Porting and execution of anomalies detection models on embedded systems in IoT: Demo abstract

Demo Abstract: Porting and Execution of Anomalies Detection Models on Embedded Systems in IoT

Bharath Sudharsan*, Pankesh Patel*, Abdul Wahid*, Muhammad Yahya*, John G. Breslin*
Muhammad Intizar Ali§
*Confirm SFI Research Centre for Smart Manufacturing, Data Science Institute, NUI Galway, Ireland
{bharath.sudharsan,pankesh.patel,abdul.wahid,muhammad.yahya,john.breslin}@insight-centre.org
§School of Electronic Engineering, Dublin City University, Ireland, ali.intizar@dcu.ie

ABSTRACT

In the Industry 4.0 era, Microcontrollers (MCUs) based tiny embedded sensor systems have become the sensing paradigm to interact with the physical world. In 2020, 25.6 billion MCUs were shipped, and over 250 billion MCUs are already operating in the wild. Such low-power, low-cost MCUs are being used as the brain to control diverse applications and soon will become the global digital nervous system. In an Industrial IoT setup, such tiny MCU-based embedded systems are equipped with anomaly detection models and mounted on production plant machines for monitoring the machine's health/condition. These models process the machine's health data (from temperature, RPM, vibration sensors) and raise timely alerts when it predicts/detects data patterns that show deviations from the normal operation state.

In this demo, we train One Class Support Vector Machines (OC-SVM) based anomaly detection models and port the trained models to their MCU executable versions. We then deploy and execute the ported models on 4 popular MCUs and report their on-board inference performance along with their memory (Flash and SRAM) consumption. The steps/procedure that we show in the demo is generic, and the viewers can use it to efficiently port a wide variety of datasets-trained classifiers and execute them on different resource-constrained MCU and small CPU-based devices.

KEYWORDS

Offline Inference, Intelligent Embedded Systems, Edge AI.

1 MOTIVATION

The resource-constrained nature of the devices that monitor a machine's condition restricts the standalone (offline) execution of large-high-quality ML models at the device level. Thus, the device manufacturers are obliged to follow an online approach of transmitting local sensor data to cloud services or edge servers for analytics and inference. In such online settings, the cost of the machine's condition monitoring device increases due to the addition of a network module (Lan, 4G or WiFi), this also increases the cyber-security risks as data leaves the device and sometimes creates privacy concerns (GDPR restrictions) as the data is shared with a third-party anomaly detection analytics as a service. Additionally, a continuous connection between the condition monitoring device and the remote service needs to be maintained, leading to network traffic and high bandwidth requirements. Finally, such internet-dependent devices are not self-contained ubiquitous systems. In this paper, we demonstrate to the audience how to perform porting and execution of anomaly detection ML classifiers on resource-constrained devices. The method we show is generic since it can port models trained using any datasets (various features dimension and class count) and can execute the generated models on a wide range of MCUs and small CPUs-based devices. By realizing the demo gathered information, the audience can make their products/devices perform ultra-fast offline analytics, thus eliminating the dependency on internet and cloud subscriptions.

IoTDI '21, May 18–21, 2021, Charlottesvle, VA, USA

ACM ISBN 978-1-4503-8354-7/21/05…$15.00
https://doi.org/10.1145/3450268.3453513

2 IMPLEMENTATION

Nowadays, modern ML frameworks like TF Micro, RCE-NN [6], Edge2Train [5] are focusing on deep optimization and generation of small-size (often in kBs) Neural Networks (NNs) that can directly be flashed and executed on resource-constrained devices [7]. In contrast to NNs, in this demo, we port unsupervised models, then execute them on popular MCUs. Many applications including our anomaly detection use-case, require being able to decide whether a new data sample belongs to the same distribution as existing observations (inlier) or should it be considered as an anomaly (outlier). The 'COVID-away' one class model [9], is the most related use-case based example. In order to detect human hand-to-face movements (considered as inlier), they trained one-class classification models only using the majority class sensor data features and did not consider the outlier distributions for creating the decision boundary. Similarly, in the following, we train the anomaly detection models (step I), then port trained classifiers to its C version (step II), and finally stitch, flash, execute and evaluate the ported models on the MCU of condition monitoring devices (step III).

2.1 Step I: Training for Anomalies Detection

We aim to identify unusual data patterns that do not conform to the expected behavior of machines. These non-conforming patterns are generally known as anomalies, discordant observations, aberrations, novelty, outliers, exceptions, peculiarities/contaminants, strangeness, surprises. In this demo, we use OC-SVM to separate the data of one specific class (target class) from other data. Isolation Forest (iForest), Minimum Covariance Determinant (MCD), Local Outlier Factor (LOF) are also applicable for the same task. For training, we picked the 128 features Gas Sensor Array Drift[1] and

[1]https://archive.ics.uci.edu/ml/datasets/Gas+Sensor+Array+Drift+Dataset

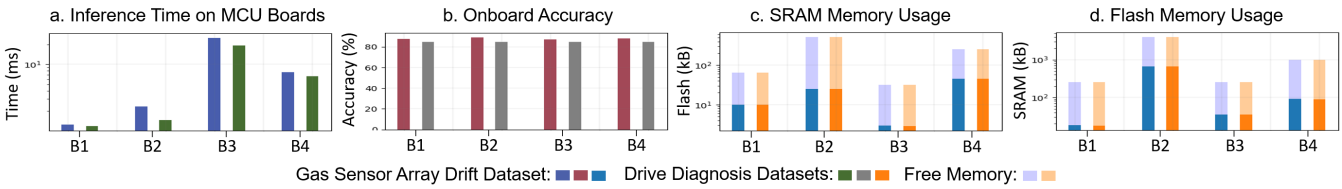

Figure 1: Demo results: inference time, accuracy, and memory consumed by MCUs during the execution of OC-SVM models.

the 48 features Sensorless Drive Diagnosis[2] datasets and trained OC-SVMs from the Python Scikit-learn library.

2.2 Step II: Generating MCU Executable Models

The sklearn-porter [3], m2cgen [2], emlearn [1] are the popular open-source libraries to generate optimized C code, using which IoT use-case models like the 'Adaptive Strategy' SVR model [4] can be ported and deployed on remote IoT devices in order to improve their wireless communication quality. In this demo, we take the trained anomaly/fault detection OC-SVMs and port them to produce its plain C versions that can be deployed and executed on MCUs, small CPUs, FPGAs of choice. Then we write/export the generated C code inside a `.h` file. When the users aim to port tree-based models like decision trees, random forests, we recommend using the SRAM optimized method [8] (OC-SVMs is not yet supported).

2.3 Step III: Executing Models on MCUs

The ported classifier exported in the `.h` file needs to be stitched with the IoT application so it can be called whenever inference needs to be made. During the design phase of the anomaly detection embedded system that mounts on the machines, the users have to include this `.h` model as a header file at the beginning of their program. Then, this `.h` file needs to be compiled along with the device's main application and flashed on MCUs. For predictions, the *predict* function inside the `.h` model file should be passed with values for which it needs predictions. We select four boards, where B1 is STM32 Nucleo (ARM Cortex M4 STM32L432KC MCU), B2 is Generic ESP32 (Xtensa LX6 microprocessor), B3 is Seeedstudio XIAO (ARM Cortex M0 SAMD21G18 MCU), B4 is Nano 33 BLE Sense (ARM Cortex M4F nRF52840 MCU). We benchmark the inference performance and memory consumption of the ported classifiers by executing them on B1-B4. We report the experimental results in Figure 1. For statistical validation, the results correspond to the average of 5 runs. In the following, we analyze the results.

Onboard Accuracy on MCUs. We fed test sets to the OC-SVM models (running on B1-B4) via COM Port, then perform inference and report the onboard accuracy of models in Figure 1 b. Feeding the same test set from the Gas Sensor dataset to all boards, we obtained 87.90 %, 89.10 %, 87.10 %, 88.00 % accuracy for B1 to B4 respectively. Similarly, for Drive Diagnosis dataset, we obtained 84.69 %, 84.59 %, 84.97 %, 84.99 % accuracy for B1 to B4 respectively. From this, we can observe that the same models, from board to board, show only 0.4 - 2 % variation in onboard accuracy.

Inference Performance on MCUs. In Figure 1 a, for both datasets, we report the time consumed (in ms) by B1-B4 for producing inference results. B1 is the fastest as it performed unit inference in 1.23 ms and 1.18 ms for the selected datasets. Followed by B2, which produced results in 2.33 ms and 1.45 ms. When MCUs can perform such fast inference, the battery-drain reduces, increasing the operating time of IoT devices operating in the wild.

Memory Consumption on MCUs. The run-time variables generated during model execution needs to be stored in the SRAM of MCUs. This SRAM space in MCUs is restricted, since adding more leads to higher power leakage and manufacturing costs. The popular open-source boards we chose have only 64 kB to a max of 520 kB of SRAM which restricts deployment and execution of large models. Before flashing, when compiling the ported models and IoT applications, the memory requirements for target boards are calculated by the compiler (such as Arduino IDE, Atmel Studio, Keil MDK, etc.) in use. In Figure 1 c-d, we provide the calculated SRAM and FLASH usage for B1-B4.

3 CONCLUSION

In this demo, we showed how to port and execute anomaly detection models on MCU-based embedded systems. When users follow the steps we provide in this paper, they can make their products/devices perform ultra-fast offline analytics without depending on internet and cloud subscriptions.

ACKNOWLEDGEMENT

This publication has emanated from research supported in part by a research grant from Science Foundation Ireland (SFI) under Grant Number SFI/16/RC/3918 (Confirm) and also by a research grant from Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289_P2 (Insight), with both grants co-funded by the European Regional Development Fund.

REFERENCES

[1] 2020. emlearn. https://github.com/emlearn/
[2] 2020. m2cgen: Code-generation for various ML models into native code.
[3] Darius Morawiec. 2020. sklearn-porter: Transpile trained scikit-learn models.
[4] Bharath Sudharsan, John G Breslin, and Muhammad Intizar Ali. 2020. Adaptive strategy to improve the quality of communication for iot edge devices. In *2020 IEEE 6th World Forum on Internet of Things (WF-IoT).*
[5] Bharath Sudharsan, John G. Breslin, and Muhammad Intizar Ali. 2020. Edge2Train: A Framework to Train Machine Learning Models (SVMs) on Resource-Constrained IoT Edge Devices. In *10th International Conference on the Internet of Things.*
[6] Bharath Sudharsan, John G Breslin, and Muhammad Intizar Ali. 2020. RCE-NN: a five-stage pipeline to execute neural networks (cnns) on resource-constrained iot edge devices. In *10th International Conference on the Internet of Things.*
[7] Bharath Sudharsan, Peter Corcoran, and Muhammad Intizar Ali. 2019. Smart Speaker Design and Implementation with Biometric Authentication and Advanced Voice Interaction Capability. In *27th AICS.* 305–316.
[8] Bharath Sudharsan, Pankesh Patel, John G. Breslin, and Muhammad Intizar Ali. 2021. Ultra-fast Machine Learning Classifier Execution on IoT Devices without SRAM Consumption. In *2021 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops).*
[9] Bharath Sudharsan, Dineshkumar Sundaram, John G. Breslin, and Muhammad Intizar Ali. 2020. Avoid Touching Your Face: A Hand-to-Face 3D Motion Dataset (COVID-Away) and Trained Models for Smartwatches. In *IoT '20 Companion.*

[2] https://archive.ics.uci.edu/ml/datasets/dataset+for+sensorless+drive+diagnosis