Investigating the Suitability of Time Series Classification Algorithms for Embedded Systems: A Case Study on Bicycle Pedaling Detection

Michael Oberle
Competence Centre for Digital Tools in Manufacturing
Fraunhofer IPA
Stuttgart, Germany
michael.oberle@ipa.fraunhofer.de

Sascha Gärtner
Competence Centre for Digital Tools in Manufacturing
Fraunhofer IPA
Stuttgart, Germany
sascha.gaertner@ipa.fraunhofer.de

Florian Maier
Software Defined Energy-Cell Manufacturing
Fraunhofer IPA
Stuttgart, Germany
florian.maier@ipa.fraunhfer.de

***Abstract*—In this paper, we investigate the performance of state-of-the-art time series classification algorithms for pedaling detection in bicycles, focusing on embedded device implementation. Using accelerometer data from a crank-mounted sensor, we benchmark various algorithms, including Rocket, MiniRocket, CNN, LSTM, and HIVECOTEV2. The Rocket algorithm achieves the highest accuracy, followed by LSTM and CNN. However, considering the memory and complexity constraints of embedded devices, the CNN model emerges as the most suitable option. Surprisingly, MiniRocket underperforms in classifying backward pedaling as a non-pedaling state, warranting further investigation. Our findings contribute valuable insights into the applicability of time series classification algorithms for pedaling detection, paving the way for advancements in user assistance systems for e-bikes and mountain bikes.**

***Index Terms*—time series classification, embedded system, pedaling detection, tensorflow lite, cnn, neural networks**

I. Introduction

Modern vehicles are equipped with sophisticated user assistance systems that significantly enhance the user experience on multiple levels. Examples of such systems include adaptive cruise control, lane-keeping assistance, and collision avoidance systems. These advanced capabilities are made possible through embedded systems that process data gathered from a wide array of sensors [1]. In recent years, artificial intelligence (AI) and machine learning (ML) have emerged as state-of-the-art techniques for interpreting and analyzing sensor data. Time series classification, in particular, has played a crucial role in this area. Time series classification is a process that involves identifying patterns or categories within time-ordered sequences of data points. This technique has proven to be highly effective in a variety of application domains [2]. Numerous approaches have been developed for time series classification, yielding impressive results across many use cases. The primary objective of this paper is to evaluate the performance of state-of-the-art time series classification algorithms in the context of a specific use case related to modern bicycles.

The remainder of the paper is structured as follows: related work in the field of time series classification is discussed, the bicycle use case under investigation is detailed, the experimental setup employed for evaluating the algorithms is described, and the obtained results are presented. Following this, the results are discussed, and the paper concludes with a summary of the main findings and their implications.

II. Related Work

A. State of the art for time series classification

Various network architectures, training methods, challenges, and opportunities in applying deep learning to time series data have been discussed [3]. The main focus lies on human activity recognition and satellite earth observation applications. Fawaz et al. [4] present a unified taxonomy of deep neural networks (DNNs) for time series classification (TSC) and an open-source deep learning framework for the TSC community. The study, which involves training 8730 deep learning models on 97 time series datasets, is the most comprehensive analysis of DNNs for TSC to date. The currently top-performing algorithm is HIVECOTE2, an ensemble algorithm. It outperforms current state-of-the-art algorithms on 112 univariate UCR archive datasets and 26 multivariate UEA archive datasets [5]. The ensemble also includes the Rocket algorithm, which uses a linear classifier with random convolutional kernels. It can achieve state-of-the-art accuracy with significantly reduced computational expense compared to HIVECOTE2, enabling rapid training and testing on large datasets. The Rocket algorithm allows training and testing on all 85 'bake off' datasets in the UCR archive in under 2 hours and can train a classifier on a large dataset of more than one million time series in approximately 1 hour [6]. MiniRocket, a reformulated version of Rocket, offers up to 75 times faster performance on larger datasets while maintaining almost the same accuracy. This method allows for state-of-the-art accuracy on all 109 datasets from the UCR archive in under 10 minutes, making it significantly faster than comparable methods and more accurate than methods with similar computational expense

[7]. Petelin et al. [8] focuses on the impact of time-series features on the performance of machine learning (ML) models used for predicting the performance of time-series forecasting algorithms. The study utilizes the tsfresh and catch22 libraries to extract time-series meta-features and employs ML regression models to predict the performance of 61 time-series forecasting algorithms. Although this study provides important advances for the algorithm selection process, it does not consider the constraints of embedded devices in terms of memory, computing power, and power consumption constraints. In addition, it does not consider feature extractions that are not humanly interpretable such as those performed by Rocket and MiniRocket. In summary, the performance of time series classification algorithms has been studied on general benchmarking datasets, and even automatic selection procedures have been researched. However, these results cannot be directly transferred to applications on low-powered embedded devices.

B. Time Series Classification on Embedded Devices

A few studies are reporting on time series classification on embedded devices. Zu et al. [9] use dilated LSTM-FCN, for multivariate time series classification to detect driver drowsiness based on compact facial landmark sequences with low computational complexity. Achieving an impressive 86.90% classification accuracy and an inference speed of 15 frames per second on embedded devices. Arablouei et al. [10] develop an end-to-end deep neural network algorithm for classifying animal behavior using accelerometry data on an IoT device in a wearable collar tag. Combining IIR and FIR filters with a multilayer perceptron, the algorithm enables real-time, low-latency behavior inference without straining the embedded system's computational, memory, or energy resources. Reiser et al. [11] present a novel framework for an end-to-end ASIC implementation of low-power hardware for time series classification using neural networks. This approach optimizes the neural network configuration for both accuracy and energy efficiency and includes a custom-designed hardware architecture, local multi-level RRAM memory, and power-down mode. The framework demonstrates a 95% reduction in energy consumption compared to state-of-the-art solutions when detecting atrial fibrillation in ECG data. Giordano et al. [12] presents a proof-of-concept device for monitoring handheld power tool usage and detecting construction tasks or misuses with an energy-efficient architecture. The device features Bluetooth low energy, NFC connectivity, a temperature and humidity sensor, and an accelerometer. A Tiny ML model is employed to classify tool usage on the edge, achieving 90.6% accuracy. The device demonstrates ultralow power consumption and a potential battery life of up to four years in operation, enabling smart IoT devices with a long lifetime for monitoring tool degradation and usage. Rajapakse et al. [13] explore the workflow for creating machine learning models for embedded devices and survey benchmarking methods. They conclude that only two benchmarking frameworks exist: Banbury et al. [14] use four reference datasets for (keyword spotting, Visual Wake Words, Image Classification, and Anomaly Detection with neuronal network-based models. Sudarshan et al. [15] use 8 datasets for audio classification, image classification and wave classifciation also on neuronal network-based models only. However, this omits state-of-the-art models such as HIVECOTE-V2 and MiniRocket, which tend to perform even better in some use cases. In summary, various applications have been developed to establish tailored solutions; however, there has been limited emphasis on a comprehensive benchmark of existing state-of-the-art algorithms, particularly in the context of embedded devices. Our study aims to address this gap.

III. Pedaling detection

Pedaling detection is a valuable capability that has applications for both classic bicycles and e-bikes. It involves determining whether the rider is actively moving the cranks forward. There are two primary use cases for pedaling detection:

1) For e-bikes, pedaling detection is essential for complying with legislation that mandates motor assistance only when the rider is actively pedaling. In this case, the motor must cease support if the rider stops pedaling, and the bike's propulsion must also halt within a defined time frame [16].
2) Pedaling detection can also serve as an activation mechanism for automatically adjusting suspension elements according to the current riding situation, particularly for mountain bikes. When a rider is actively pedaling, it is undesirable for their energy to be wasted through shock movement. As a result, the compression valves are automatically closed during pedaling. Conversely, when the rider is not pedaling, typically during downhill sections, compression valves are opened to provide maximum suspension performance [17].

There are several methods for achieving pedaling detection. The most common technique involves attaching a magnet to the crank and utilizing a Hall sensor to periodically detect its movement. However, this approach has the drawback of being unresponsive due to the required proximity between the magnet and the Hall sensor. An alternative method is to employ a torque meter, but this increases construction complexity and drive train cost as it necessitates the use of resistance strain gauges. A more cost-effective option is to use a gyroscope mounted on the crank to detect movement. This allows the simple detection of crank movements as only a threshold for the rotation rate needs to be monitored. However, a gyroscope has a high energy consumption, requiring frequent battery charging for the sensor, which can lead to a cumbersome user experience. An alternative solution is to use accelerometer-based crank movement detection. Unlike the gyroscope, the sensor signal cannot be directly interpreted as crank movement due to other acceleration movements present on a bicycle. Consequently, an algorithm that evaluates the sensor signals and classifies them into the two pedaling states—pedaling and not pedaling—would be necessary for this approach.

IV. Experiment Setup

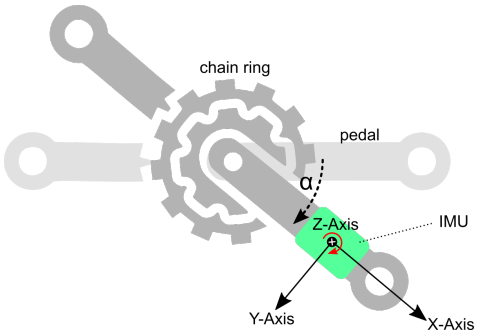

Fig. 1. Crank-mounted accelerometer measurement axis.

In order to identify a suitable algorithm capable of running on an embedded device, we designed an experiment to benchmark several algorithms. Our experiment setup is based on an IMU sensor, which includes an accelerometer mounted to the crank of a bicycle. Figure 1 illustrates the measurement axis of the sensor mounted to the crank, with only the x- and y-axis being relevant for detecting crank movement. The experiment involves collecting several data points that include typical pedaling movements, such as regular forward pedaling, no pedaling, and backward pedaling. It is important to interpret backward pedaling as no pedaling, particularly for the e-bike use case. To collect the necessary data, we used an Adafruit Feather Sense Board based on an NRF52840 SoC, a common foundation for many low-energy commercial systems. This board integrates the ST Microelectronics LSM6DS33 system-in-package featuring a 3D digital accelerometer and a 3D digital gyroscope. We implemented a BLE peripheral that publishes accelerometer and gyroscope signals at a frequency of 20 Hz and a sensor sampling rate of 400 Hz. For data collection, we developed a Flutter smartphone app that acts as BLE central, connecting to our BLE Peripheral implemented on the Adafruit Feather Sense, subscribing to the measurement BLE Characteristic of accelerometer and gyroscope, and storing the measurement samples in a CSV file. We mounted the sensor to the right crank on a 26" mountain bike hardtail with 160mm front suspension and mounted an iPhone running the Flutter app for data collection on the handlebar. We selected a long-standing, ambitious hobby athlete as a rider for data collection to ensure that all typical riding scenarios could be covered. We instructed the rider to ride on local mountain bike trails, including steep ascents and descents, rough terrain, and also dedicated cycleways and public city roads. In a debriefing using the additionally collectd GPS signal we verfied that all typcial riding scenarios were covered. In total, we gathered 173 minutes of actual cycling data involving all previously described situations, resulting in 9.4 MB of data. Additionally, the rider recorded a short, specific dataset for validation purposes, comprising an alternating sequence of pedaling, stopping, and backward pedaling within a 40-second time span. To label the data, we used the gyroscope values, which can easily identify whether the crank is moving in a forward or backward direction. The label "pedaling" is set if the crank is moving at a speed of more than 1 rad/s for a window of 5 measurements; otherwise, the label is set to "not pedaling". This labeling mechanism can be done automatically and does not require human effort. Figure 2 displays both gyroscope and accelerometer measurements for a section of the validation dataset. The rider first pedals forward, then backward, and forward again. In order to prepare the data for time series classification, we preprocessed it as follows: To assess the influence of publishing frequency, we constructed alternative datasets that resample the data to 100 ms and 150 ms intervals, in addition to the original 50 ms interval. We then cut same-sized sequences of length 10 and 20 elements for time series classification purposes. The label of the last row was used to create the label for each sequence. Each new sequence is shifted by exactly one row, resulting in a maximum of 20,480 training rows for the 50 ms interval at sequence length 10.

V. Results

A. Sequence length screening

We began the investigation by first screening possible variations between the different sequence datasets. We assumed that the varying sequence durations resulting from the sample intervals and number of elements could impact the accuracy. For example, a sequence length of 10 elements with an interval of 50 ms considers only sensor signals of a 0.5-second time frame. In contrast, the sequence with 150 ms interval and 20 elements has a 3-second time frame. Consequently, this could influence the shape of the patterns that can be recognized in the sequence by the algorithms. We employed the MiniRocket classification algorithm with default settings, which provides state-of-the-art results with minimal training time to screen for these differences. The results varied only slightly, with a maximum accuracy of 0.650 for a 150 ms sampling interval and 20 elements sequence length and 0.638 for a 50 ms sampling interval and 10 elements sequence length. We expected a loss in accuracy for shorter durations due to less distinguishable signal patterns. However, the slight margin of the difference indicates that only a fraction of the longer sequences are relevant for the classification decision. To decide on the sequence dataset for our further study, we considered the practical effect of the different sequence durations. The dataset with a 150 ms sampling rate and 20 elements results in a 3-second long delay after the start of the sensor till the first pedaling state classification is possible. This is not desirable, particularly for the e-bike use case in which a rider would like support from the motor shortly after the crank movement starts. For this reason, we selected the second-best option at a 50 ms sampling rate and 20 elements, yielding almost the same accuracy of 0.649.

B. Algorithm screening

Using these sequence parameters, we applied the following state-of-the-art time series algorithms: DrCIF, Individual TDE, TSFresh, HIVECOTEV2, Rocket, MiniRocket CNN, and LSTM (see Table I for configuration details). All training

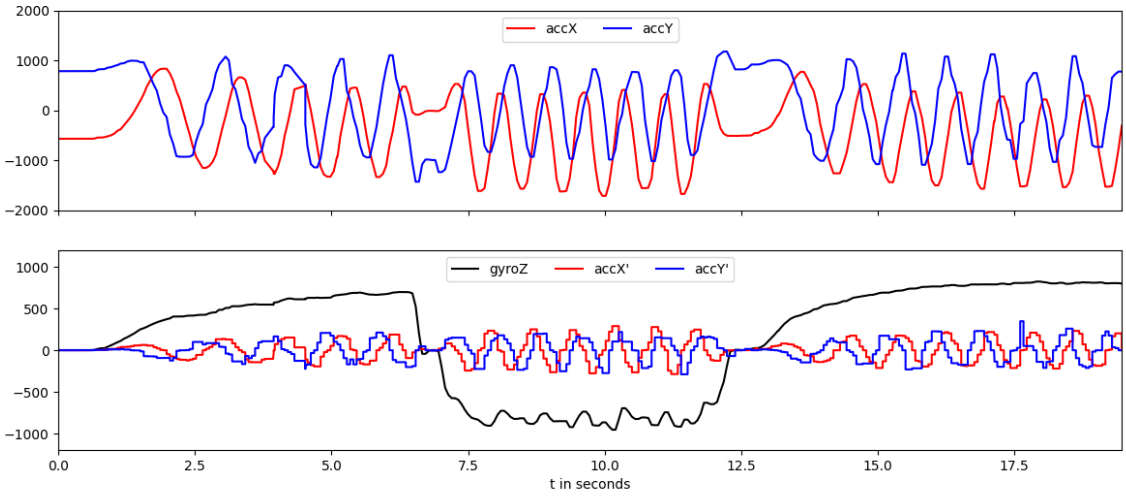

Fig. 2. Accelerometer x-axis and y-axis signal.

TABLE I
ALGORITHMS AND THEIR PARAMETERS IN THE SCREENING EXPERIMENT

Algorithm	Parameters
HIVECOTEV2	Default settings
MiniRocket	num_kernels=10000, RidgeClassifierCV
Rocket	num_kernels=10000, RidgeClassifierCV
LSTM	64 units, tanh activation, Dense(50, relu), Dense(2, softmax)
CNN	64 filters, kernel size 3, pool size 2, Dense(50, relu), Dense(2, softmax)
TSFreshClassifier	default_fc_parameters="minimal", estimators=200 random forest trees
IndividualTDE	Default settings
DrCIF	n_estimators=3, n_intervals=2, att_subsample_size=2

TABLE II
TIME SERIES CLASSIFICATION ALGORITHM RESULTS

Algorithm	Accuracy	Training duration (s)	Inference Duration (s)	Size
HIVECOTEV2	0.815	913925.6	8.9	27.6MB
MiniRocket	0.622	241.6	0.186	1.46MB
Rocket	0.951	206.5	0.078	1.45MB
LSTM	0.947	458.2	0.05	495.49KB
CNN	0.918	126.9	0.046	129.58KB
TSFreshClassifier	0.631	40.04	0.045	4.46MB
Individual TDE	0.584	685	0.033	2.27MB
DrCIF	0.503	40.55	0.009	589.93KB

was conducted on an Intel(R) 64-Core Xeon Gold 6142 CPU with 2.60 GHz and 384 GB RAM. Table II outlines the results for the training of each algorithm, including the actual accuracy and required training time. Additionally, we present two further metrics highly relevant for embedded systems: inference time and memory size. Inference time offers insight into a model's complexity and, thus, its suitability for an embedded device. The memory size of the model is also crucial, as embedded devices have strict memory limitations, and excessively large models cannot be run on them. The results reveal the highest accuracy for the original Rocket algorithm, with an accuracy of 0.951, followed closely by LSTM at 0.947 and CNN at 0.918. There is a more significant gap toward HIVECOTEV2, with an accuracy of 0.813. All other algorithms score lower than the MiniRocket's accuracy. Figures 3 and 4 provides a comparison between the actual pedaling values, as indicated by the gyrometer, and the predicted values generated by MiniRocket. It is evident that the MiniRocket algorithm cannot accurately capture backward pedaling as not pedaling. In contrast, the LSTM can adequately capture these differences, with only a few misclassifications for a small number of sequences. Regarding the suitability for embedded

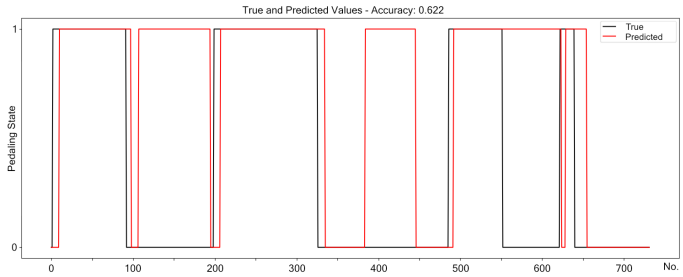

Fig. 3. MiniRocket predicted values vs. true values in validation set

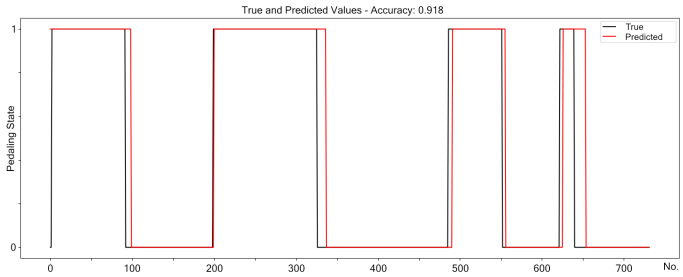

Fig. 4. CNN predicted values vs. true values in validation set

devices, the model size is the limiting factor. Among all benchmarked algorithms, only the CNN is suitable for transfer to a state-of-the-art embedded SoC such as NRF52840, as it falls below the 256 KB limit and exhibits relatively low complexity, indicated by the inference time of 0.046s.

C. Grid Search Hyperparameters for CNN

We conducted an extensive hyperparameter grid search to optimize the performance of our convolutional neural network (CNN) model. This search was integral to determining the most effective configuration for our primary task: detecting bicycle pedaling using embedded systems. The hyperparameters investigated included neurons, kernel size, pool size, and the number of filters. We considered a range of values for each parameter: neurons (25, 50, 75), kernel size (3, 6, 9, 12), pool size (2, 4, 6, 8), and filters (16, 32, 64, 128, 256). To mitigate the inherent randomness in neural network initialization, we ran each combination of these parameters 10 times. The results from these runs were then averaged to obtain a more reliable measure of each configuration's performance. Our search led to the identification of an optimal configuration: 256 filters, 75 neurons, a kernel size of 3, and a pool size of 2. This combination yielded a mean accuracy of 0.934, with minimal deviation in maximum and minimum accuracy scores (see also Figure 4 for comparison of predicted to true values in the validation data). This consistency underscores the stability and reliability of the solution in a real-world application scenario. Following the success of the grid search, we proceeded to adapt our model for practical deployment on an embedded system controller. Utilizing TensorFlow Lite (TFLite) [18], we converted the CNN model into a format suitable our target device: an Arduino Nano33. This device is also based on the NRF52840 SoC but is in contrast to the Aadafruit Feather sense directly supported by TFLite for Microcontrollers. As the NRF52840 supports floating point operations, we used an integer conversion of the model with float fallback. This fully quantizes the models, including parameters and activations, and only uses float32 operators if no integer implementation of the operator is available in TFLite [19]. To enable this full integer quantization, we provided a small representative dataset of 500 rows to the TFLite converter. This allows for calibrating the range of all floating-point tensors in the model, including non-constant tensors such as model input, activations, and model output, by running several inference cycles. This conversion process was crucial for compatibility and offered a significant size reduction of the model. The final model size was compressed to a mere 16KB, making it an excellent fit for the memory constraints of the NRF52840 SoC. We deployed the model to the controller and analyzed inference time and power consumption. The optimized model demonstrated an impressive average inference time of just 4.3ms in our tests. In addition, the pedaling detection consumed 3.11 mA less than a pedal detection based on gyroscope thresholds.

VI. Discussion

Upon initially examining the data and the nature of the problem, we did not expect a complex decision-making process. The problem involves only two data dimensions: x-axis accelerometer and y-axis accelerometer. Additionally, an analysis of the sensor data suggested that the two states, pedaling and not pedaling, could be easily distinguished by the presence or absence of an overlaying sinus pattern. However, backward pedaling, which needs to be interpreted as not pedaling, is more challenging to discern. It also features an overlaying sinus pattern but with a changed order of sensor values. Contrary to our expectations, not all high-ranking algorithms in existing benchmarks could easily solve the decision problem. In particular, HIVECOTEV2, a top-performing algorithm in many benchmarks, performs poorly in comparison. HIVECOTEV2 combines three classifiers (TDE, DrCIF, and an ensemble of Rocket classifiers (ARSENAL) that we apply individually with a Shapelet Transform Classifier in a meta-ensemble algorithm. Each classifier generates a probability distribution over the classes for a given time series. It then combines these probability distributions by weighting them according to an estimate of each classifier's testing accuracy [5]. It is possible that the poor performance of the classification system is due to a combination of poor classifiers (such as DrCIF and TDE) with the best-performing Rocket, resulting in lower accuracy. It is also possible that the ARSENAL classifier is not performing as well as the individual rocket classifier. To determine the actual cause, we would need to investigate the ARSENAL and also the not yet applied Shapelet Transform Classifiers separately. However, because the algorithm and its models require significant resources for training and inference, they are not suitable for embedded devices. Another surprising finding is the significant difference

in performance between MiniRocket and the original Rocket algorithm. The reason for MiniRocket's poor performance is its inability to classify backward pedaling as a non-pedaling state, even when using a large number of kernels (10000). Since we used the same number of kernels for the original Rocket algorithm, the difference must lie in other differences between these algorithms, such as the kernel length, range of weights, bias terms, dilation, and padding. However, a more in-depth analysis would be necessary to determine the cause, which was not the focus of this study. The favorable performance of the two neural network-based methods, CNN and LSTM, is a positive development, as they can be more easily transferred to embedded devices using existing tools such as TFLite. Our study compared the performance of our model on different hardware environments. Surprisingly, the 64 MHz Cortex-M4 NRF52840 performed around 10 times faster than the Intel 64-Core Xeon Gold 6142 CPU. This is because our model was run in Python on the Xeon CPU, which is slower due to its high-level nature and the overhead associated with its interpreter. In contrast, we used a direct C array model representation on the NRF52840, which gives lower-level control over hardware resources and is more efficient. This observation highlights the importance of testing and optimizing models in the intended deployment environment, particularly for machine learning models in varied and potentially resource-constrained environments.

VII. Conclusion

In this study, we investigated the performance of various state-of-the-art time series classification algorithms to detect pedaling states in bicycles. The core objective was to identify an algorithm that not only delivers accurate results but also operates efficiently on an embedded device. Our comprehensive analysis revealed that while several algorithms showed potential, not all could effectively address the decision problem within the constraints of our use case. The top performers were Rocket, LSTM, and CNN models. Although Rocket had the highest accuracy, the CNN model was more suitable for embedded systems due to its compliance with memory limitations and lower complexity. The MiniRocket algorithm struggled to classify backward pedaling accurately, which requires further investigation. A key finding of our study was the effective optimization of the CNN model for the NRF52840 SoC, achieving a mean accuracy of 0.934. The model size was reduced to 16KB with post-training quantization using TfLite. The inference time of the deployed model is 4.3ms, 10x faster than on an Intel Xeon server, and offers significant energy savings over gyroscope-based pedaling detection. Therefore, this study offers insights into the efficacy of time series classification algorithms for pedaling detection in bicycles and emphasizes the criticality of environment-specific model testing and optimization in general.

References

[1] A. Girard, S. Spry, and J. Hedrick, "Intelligent cruise control applications: real-time embedded hybrid control software," *IEEE Robotics & Automation Magazine*, vol. 12, no. 1, pp. 22–28, 2005.

[2] A. P. Ruiz, M. Flynn, J. Large, M. Middlehurst, and A. Bagnall, "The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, vol. 35, no. 2, pp. 401–449, 2021.

[3] N. M. Foumani, L. Miller, C. W. Tan, G. I. Webb, G. Forestier, and M. Salehi, "Deep Learning for Time Series Classification and Extrinsic Regression: A Current Survey," *arXiv*, 2023.

[4] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[5] M. Middlehurst, J. Large, M. Flynn, J. Lines, A. Bostrom, and A. Bagnall, "HIVE-COTE 2.0: a new meta ensemble for time series classification," *Machine Learning*, vol. 110, no. 11-12, pp. 3211–3243, 2021.

[6] A. Dempster, F. Petitjean, and G. I. Webb, "ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels," *Data Mining and Knowledge Discovery*, vol. 34, no. 5, pp. 1454–1495, 2020.

[7] F. Zhu, B. C. Ooi, C. Miao, H. Wang, I. Skrypnyk, W. Hsu, S. Chawla, A. Dempster, D. F. Schmidt, and G. I. Webb, "MiniRocket: A Very Fast (Almost) Deterministic Transform for Time Series Classification," *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 248–257, 2021.

[8] G. Petelin, G. Cenikj, and T. Eftimov, "Towards understanding the importance of time-series features in automated algorithm performance prediction," *Expert Systems with Applications*, vol. 213, p. 119023, 2023.

[9] S. Zu, Y. Jin, D. Yang, and H. Xu, "DrowsyNet: Multivariate Time Series Classification for Embedded Driver Drowsiness Detection," in *2022 8th International Conference on Control, Automation and Robotics (ICCAR)*, 2022, pp. 442–451.

[10] R. Arablouei, L. Wang, L. Currie, J. Yates, F. A. Alvarenga, and G. J. Bishop-Hurley, "Animal behavior classification via deep learning on embedded systems," *Computers and Electronics in Agriculture*, vol. 207, p. 107707, 2023.

[11] D. Reiser, P. Reichel, S. Pechmann, M. Mallah, M. Oppelt, A. Hagelauer, M. Breiling, D. Fey, and M. Reichenbach, "A Framework for Ultra Low-Power Hardware Accelerators Using NNs for Embedded Time Series Classification," *Journal of Low Power Electronics and Applications*, vol. 12, no. 1, p. 2, 2021.

[12] M. Giordano, N. Baumann, M. Crabolu, R. Fischer, G. Bellusci, and M. Magno, "Design and Performance Evaluation of an Ultralow-Power Smart IoT Device With Embedded TinyML for Asset Activity Monitoring," *IEEE Transactions on Instrumentation and Measurement*, vol. 71, pp. 1–11, 2022.

[13] V. Rajapakse, I. Karunanayake, and N. Ahmed, "Intelligence at the Extreme Edge: A Survey on Reformable TinyML," *ACM Computing Surveys*, vol. 55, no. 13s, pp. 1–30, 2022.

[14] C. Banbury, C. Zhou, I. Fedorov, R. M. Navarro, U. Thakker, D. Gope, V. J. Reddi, M. Mattina, and P. N. Whatmough, "MicroNets: Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers," in *Proceedings of the 4 th MLSys Conference*, 2021, pp. 517–532.

[15] 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 *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*, 2021, pp. 883–884.

[16] M. Oberle, M. Stöhr, A. Brandstetter, and S. Gärtner, "Design and Development of a Cyber-Physical System E-Bike Retrofit Prototype," in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2022, pp. 1–8.

[17] G. Nichols, B. Jordan, and S. Fanto, "Pedal activity sensor and methods of pedaling analysis," Patent, 11, 2018. [Online]. Available: https://patents.google.com/patent/US20180011122A1/en

[18] H. Han and J. Siebert, "TinyML: A Systematic Review and Synthesis of Existing Research," in *2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, 2022, pp. 269–274.

[19] Google, "Post-training quantization - TensorFlow Lite," 2023. [Online]. Available: https://www.tensorflow.org/lite/performance/post_training_quantization