A Comparative Study of different Edge-AI algorithms

Juuso Kapulainen[1]

Abstract— **Edge Artificial Intelligence (Edge-AI) and tiny Machine Learning (tinyML) play a crucial role in resource-constrained environments, enabling smart and efficient solutions. This study compares the performance of Sparse Autoencoders, Self-Organizing Maps (SOM), and Undercomplete Autoencoders within the constraints of edge devices, specifically focusing on incremental learning, normal training data only, and Arduino Nano's memory limitations. The algorithms were evaluated using metrics like accuracy, precision, recall, F1 score, AUROC, and model memory size as well as the training time of the model. The SOM algorithm emerged as the superior model, outperforming the others in almost every metric while adhering to the given constraints. In conclusion, the SOM algorithm demonstrated its potential as a viable Edge-AI solution for future projects, offering superior performance under the constraints of resource-limited environments. This research emphasizes the importance of choosing appropriate algorithms for Edge-AI applications and offers insights for developing efficient tinyML solutions.**

I. Introduction

The field of Artificial Intelligence has made significant advancements in recent years and new paradigms have emerged to challenge the traditional models of machine learning. ”TinyML” and ”Edge-AI” are concepts that refer to the application of machine learning algorithms on small, low-power devices, such as sensors, IoT devices, and embedded systems. These algorithms are designed to run locally on the device, rather than in the cloud or on a centralized server which have been the traditional platforms for Artificial Intelligence solutions.

Compared to cloud computing, edge computing has the following advantages: (1) computation is performed locally or at the edge servers, lowering latency; (2) data does not need to be uploaded to the cloud, reducing bandwidth consumption; (3) data is stored at the edge, which protects user privacy[1]. Specifically, TinyML focuses on deploying machine learning and deep neural network models to the embedded, resource-constrained devices powered by microcontrollers (MCUs), thereby offering solutions for applications with low latency and low communication bandwidth constraints[2]. Traditional TinyML tools assume that the model is trained in powerful machines or the cloud, and it is afterward uploaded to the edge device. The resource-constrained edge device, such as MCU, only needs to perform inference. [3]

This research paper presents a comparative study of machine learning algorithms for Edge-AI environments. Three specific boundary conditions have been considered while comparing algorithms: the absence of anomalies in the training phase, incremental learning in training phase of the model, and the limited memory capacity of the devices training and running the model. The results of this study provide valuable insights into the most suitable algorithms for resource-constrained Edge-AI environments such as Arduino Uno or Nano. In the implementation of this research, three machine learning algorithms have been selected for comparison. These algorithms have been implemented with the boundary conditions taken into account, ensuring that the results are accurate and relevant. The vibration dataset is used for the implementation, with the training part used for training and the testing part used for testing the model. The data gathered from the implemented models is then evaluated against each other to compare their performance. The implementation process is designed to provide a thorough evaluation of the algorithms, enabling a detailed comparison of their performance and suitability for Edge-AI environments.

The research paper is structured as follows: In Chapter 2, ”TinyML and Edge-AI”, a brief introduction to to TinyML/Edge-AI is provided, thus providing a background for the study. Chapter 3, ”Algorithms”, presents the selected machine learning algorithms. In Chapter 4, ”Experimental Setup”, the details of how the models were implemented and evaluated are discussed in significant detail. Chapter 5, ”Analysis”, provides an analysis of the data gathered, including a comparison of the algorithms based on the results. Finally, Chapter 6, ”Conclusion”, summarizes the key findings and provides insights into the most suitable algorithms for Edge-AI environments.

II. TinyML and Edge-AI

Traditionally, AI was mainly implemented on cloud computing platforms, where large amounts of computational power and storage were required to process the vast amounts of data generated by IoT devices. However, this approach has several drawbacks, such as huge energy consumption, privacy issues, network and processing latency, and reliability issues [9]. To address these limitations, edge computing has emerged as a new possible platform for AI.

TinyML and Edge-AI are two related concepts that aim to bring computation closer to the data source, enabling faster processing and decision-making. TinyML refers to the deployment of machine learning models to extremely resource-constrained devices powered by microcontrollers, such as those used in IoT devices [16]. This technology enables machine learning models to run on devices with limited computational power, storage capacity, and energy

*This work was not supported by any organization

[1]J. Kapulainen is with faculty of Information Technology, Department of Mathematical Information Technology, University of Jyväskylä a,P.O.Box 35 (Agora), FIN-40014 University of Jyväskylä, Finland `juuso.kapulainen at student.jyu.fi`

consumption. Edge-AI, on the other hand, refers to the deployment of AI algorithms at the edge of the network, near the data source [9]. The goal of Edge-AI is to provide real-time data processing and decision-making capabilities, without the need for a connection to the cloud. Edge-AI has several advantages over cloud computing, including lower latency, reduced bandwidth consumption, and increased privacy. The inherent features like versatility, cost-effectiveness, and simplicity of TinyML framework warrant attention for their potential to transform the whole IoT ecosystem [11].

In addition to its advantages, TinyML and Edge-AI also present several challenges. One of the main challenges is the need to develop machine learning algorithms that can run efficiently on resource-constrained devices. This requires the development of algorithms that can operate with limited computational power, memory capacity, and energy consumption. Ordinarily, the TinyML system is flashed with binary files which are generated from the trained model on a more computationally powerful host machine [9]. However, the need to also perform the learning at the edge has the caused the need to develop algorithms that can perform incremental on-device learning, which cannot take the entire data set as input.

A. Incremental learning

Whereas in "traditional" machine learning, we're given a complete dataset to be used as a training data, in incremental learning, we don't have all of the data available when creating the model. Instead, the data points arrive one at a time and the model has to learn and adapt as the data comes. In such settings, it is necessary to update an existing classifier in an incremental fashion to accommodate new data without compromising classification performance on old data. [17] In such cases learning new information without forgetting previously acquired knowledge the model implements incrmental learning process. Online incremental learning is a subarea of incremental learning that are additionally bounded by run-time and capability of lifelong learning with limited data compared to offline learning. These constraints are very much related to real life applications where new data comes in sequentially and is in conflict with the traditional approach, where usually complete data is available. Here each new data is used only once to update the model instead of training the model using the new data with multiple epochs as in offline mode. [2] This let us perform a training process on a batch of data and requires only a single batch of data into main memory at the time. This approach does not require all the data to be in the main memory instead only a small portion of it. Using incremental learning techniques, machine learning models can adapt to constantly arriving data streams, such as the vibration data that is used in this study.

B. Related works

The development of TinyML has been a rapidly growing research area in recent years, with the aim of making machine learning more available for microcontrollers in the Internet of Things. The challenge of implementing machine learning on devices with limited power, memory, and computation resources has inspired numerous studies to explore many different solutions. As a result, the area of TinyML has seen remarkable progress, with numerous studies contributing to its advancement. Despite these efforts, the issue of on-device incremental training remains an active area of research, and there is a need for further exploration of the topic so we are able to develop more effective solutions.

To address the current challenges of "offline learning" TinyML/EdgeAI, Ren, Anicic and Runkler[3] proposed a novel system called TinyOL (TinyML with Online-Learning) that enables incremental on-device training on streaming data. The authors conducted experiments under supervised and unsupervised setups using an autoencoder neural network and demonstrated the effectiveness and feasibility of TinyOL. They concluded that TinyOL, a model trained online, performs well for tiny devices despite limitations in data and computational power, compared to models trained offline with more resources.

Antonini et al. [12] introduce an adaptable and unsupervised anomaly detection system for extreme industrial environments like submersible pumps, utilizing IoT, edge computing, and Tiny-MLOps methodologies. The system employs an IoT sensing kit with an ESP32 microcontroller and MicroPython firmware to run a processing pipeline that collects data, trains an anomaly detection model incrementally straight from sensor data, and alerts an external gateway in case of anomalies. The isolation forest algorithm is used for anomaly detection, and the model can be trained on the microcontroller in relatively short time period.

III. ALGROTIHMS

In this study, we have chosen Sparse auto-encoder, Undercomplete auto-encoder, and Self-organizing Maps as our main algorithms for investigation, given their varied approaches to tackling the problem at hand. Additionally, the inclusion of two distinct autoencoder models allows for a comparative analysis to discern the differences and potential advantages between these specific variants of autoencoder architectures.

A. Sparse autoencoder

The Sparse Autoencoder (SAE) is a neural network architecture that learns to encode input data into a lower-dimensional space while minimizing the reconstruction error. Compared to other autoencoders, sparse autoencoders offer us an alternative method for introducing an information bottleneck without requiring a reduction in the number of nodes at our hidden layers. This is implemented by adding a sparsity constraint on the hidden units, and thus the autoencoder will still discover interesting structure in the data, even if the number of hidden units is large. [19] To implement sparsity, sparse autoencoders randomly limit the number of active nodes during training to create the bottleneck in the information flow, instead of reducing the number of nodes in the hidden layers [10]. A sparse autoencoder is simply an autoencoder whose training criterion involves a sparsity penalty on the code layer in addition to the reconstruction error. An autoencoder that has been regularized to be sparse must respond to unique statistical features of the dataset it has been trained on, rather than simply acting as an identity function. [5]

B. Undercomplete autoencoder

The Undercomplete Autoencoder (UAE) is another type of neural network architecture. It is an autoencoder whose code dimension is less than the input dimension. Learning an undercomplete representation forces the autoencoder to capture the most important features of the training data. [5] Similar to SAEs, UAEs are designed to encode input data into a lower-dimensional space while minimizing the reconstruction error. However, unlike SAEs, UAEs do not incorporate sparsity constraints, instead relying on the network architecture itself to learn an efficient representation of the input data. Undercomplete autoencoders constraint the number of nodes in the hidden layers to limit the amount of information that flows through the network. In this way, during training, the network will learn to reconstruct the essential features from the original input from the compressed representation. Undercomplete autoencoders use a "bottleneck" in their architecture to restrict the flow of information [10]. UAEs have been successfully used in a variety of applications, including anomaly detection, feature learning, and image and speech recognition, and offer a useful alternative to SAEs.

UAEs are particularly useful for applications where the input data has a high degree of correlation or redundancy, as they are able to effectively capture this information and represent it in a compressed form. However, because UAEs do not enforce sparsity, they may require a larger number of hidden units compared to SAEs to achieve similar levels of compression.

C. Self-Organizing Map

The Self-Organizing Map (SOM) algorithm is is a type of neural network that organizes high-dimensional data into a two-dimensional grid, maintaining the relative distances between data points. This makes it useful for clustering and analyzing complex datasets [18]. Unlike SAEs and UAEs, SOMs do not learn an explicit encoding of the input data, but rather map the input data onto a lower-dimensional grid of nodes. SOMs are particularly useful for applications where the input data has a high degree of complexity and structure. SOMs have been successfully used in a variety of applications, including data visualization, clustering, and anomaly detection, and offer a powerful toolset for exploring and understanding complex data in machine learning applications.

The SOM algorithm distinguishes two stages: the competitive stage and the cooperative stage. In the first stage, the best matching neuron is selected, i.e., the "winner", and in the second stage, the weights of the winner are adapted as well as those of its immediate lattice neighbors. [7] SOMs use competitive learning during which it uses lateral interactions among the neurons to form a semantic map where similar patterns are mapped closer together than dissimilar ones. A SOM is a single layer neural network with units set along an n-dimensional grid. Topologically SOMs most often use a two-dimensional grid, although one-dimensional, higher-dimensional and irregular grids are also possible. [6]

IV. Experimental Setup

A. Platform

To compare anomaly detection algorithms, this research addresses three specific constraints: the absence of anomalies during the training phase, incremental learning in the model's training phase, and the limited memory capacity of the devices used for training and deploying the model.

The memory capacity of the devices employed for training and running the model is carefully considered. The learned normal behavior model is designed to be compact, occupying no more than 80 percent of the memory of an Arduino Nano, which is 32KB. The memory footprint of the model is monitored using built-in tools of TensorFlow Lite, which the model is built upon. As necessary, algorithms and models were adjusted to adhere to the imposed memory constraints.

In this study, we employ a diverse array of technologies and libraries to develop and evaluate three distinct incremental learning models tailored for EdgeAI/TinyML applications. Python serves as the primary programming language, with Anaconda as the package and environment management system, streamlining the installation and management of dependencies. For model construction and training, we utilize Keras and TensorFlow, both of which are renowned for their flexibility and efficiency in deep learning tasks. The data preprocessing phase leverages the Pandas library for data manipulation and the NumPy library for numerical computations, ensuring optimal handling of the datasets. Scikit-learn plays a pivotal role in data scaling and additional preprocessing steps, contributing to the overall robustness of the models. Furthermore, we adopt the MiniSom library to implement the Self-Organizing Map (SOM) model, a key component of our experimentation. To facilitate deployment on resource-constrained devices, we employ TFLite for model conversion, yielding optimized, compact models tailored for peripheral devices. Collectively, these technologies and methodologies enable the development and implementation of versatile, efficient, and reproducible learning models that cater to the specific requirements of our research experiments.

B. Dataset

The dataset used in this study, titled "Vibration data collected with the simple fan + battery environment (raw accelerometer samples)," is sourced from a GitHub repository [8]. The dataset was provided by the supervising professor. The dataset comprises raw accelerometer samples that capture the normal and anomalous behavior of the testbed.

The dataset is divided into two subsets: "train.csv" and "test.csv". The training set consists of 120,000 entries, while the testing set contains 110,000 entries. Each entry in the dataset is labeled as either normal or anomalous. Importantly, the training dataset exclusively contains normal data, ensuring that the models are only exposed to normal behavior during the training phase. In contrast, the testing set incorporates 30,000 anomalous data points, enabling the evaluation of the models' performance in detecting anomalies during the inference phase. By maintaining a clear separation between normal and anomalous data, the experimental setup ensures that the models are challenged to generalize effectively when detecting anomalous behavior in real-world applications.

To accommodate the incremental learning requirement during the model's training phase, the dataset is divided into smaller batches, and these batches are sequentially fed into the model. This approach facilitates continuous training of the model and closely simulates real-world use cases for these models.

C. Evaluation metrics

In this study, several key performance metrics are employed to evaluate the effectiveness of the anomaly detection models. These metrics facilitate a comprehensive assessment of the models' performance and allow for comparisons between different approaches. The primary metrics used in this study are accuracy, precision, recall, F1 score, and area under the receiver operating characteristic curve (AUROC).

Accuracy measures the proportion of correct predictions, both normal and anomalous, out of the total number of predictions made. Precision represents the ratio of true positive predictions to the sum of true positive and false positive predictions, reflecting the model's ability to correctly identify anomalies while minimizing false alarms. Recall is the ratio of true positive predictions to the sum of true positive and false negative predictions, reflecting the ability to detect the majority of actual anomalies. The F1 -score means the harmonic mean between precision and recall. [13] Finally, the AUROC is a metric that evaluates the overall performance of the model across various decision thresholds. It quantifies the trade-off between the true positive rate and the false positive rate over a range of threshold values. Its value will always be between 0 and 1.0 with higher value indicating a superior ability to distinguish between normal and anomalous instances. [15]

In addition to these more tradiotinal evalutation metrics, this study will consider the memory size and training time of the produced models. Funtion of these is to make sure that the model will fit the set memory boundaries as well as to include information about the training time of each model. Together, these metrics offer a comprehensive evaluation of the anomaly detection models, enabling a thorough understanding of their strengths and weaknesses in detecting anomalous behavior.

Regarding the accuracy evaluation metric, it might fail to keep the integrity of the reliability of the outcome for unbalanced data modeling. This problem is known as Accuracy Paradox. Thus, in cases where the cost of misclassifying the minority class is high, as in anomaly detection, relying solely on accuracy is not recommended. As such, int his stu,y precision, recall, F1 score and AUROC are considered as well when evaluating the model's performance. These metrics provide a better indication of the model's ability to correctly identify the minority class, ensuring that the model can effectively detect anomalies. [13]

D. Model tuning

In this study, extensive hyperparameter tuning was conducted to optimize the performance of the implemented models. For the autoencoders, various optimizers, loss functions, learning rates, sparsity constraints, and activation functions, such as ReLU, LeakyReLU, and ELU, were explored. Different neuron sizes and layer configurations were also experimented with, adhering to the given memory constraint. Although regularization techniques were investigated, no significant improvement was observed in the model performance.

For the SOM model, several parameters, including size, sigma, learning rate, and seed, were fine-tuned to achieve the best performance. A size of 21x21 was determined to be the largest that could fit the memory constraint for the current implementation. Moreover, different batch sizes were examined for both autoencoders and SOM. While autoencoders did not perform well with a batch size of one, SOM demonstrated remarkable performance under this condition. With autoencoders, a batch size of ten was chosen, as it still facilitates incremental learning relevant to the task at hand, while smaller batch sizes were found to negatively impact model performance.

The final hyperparameters used for the models in this study are as follows: For autoencoders, the optimizer was set to Adam with a learning rate of 1e-5 (UAE) and 1e-4 (SAE), and the activation function was LeakyReLU. We explored various threshold values for anomaly detection and determined that a threshold of 75 provided a suitable balance between sensitivity and specificity in identifying anomalies. The SOM model utilized a size of 21x21, sigma of 0.5, learning rate of 0.4, and a random seed of 123. These optimized parameters ensured the models' effectiveness and adaptability for the given problem.

V. Analysis

The results of the study demonstrate the performance of the three anomaly detection algorithms in terms of accuracy, precision, recall, F1 score, AUROC, and model size. The evaluation results for all the models in this study are presented in Table I. The highest value for each evaluation metric is emphasized in bold, showcasing the superior performance of the respective model for that particular metric. The Sparse Autoencoder achieved an accuracy of 0.86, precision of 0.77, recall of 0.71, F1 score of 0.74, AUROC of 0.88, and a model size of 13.24 KB. The Self-Organizing Map (SOM) method exhibited an accuracy of 0.91, precision of 0.81, recall of 0.89, F1 score of 0.85, AUROC of 0.96, and a model size of 24.19 KB. Finally, the Unsupervised Autoencoder (UAE) attained an accuracy of 0.83, precision of 0.71, recall of 0.65, F1 score of 0.68, AUROC of 0.86, and the smallest model size of 8.57 KB. The ROC curves of each algorithm can be seen in figures 1, 2 and 3. An ROC graph depicts relative tradeoffs between benefits (true positives) and costs (false positives) [15].

TABLE I
PERFORMANCE METRICS FOR THREE ALGORITHMS

	Accuracy	Precision	Recall	F1 Score	AUROC	Size
SAE	0.86	0.77	0.71	0.74	0.88	13.24KB
SOM	**0.91**	**0.81**	**0.89**	**0.85**	**0.96**	24.19KB
UAE	0.83	0.71	0.65	0.68	0.86	**8.57KB**

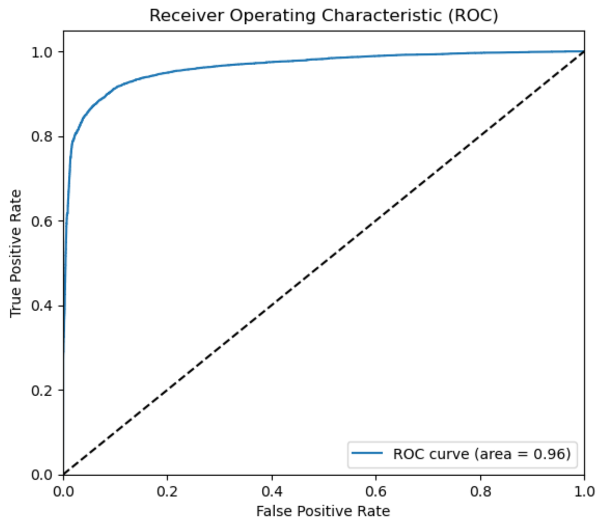

Fig. 1. ROC curve of SOM algorithm

In terms of training time, the Self-Organizing Map (SOM) model took the longest, with a duration of 101.03 seconds. Meanwhile, both the Sparse Autoencoder (SAE) and Unsupervised Autoencoder (UAE) models demonstrated considerably shorter training times with at 62.27 seconds and 61.66 seconds. However, it is important to consider that the batch size used for the SAE and UAE models was 10, in contrast to the SOM model, which used a batch size of 1. The difference in batch size may have significantly impacted the training times, potentially giving the SAE and UAE models an advantage in this particular comparison. Larger batch size was used with autoencoders as they did not perform well with a batch size of one. When the Self-Organizing Map (SOM) was implemented with a batch size of 10, its training time decreased significantly to just 10.27 seconds. However, this change resulted in a minor reduction in the model's performance. Therefore, it was opted to retain the batch size of one for the SOM, as it provided exceptional performance results while maintaining an acceptable training time. Furthermore, with a batch size of one the model learns effectively by processing one data instance at a time, which can be advantageous in applications such as this as the model only needs to have one entry in the memory at the time.

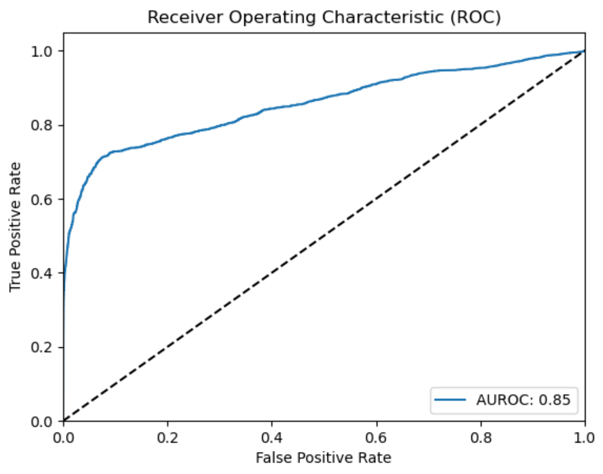

Fig. 2. ROC curve of SAE algorithm

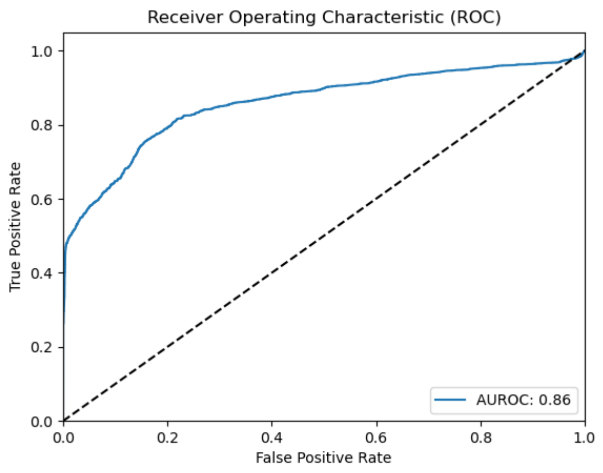

Fig. 3. ROC curve of UAE algorithm

These results highlight the varying strengths and weaknesses of each algorithm. The SOM method achieved the highest accuracy, precision, recall, F1 score, and AUROC, indicating its superior performance in detecting and differentiating between normal and anomalous data points. However, it also had the largest model size. The UAE had the smallest model size, but its performance in terms of accuracy, recall, and AUROC was lower than that of the SOM. The Sparse Autoencoder demonstrated a balanced performance across all metrics. Nonetheless, the size disparity among these models is not a significant concern, as all of them remained below the threshold of 25.1 KB, which corresponds to 80 percent of the memory capacity of an Arduino Nano. Both autoencoder variants showed relatively similar performance, suggesting that their capabilities are fairly comparable when operating under the imposed present in this study.

Upon analyzing the results, it becomes clear that the performance of the Self-Organizing Map (SOM) model is notably superior to the other models. Superior accuracy, precision, recall, F1 score, and AUROC values produced by the SOM model demonstrate its ability to effectively identify anomalies in the data. These performance metrics indicate that the SOM model could be considered most suited for deployment in actual real-world environments among the algorithms studied. Considering the critical nature of anomaly detection in real-world applications, the performance of the SOM model emerges as the key finding of this study, underscoring its viability as a solution for effective anomaly detection.

VI. Conclusion

This study aimed to evaluate and compare the performance of three anomaly detection algorithms (Sparse Autoencoder, Self-Organizing Maps and Undercomplete Autoencoder) with specific boundary conditions, which were; need for incremental learning, training only on normal data, and memory constraints of Arduino Nano devices.

All the algorithms were successfully implemented within the given boundary conditions. However, the Self-Organizing Maps algorithm emerged as the best model by a significant margin, outperforming the other two autoencoders in terms of accuracy, precision, recall, F1-score, and AUROC. The model based on Self-Organizing Maps algorithm did have considerably larger size than the autoencoder based models. Size of the SOM model is not an issue as it still fits the given memory constraint, but this should be noted on future studies. Interestingly, the autoencoders provided quite similar results, indicating their comparable capabilities within the given constraints.

Despite the promising results, this study has some possible shortcomings. Firstly, the evaluation metrics employed in the study provide useful insights into model performance but may not fully capture the nuances of each model's capabilities in a production environment. Moreover, while the study compared three selected algorithms, a more comprehensive comparison with additional algorithms could have provided a broader view of learning techniques, potentially revealing further insights. However, due to the limited timeline and scope of this research, such an extensive analysis was not feasible in the current study. Lastly, the hardware constraints were primarily focused on memory capacity, but other important factors, such as energy consumption, were not thoroughly addressed, which could impact the feasibility of deploying these models in actual resource-constrained devices.

The significance of these findings is that the SOM algorithm, with its superior performance, can be considered as a potentially usable algorithm for real-time anomaly detection in future applications. This study demonstrates the feasibility of implementing effective and efficient machine learning models in resource-constrained environments, opening up new opportunities for leveraging these techniques in various use cases. Further research and development efforts can be directed towards refining the SOM algorithm and exploring its potential in diverse settings.

References

[1] Su, Weixing, Linfeng Li, Fang Liu, Maowei He, and Xiaodan Liang. 2022. “AI on the Ed-ge: A Comprehensive Review.” Artificial Intelligence Review 55 (8): 6125–83. https://doi.org/10.1007/s10462-022-10141-4.

[2] Soro, Stanislava. 2020. “TinyML for Ubiquitous Edge AI.” MITRE TECHNICAL RE-PORT. Mitre.

[3] Ren Haoyu, Darko Anicic, and Thomas Runkler. 2021. “TinyOL: TinyML with Online-Learning on Microcontrollers.” arXiv. http://arxiv.org/abs/2103.08295.

[4] Jiangpeng He, Runyu Mao, Zeman Shao and Fengqing Zhu. 2020 ”Incremental Learning In Online Scenario.” School of Electrical and Computer Engineering, Purdue University

[5] Ian Goodfellow, Yoshua Bengio and Aaron Courville. 2016. ”Deep Learning”. MIT Press.

[6] Dubravko Miljković. 2017. ”Brief Review of Self-Organizing Maps.”. Hrvatska elektroprivreda.

[7] Mark van Hulle. 2012. ”Self-organizing Maps.” Handbook of natural computing

[8] https://github.com/mizolotu/tinyml_experiments/tree/main/data/fan

[9] Parthe Ray. 2022. ”A review on TinyML: State-of-the-art and prospects”. Journal of King Saud University - Computer and Information Sciences

[10] Juan Manuel Davila Delgado, Lukumon Oyedele. 2021. ”Deep learning with small datasets: using autoencoders to address limited datasets in construction management”. Applied Soft Computing.

[11] Lachit Dutta, Swapna Bharali. 2021. ”TinyML Meets IoT: A Comprehensive Survey”. Internet of Things.

[12] Mattia Antonini, Miguel Pincheira, Massimo Vecchio, Fabio Antonelli. 2023. ”An Adaptable and Unsupervised TinyML Anomaly Detection System for Extreme Industrial Environments.”

[13] Dalianis Hercules . 2018. ”Evaluation Metrics and Evaluation”.

[14] Muhammad Fahim Uddin. 2019. ”Addressing Accuracy Paradox Using Enhanched Weighted Performance Metric in Machine Learning”

[15] Fawcett Tom. 2006. ”Introduction to ROC analysis.”

[16] Hui Han, Julien Siebert. 2022. ”TinyML: A Systematic Review and Synthesis of Existing Research”.

[17] Robi Polikar, Lalita Udpa, Satish Udpa and Vasant Honavar. 2001. ”Learn++: An Incremental Learning Algorithm for Supervised Neural Networks”

[18] Jaakko Hollmen. 1996. ”Self-Organizing Map (SOM)”. https://users.ics.aalto.fi/jhollmen/dippa/node9.html

[19] Andrew Ng. 2011. ”CS294A Lecture notes”