol. 3, no. 5, pp. 392–410, Oct. 2019.

[5] G. Lampropoulos, K. Siakas, and T. Anastasiadis, “Internet of Things in the context of industry 4.0: An overview,” Int. J. Entrepreneurial Knowl., vol. 7, no. 1, pp. 4–19, Jul. 2019.

[6] A. C. Pereira and F. Romero, “A review of the meanings and the implications of the industry 4.0 concept,” Proc. Manuf., vol. 13, pp. 1206–1214, Jan. 2017.

[7] P. Cervalli, A. Lundvall, P. Jonsson, S. Carson, R. Möller, R. Saksena, D. Lu, and I. Celik. (2021). Ericsson Mobility Report. [Online]. Available: https://www.ericsson.com/4ad7e9/assets/local/reports-papers/mobility-report/documents/2021/ericsson-mobility-report-november-2021.pdf

[8] S. J. Ramson, S. Vishnu, and M. Shamugam, “Applications of Internet of Things (IoT)—An overview,” in Proc. 5th Int. Conf. Devices, Circuits Syst. (ICDCS), Mar. 2020, pp. 92–95.

[9] X. Ma, T. Yao, M. Hu, Y. Dong, W. Liu, F. Wang, and J. Liu, “A survey on deep learning empowered IoT applications,” IEEE Access, vol. 7, pp. 181721–181732, 2019.

[10] K. Kim, “The smallest engine transforming humanity: The past, present, and future,” in IEDM Tech. Dig., Dec. 2021, p. 1.

[11] M. Satyanarayanan, “The emergence of edge computing,” Computer, vol. 50, no. 1, pp. 30–39, Jan. 2017.

[12] K. Cao, Y. Liu, G. Meng, and Q. Sun, “An overview on edge computing research,” IEEE Access, vol. 8, pp. 85714–85728, 2020.

[13] R. Sanchez-Ibarra and A. F. Skarmeta, “TinyML-enabled frugal smart objects: Challenges and opportunities,” IEEE Circuits Syst. Mag., vol. 20, no. 3, pp. 4–18, 3rd Quart., 2020.

[14] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang, and P. Warden, “Tensorflow lite micro: Embedded machine learning for TinyML systems,” Proc. Mach. Learn. Syst., vol. 3, pp. 800–811, Mar. 2021.

[15] H. Doyu, R. Morabito, and M. Brachmann, “A TinyMLaaS ecosystem for machine learning in IoT: Overview and research challenges,” in Proc. Int. Symp. VLSI Design, Autom. Test (VLSI-DAT), Apr. 2021, pp. 1–5.

[16] M. Shafique, T. Theocharides, V. J. Reddy, and B. Murmann, “TinyML: Current progress, research challenges, and future roadmap,” in Proc. 58th ACM/IEEE Design Automat. Conf. (DAC), Dec. 2021, pp. 1303–1306.

[17] H. Han and J. Siebert, “TinyML: A systematic review and synthesis of existing research,” in Proc. Int. Conf. Artif. Intell. Inf. Commun. (ICAIHC), Feb. 2022, pp. 269–274.

[18] G. Amato, F. Carrara, F. Falchi, C. Gennaro, C. Meghini, and C. Vairo, “Deep learning for indoor parking lot occupancy detection,” Expert Syst. Appl., vol. 72, pp. 327–334, Apr. 2017.

[19] K. Tjonck, C. R. Kanchalar, J. Vankeirsbilck, H. Hallez, J. Boydens, and B. Pang, “Real-time activity tracking using TinyML to support elderly care,” in Proc. 30th Int. Conf. Sci. Conf. Electron. (ET), Sep. 2021, pp. 1–6.

[20] D. L. Dutta and S. Bhardwaj, “TinyML meets IoT: A comprehensive survey,” Internet Things, vol. 16, Dec. 2021, Art. no. 100461.

[21] C. R. Banbury, V. J. Reddi, M. Lam, W. Fu, A. Fazel, J. Holleman, X. Huang, R. Hurtado, D. Kanter, A. Lokhmotov, D. Patterson, D. Pau, J.-S. O. I. Sieracki, U. Thakker, M. Verhelst, and P. Yadav, “Benchmarking TinyML systems: Challenges and direction,” 2020, arXiv:2003.04821.

[22] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional networks for biomedical image segmentation,” in Medical Image Computing and Computer-Assisted Intervention—MICCAI 2015 (Lecture Notes in Computer Science), vol. 9351, N. Navab, J. Hornegger, W. Wells, and A. Frangi, Eds. Cham, Germany: Springer, 2015, doi: 10.1007/978-3-319-24574-4_28.

[23] Y. P. Raykov, E. Ozer, G. Dasika, A. Boukouvalas, and M. A. Little, “Predicting room occupancy with a single passive infrared (PIR) sensor through behavior extraction,” in Proc. ACM Int. Joint Conf. on Pervasive Ubiquitous Comput., New York, NY, USA, Sep. 2016, pp. 1016–1027.

[24] C. Lecch, Y. P. Raykov, E. Ozer, and G. V. Merrett, “Real-time room occupancy estimation with Bayesian machine learning using a single PIR sensor and microcontroller,” in Proc. IEEE Sensors Appl. Symp. (SAS), May 2017, pp. 1–6.

[25] M. Abedi and F. Jazizadeh, “Deep-learning for occupancy detection using Doppler radar and infrared thermal array sensors,” in Proc. Int. Symp. Autom. Robot. Construction (IAARC), May 2019, pp. 1098–1105.

[26] A. Metwally, J. G. Queralta, V. K. Sarker, T. N. Gia, O. Nasir, and T. Westerlund, “Edge computing with embedded AI: Thermal image analysis for occupancy estimation in intelligent buildings,” in Proc. Intell. Embedded Syst. Architectures Appl. Workshop, Oct. 2019, pp. 1–6.

[27] M. Kraft, P. Aszkszki, D. Pecziyski, and M. Fularz, “Low-cost thermal camera-based counting occupancy meter facilitating energy saving in smart buildings,” Energies, vol. 14, no. 15, p. 4542, Jul. 2021.

[28] N. Ilyas, A. Shahzad, and K. Kim, “Convolutional-neural network-based image crowd counting: Review, categorization, analysis, and performance evaluation,” Sensors, vol. 20, no. 1, p. 43, Dec. 2020.

[29] B. Sudharsan, S. Salerno, D.-D. Nguyen, M. Yahya, A. Wahid, P. Yadav, J. G. Breslin, and M. I. Ali, “TinyML benchmark: Executing fully connected neural networks on commodity microcontrollers,” in Proc. IEEE 7th World Forum Internet Things (WF-IoT), Jun. 2021, pp. 883–884.

[30] (2019). Melexis. MLX90640 32×24 IR Array. [Online]. Available: https://www.mouser.com/mexmlfsheet-mlx90640-datasheet-muxlelis.pdf

[31] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in Proc. 3rd Int. Conf. Learn. Represent. (ICLR), San Diego, CA, USA, May 2015, pp. 1–15.
M. Piechocki et al.: Efficient People Counting in Thermal Images

[32] Google. Tensorflow Lite. Accessed: Apr. 30, 2022. [Online]. Available: https://www.tensorflow.org/lite

[33] Intel. Openvino. Accessed: Apr. 30, 2022. [Online]. Available: https://docs.openvino.ai/2021.4/index.html

[34] Google. Tensorflow Lite for Microcontrollers. Accessed: Apr. 30, 2022. [Online]. Available: https://www.tensorflow.org/lite/microcontrollers

[35] STMicroelectronics. X-Cube-AI. Accessed: Apr. 30, 2022. [Online]. Available: https://www.st.com/en/embedded-software/x-cube-ai.html

[36] R. Lauer. (2021). Optimizing Raspberry PI Power Consumption. Accessed: Apr. 30, 2022. [Online]. Available: https://blues.io/blog/tips-tricks-optimizing-raspberry-pi-power

MATEUSZ PIECHOCKI received the master’s degree (Hons.) in the automatic control and robotics field, in July 2021. He is currently pursuing the Ph.D. degree in the automation, electronic and electrical engineering discipline with the Poznan University of Technology (PUT).

He has been with PUT, since 2021, where he is also a Computer Vision Engineer with the Machine Intelligence Vision Laboratory and the Institute of Robotics. His research interests include intelligent building systems optimization using image processing algorithms, quantization techniques, resource-constrained IoT devices, and new technologies, especially deep learning, computer vision, and edge processing.

MAREK KRAFT (Member, IEEE) received the Ph.D. degree in the field of automation and robotics, in 2013.

He is currently an Assistant Professor and a Researcher at the Faculty of Control, Robotics and Electrical Engineering, Poznan University of Technology. He defended his doctoral dissertation, in 2013. For over ten years, he has been working in the field of image processing, machine learning, robotics, and in engineering practice, he has also been involved in designing embedded and measurement systems. He is the author of over 40 scientific publications and a member of the international organizations, such as ACM. His area of competence covers the processing of images and video streams, with particular emphasis on machine learning methods: recognition, detection, tracking of objects in images and video, binary and semantic segmentation, and medical image processing. In addition, his proficiency includes implementation of the above-mentioned methods in embedded architectures and their practical use in vision-based navigation, medical diagnostics, autonomous vehicles, safety, and monitoring applications.

TOMASZ PAICHROWSKI received the Ph.D. and D.Sc. degrees in control of electrical drives from the Poznan University of Technology, Poznań, Poland, in 2005 and 2016, respectively.

He is currently an Assistant Professor with the Faculty of Control, Robotics and Electrical Engineering, PUT, and the Deputy Director of the Institute of Robotics and Machine Intelligence. He is the author or coauthor of over 120 scientific papers. His research interests include control of synchronous permanent-magnet motors, especially control for multi-mass system with variable mechanical parameters, where control systems, the nonlinear, adaptive, robust control algorithms, computational intelligence methods are applied, and energy-efficient control of electric vehicles and buildings using machine learning.

PRZEMYSLAW ASZKOWSKI received the master’s degree in automation and robotics from the Poznan University of Technology, in 2018, where he is currently pursuing the Ph.D. degree in the automation, electronic and electrical engineering discipline. He has more than four years of professional experience with commercial research and development projects, including digital signal processing and laboratory equipment control. Since 2021, he has been a part of the research team at the Machine Intelligence Vision Laboratory and the Institute of Robotics. His research interests include computer vision with machine learning, with an emphasis on multispectral imaging in environmental and agricultural applications.

DOMINIK PIECZYNSKI received the master’s degree in information engineering, in 2018.

He works as a Research Assistant at the Faculty of Control, Robotics and Electrical Engineering, Poznan University of Technology. His research interest includes applying deep neural networks in various computer vision tasks. He worked with person re-identification, face identification, medical, and agricultural segmentation. Currently, he is focusing on using synthetically generated data to improve neural network’s learning capacity when the labeled real-world data is scarce.