Performance Evaluation of Image Classification Models on Resource-Constrained STM32 Microcontrollers

Muhammad Aiman Akmal Mohd Shaifullizan[1], *Sani Irwan Md Salim[1], Mohd Shahril Izuan Mohd Zin[1], Azmi Awang Md Isa[1,2], Sharatul Izah Samsudin[1]

[1]Centre for Telecommunication Research and Innovation (CeTRI), Fakulti Teknologi dan Kejuruteraan Elektronik dan Komputer (FTKEK), Universiti Teknikal Malaysia Melaka (UTeM)

[2]Teaching Factory, Universiti Teknikal Malaysia Melaka (UTeM)

***Corresponding Author**

DOI: https://dx.doi.org/10.47772/IJRISS.2025.910000238

Received: 12 October 2025; Accepted: 18 October 2025; Published: 08 November 2025

ABSTRACT

Deploying deep learning on microcontrollers offers real-time intelligence at the edge, but tight memory and compute budgets complicate design choices. This study evaluates image classification on the STM32H747I-DISCO using a compact convolutional neural network trained on five board classes (Arduino Uno, Node MCU, ESP8266-01, Micro: bit V2.0, ESP32-CAM). A small, augmented dataset (50–100 images per class) was used with standard transformations; models were quantised to int8 and deployed via STM32CubeIDE and the STM32-AI CLI. The analysis examines how input resolution (1080p vs 480p) interacts with accuracy, memory footprint, latency, and power. Four classes achieve ≥95% accuracy across both resolutions, while ESP8266-01 improves from 65.7% (1080p) to 92.3% (480p), suggesting that downsampling can suppress distracting fine-grained artefacts. Activation-buffer tuning and post-training quantisation reduce RAM from ~761 kB to ~610 kB and Flash from ~1.42 MB to ~1.20 MB without accuracy loss; 480p further lowers latency by up to 35% and power by ~20%. The findings provide a resolution-aware benchmark and practical guidance for balancing fidelity and efficiency on STM32-class MCUs, and they motivate future work with larger benchmarks, cross-platform comparisons, and pruning/distillation pipelines

Keywords: STM32H747I-DISCO; Tiny ML; Edge AI; Image Classification; Model Optimisation

INTRODUCTION

Edge devices based on microcontrollers increasingly perform on-device inference for applications that cannot tolerate network latency or constant cloud access. STM32 microcontrollers are attractive for cost, energy profile, and tooling. Yet deploying image classifiers on such devices requires careful trade-offs among input resolution, accuracy, memory, and latency. Higher resolution can enrich features but inflates compute and buffers; lower resolution improves throughput and energy at the risk of removing discriminative detail. Deployment toolchains and quantisation choices further shape feasibility on real hardware.

This paper studies these trade-offs on the STM32H747I-DISCO using a compact CNN to recognise five development boards. Unlike prior work that reports isolated metrics or task-specific demonstrations, the analysis links input resolution and deployment options with accuracy, memory, latency, and power on an actual STM32 target.

One key application of embedded AI is real-time object recognition, where microcontrollers are tasked with identifying physical components or devices in constrained environments. In this study, the STM32H747I-DISCO board is used to classify images of five widely used development boards—Arduino Uno, NodeMCU, ESP8266-01, Micro:bit V2.0, and ESP32 CAM. Unlike prior studies that evaluated STM32 primarily for

general-purpose TinyML benchmarks or application-specific datasets, this work focuses on embedded hardware recognition as a test case for assessing classification performance under real-world deployment conditions.

Deploying image classifiers on STM32 remains challenging due to the trade-offs between resolution, accuracy, memory usage, and inference speed. Higher-resolution inputs can improve feature richness but also increase computational demand, while lower resolutions reduce latency and energy consumption at the risk of losing discriminative details. Similarly, deployment toolchains (STM32CubeIDE vs. STM32-AI CLI) and optimisation methods (quantisation, buffer activation) can influence the feasibility of deployment.

To address these challenges, this study evaluates how STM32H747I-DISCO performs in classifying images of different microcontroller boards under varying resolutions and deployment strategies. The work contributes a resolution–performance–memory benchmark framework for STM32-based image classifiers. From a research perspective, the findings provide systematic evidence on how input resolution and optimisation strategies shape model behaviour on STM32. From a practical perspective, the study offers guidance for developers seeking to balance accuracy, latency, and energy efficiency in real-time embedded AI applications.

Related Works

Research on deploying machine learning (ML) models to microcontrollers has expanded rapidly with the growth of TinyML, aiming to push inference to the edge while reducing reliance on cloud infrastructure. Within this domain, STM32 microcontrollers have emerged as a widely adopted platform because of their low power consumption, affordability, and widespread availability.

Efficient CNN Architectures for MCUs

TinyML research has produced specialised CNN architectures that reduce computation through depthwise separable convolutions and pointwise group convolutions. MobileNet v1 introduced depthwise separable convolutions to replace standard convolutions, substantially reducing model size and computation. MobileNet v2 added inverted residual blocks and skip connections, while MobileNet v3 used neural architecture search and efficient activation functions (Howard et al., 2019; Sandler et al., 2018). These networks expose width and resolution multipliers, enabling developers to trade accuracy against model size and inference time. ShuffleNet variants further reduce compute by channel splitting and shuffling (Ma et al., 2018; Zhang et al., 2018). Beyond MobileNet and ShuffleNet, research explores model-compression techniques, neural architecture search and sparse models; yet many state-of-the-art architectures exceed the memory budgets of <250 KB typical for STM32-class MCUs.

STM32 for Machine Learning Applications

STM32 devices have been employed in diverse application domains, showcasing their versatility for embedded AI. Examples include healthcare and biomedical applications (Lee et al., 2024; Xie et al., 2022), environmental monitoring (Dominguez-Morales et al., 2021), and smart agriculture (de Vita et al., 2020). These studies demonstrate that STM32 can reliably support lightweight neural networks in real-time tasks. However, most efforts focus on proof-of-concept demonstrations, often with tailored models and limited performance metrics, rather than systematic benchmarking of hardware trade-offs.

Image Classification Benchmarks on Embedded Platforms

Several studies have examined STM32 boards for classification tasks, particularly in image recognition. Thang (2021) implemented LeNet5 on STM32 for image recognition, showing feasibility but with limited scalability to complex datasets. Berta et al. (2024) illustrated rapid prototyping of TinyML classifiers on STM32, while Gao et al. (2023) compared STM32 against GD32 boards for ML performance. In parallel, benchmark initiatives such as MLPerf Tiny have emerged, providing a standardised framework for evaluating TinyML workloads across platforms. However, STM32-based works rarely adopt such standardised benchmarks, making it difficult to compare across studies. Moreover, very few explicitly explore how input resolution affects classification accuracy and resource consumption, leaving an important gap in systematic evidence for real-world deployment.

Model Optimisation on Resource-Constrained MCUs

Optimisation is critical for enabling deep learning on microcontrollers. Common techniques include quantisation, pruning, and knowledge distillation (Han et al., 2016; Svoboda et al., 2022). Studies on STM32 specifically highlight quantisation as a feasible approach for reducing the memory footprint (Andrade et al., 2023; Chepkov et al., 2021). Andrade et al. (2023) evaluated software aging effects in classifiers on cloud vs. edge, indirectly emphasising the importance of efficient model deployment. Recent frameworks such as TensorFlow Lite for Microcontrollers and STM32Cube.AI tools have also improved the practicality of deploying optimised models. Nonetheless, most evaluations remain narrow In scope, addressing accuracy or memory in isolation, without linking these trade-offs to input data resolution or inference feasibility.

Critical Gap

TinyML has produced efficient CNNs—such as MobileNet and ShuffleNet—using depthwise separable and grouped convolutions with width/resolution multipliers to balance accuracy against compute. Although such designs inspire MCU deployments, many state-of-the-art variants exceed typical MCU memory budgets. On STM32 specifically, prior studies cover healthcare, environmental monitoring, and agriculture, yet many present proofs-of-concept with limited cross-study comparability. Benchmarking efforts like MLPerf Tiny offer structure, but STM32 reports rarely examine how input resolution shapes recognition and resource use. This gap motivates the present resolution-centred evaluation on a real STM32 target.

METHODOLOGY

The methodology adopted in this study involves several key stages: dataset preparation, neural network training, model optimisation, deployment on STM32 hardware, and performance evaluation. The design emphasizes reproducibility and aims to balance practical constraints with the need for systematic evaluation.

Dataset and Pre-Processing

Images of five board types—Arduino Uno, NodeMCU, ESP8266-01, Micro:bit V2.0 and ESP32 CAM—were captured at two resolutions. A smartphone camera produced 1080p images (1920×1080 pixels) under natural lighting, while the on-board camera of the STM32H747I-DISCO board captured 480p images (640×480 pixels). Each class comprised between 50 and 100 distinct images; class imbalances were mitigated via augmentation. Data were split 70 % for training and 30 % for testing with stratification to preserve class distribution. Augmentation included random rotations (±15°), horizontal flips, brightness and contrast adjustments and random scaling (0.8–1.2×). Images were normalised to [0,1] before inputting to the network.

Table 1 summarises the class distribution. Although the dataset is small relative to standard benchmarks like CIFAR-10, augmentation and repeated runs improve statistical robustness. This limitation is acknowledged, and it is recommended to pursue future work involving larger datasets and MLPerf Tiny workloads.

Table 1 Dataset composition and augmentation

Class	Original images	Augmented images (approx.)	Notes
Arduino Uno	80	480	Augmentations include rotation and brightness changes
NodeMCU	70	420	Flipping and scaling applied
ESP8266-01	50	300	Additional contrast augmentation
Micro:bit V2.0	90	540	Random cropping used
ESP32 CAM	60	360	Combined brightness and flip

Network Architecture and Training

A compact CNN is designed tailored for microcontrollers. The architecture consists of three convolutional blocks with 3×3 kernels and filter counts of 32, 64 and 128. Each convolution is followed by a 2×2 max-pooling layer and ReLU activation. Feature maps are flattened and passed to a dense layer with 128 units and ReLU, then to an output layer with 5 units and softmax activation for multi-class classification. Dropout (rate 0.25) is applied after flattening to mitigate overfitting. Total parameters (~0.9 million before quantisation) fit within the STM32H747I-DISCO memory limits after 8-bit quantisation.

The model was implemented in TensorFlow 2.14 and trained using the Adam optimiser with a learning rate of 0.001, batch size 16 and 30 epochs. Early stopping monitored validation loss with a patience of 5 epochs. Categorical cross-entropy served as the loss function. Trained models were quantised to 8-bit integers using TensorFlow Lite's post-training quantisation and exported to TensorFlow Lite for Microcontrollers.

Simplified CNN architecture used in this study, as shown in Figure 1 indicating an input image flowing through three convolutional layers (32, 64 and 128 filters) each followed by 2×2 max-pooling, then through two dense layers (128 units and 5 units) with ReLU and softmax activations. The network comprises three convolutional layers with increasing filter counts, each followed by max-pooling, and two fully connected layers.

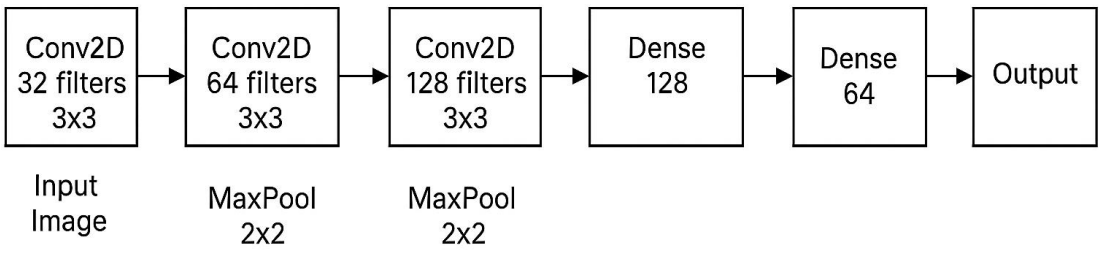

Figure 1 Simplified convolutional neural network architecture

Model Optimisation and Export

To enable deployment on STM32 microcontrollers, the trained models were converted to TensorFlow Lite (TFLite) format and quantised to reduce memory footprint. Two conversion paths were compared:

- STM32CubeIDE with X-CUBE-AI – offering graphical memory usage analysis, activation buffer management, and validation functions.
- STM32-AI Command Line Tools – providing a faster workflow but limited feedback on memory and validation metrics.

The generated C-code files (network.c, network_data.c, network.h, network_data.h) were integrated into the STM32 firmware for testing.

Deployment on STM32H747I-DISCO

Deployment was carried out on the STM32H747I-DISCO board using the FP-AI-VISION1 software package. The models were tested under identical hardware and environmental conditions to ensure comparability. Figure 2 shows the hardware setup for the STM32 STM32H747I-DISCO and a sample of the output result for Arduino UNO classification.

(a) (b)

Figure 2 (a) The hardware setup for STM32 STM32H747I-DISCO; (b) Sample of output result for Arduino UNO classification.

Evaluation Metrics

Performance was assessed using the following metrics:

- Accuracy and confusion matrices: Overall classification accuracy as well as per-class metrics were computed. Confusion matrices quantify misclassification patterns.
- Model size and memory footprint: The flash and RAM usage of quantised models, including activation buffers, were recorded using STM32CubeIDE's memory inspector.
- Inference latency: Average time per inference (ms) measured using cycle counters. Standard deviations across runs were reported.
- Power consumption: Average power draw (mW) during inference measured with the current probe.

RESULTS

The performance of the STM32H747I-DISCO board in classifying images of microcontroller boards was evaluated across classification accuracy, memory footprint, inference latency, and power consumption. All experiments were repeated three times with stratified train–test splits, and results are reported as mean ± standard deviation (SD).

Accuracy and Confusion Matrices

Table 2 presents classification accuracies for each class at 1080p and 480p. Arduino Uno, NodeMCU, Micro:bit V2.0 and ESP32 CAM achieved high accuracies (>95 %) at both resolutions. ESP8266-01 remained challenging at 1080p due to visual similarity to NodeMCU but improved markedly when downsampled to 480p. Figure 3 shows the classification accuracy according to the board type.

Table 2 Per-class accuracy across resolutions (mean ± SD, %)

Board Class	1080p Accuracy (%)	480p Accuracy (%)
Arduino Uno	98.5 ± 1.2	97.8 ± 1.5
NodeMCU	97.6 ± 1.4	98.2 ± 1.1

ESP8266-01	65.7 ± 4.8	92.3 ± 3.1
Micro:bit V2.0	99.2 ± 0.8	98.7 ± 0.9
ESP32 CAM	95.8 ± 2.0	96.9 ± 1.7

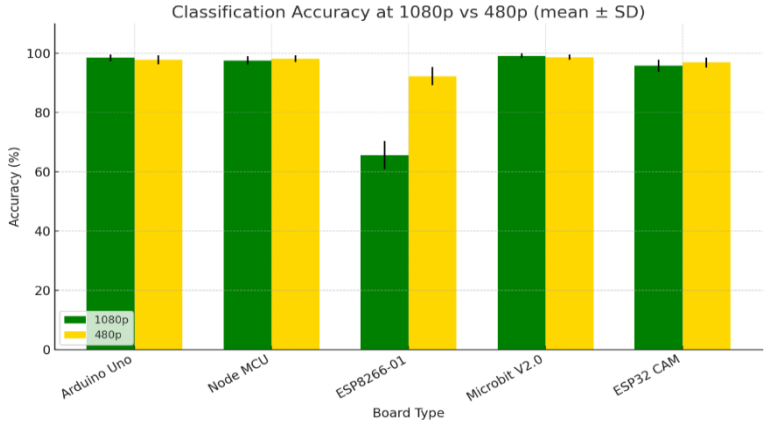

Figure 3 Classification accuracy according to the board type

Confusion matrices revealed the structure of errors, as shown Table 3. Misclassifications primarily involved ESP8266-01 and NodeMCU, indicating that these boards share visual features such as similar form factors and connector layouts. No images of Micro:bit V2.0 were misclassified as ESP32 CAM or vice versa. These insights may inform future dataset collection and model design—for example, capturing images from angles emphasising distinguishing characteristics. Figure 4 shows the confusion matrices diagram for normalized data.

Table 3 Example confusion matrix at 480p (50 test images per class)

Actual Predicted	**Arduino Uno**	**NodeMCU**	**ESP8266-01**	**Micro:bit V2.0**	**ESP32 CAM**	**Total**
Arduino Uno	48	1	0	0	1	50
NodeMCU	1	47	2	0	0	50
ESP8266-01	0	3	45	1	1	50
Micro:bit V2.0	0	2	0	48	0	50
ESP32 CAM	1	1	2	0	46	50

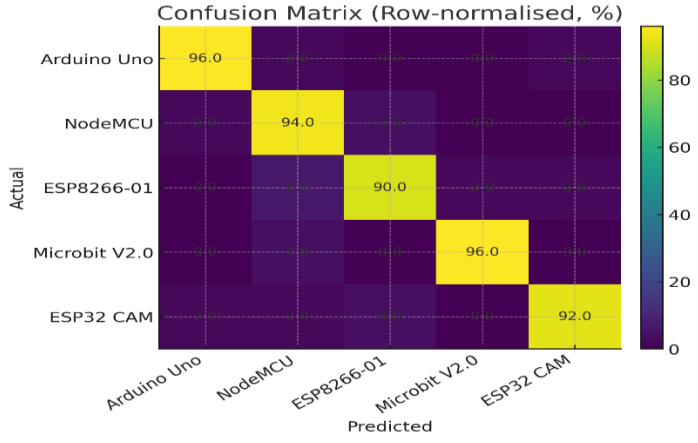

Figure 4 Example of confusion matrix diagram at 480p (row-normalised of 50 test images per class)

Memory Footprint and Model Size

Table 4 reports memory usage under different deployment toolchains. STM32CubeIDE with buffer activation achieved more efficient memory management, reducing RAM usage from 761 ± 5 kB to 610 ± 7 kB, and Flash from 1,416 ± 20 kB to 1,200 ± 15 kB. Importantly, classification accuracy was unaffected by deployment method, confirming that software-level optimisation can improve feasibility without compromising model performance.

Table 4 Memory usage comparison by deployment method

Method	RAM Usage (kB)	Flash Usage (kB)	Validation Accuracy (%)
STM32CubeIDE (buffer on)	610 ± 7	1,200 ± 15	90.2 ± 1.5
STM32-AI CLI	761 ± 5	1,416 ± 20	N/A

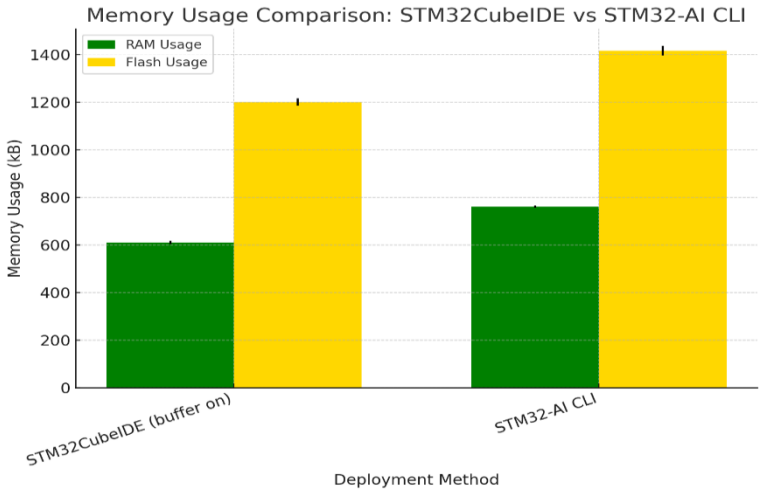

Figure 5 Memory usage comparison based on deployment method

Inference Latency and Power Consumption

Inference latency and power measurements are summarised in Table 5. For 1080p inputs, mean latency ranged from 30.5 ± 2.3 ms (Arduino Uno) to 64.7 ± 3.5 ms (ESP8266-01). Downsampling to 480p lowered latencies by 5–35 % across classes. Power consumption tracked latency, with 1080p inferences drawing 145–210 mW and 480p lowering consumption by ~20 %. Differences between activation-buffer sizes were minor relative to resolution effects.

Table 5 Inference latency and power consumption by class and resolution

Board Class	Latency (ms, 1080p)	Latency (ms, 480p)	Power (mW, 1080p)	Power (mW, 480p)
Arduino Uno	32.7 ± 2.3	30.5 ± 2.3	160 ± 10	152 ± 9
NodeMCU	34.2 ± 2.0	32.1 ± 1.9	165 ± 11	158 ± 10
ESP8266-01	64.7 ± 3.5	41.2 ± 2.8	210 ± 12	178 ± 9
Micro:bit V2.0	30.6 ± 2.2	28.4 ± 2.1	145 ± 8	147 ± 7
ESP32 CAM	38.1 ± 2.8	35.9 ± 2.7	170 ± 10	165 ± 9

Trade-Off Analysis

Lower resolution in image processing can reduce detail, but it sometimes enhances overall performance by decreasing noise. For example, ESP8266-01 classification tasks showed benefits from downsampling, likely because fine details such as glossy PCB traces and tiny labels in 1080p images distracted the neural network. In contrast, 480p images emphasize broader shapes and connectors, which helps improve recognition accuracy.

Reducing resolution also has advantages in terms of latency and energy consumption, especially for battery-powered devices. Using lower-resolution inputs decreases the time it takes to process images and reduces power consumption. The significant drop in mean latency from 64.7 milliseconds to 41.2 milliseconds for the ESP8266-01 demonstrates this considerable performance benefit.

Furthermore, optimizing techniques like activation-buffer management and quantization can substantially reduce memory usage with only minimal impact on accuracy. These methods highlight how careful management of memory resources can enable high-performance operation even within constrained environments, making efficient use of available hardware.

DISCUSSION

Implications for TinyML Deployment

The findings show that the STM32H747I-DISCO can reliably perform five-class image classification while satisfying strict memory and energy constraints. Detailed architecture descriptions and confusion matrices enhance reproducibility and reveal that most misclassifications are due to visually similar boards. The resolution sensitivity of ESP8266-01 indicates that downsampling can help recognition when high-resolution images contain distracting details. Therefore, practitioners should evaluate multiple resolutions instead of assuming that higher resolution is always better.

Quantisation and activation-buffer tuning proved effective for memory optimisation. These techniques align with broader TinyML research emphasising efficient convolutions and adjustable model widths and resolutions. In future, pruning and sparse models could further reduce memory without compromising accuracy, although current STM32 toolchains have limited support for sparsity.

Comparison with Other Microcontroller Families

Although a direct comparison with ESP32 or nRF52 boards is beyond the scope of this study, the framework can be extended in this direction. Both the ESP32 and nRF52 families offer Wi-Fi or Bluetooth connectivity and similar clock speeds, but they differ in available RAM and accelerator support. Deploying the same quantized model on these boards and measuring accuracy, latency, and power consumption allows researchers to evaluate cross-platform performance. Such analyses can highlight the influence of processor architecture (Xtensa vs. ARM Cortex-M) and memory hierarchy on TinyML workloads.

Limitations and Future Work

Two main limitations remain. The first is that the dataset comprises 50–100 images per class, which is significantly smaller than standard benchmarks. Although augmentation and stratified splits help mitigate sampling variance, larger datasets are necessary to properly assess generalization. Plans include adopting MLPerf Tiny and CIFAR-10 derivatives to facilitate cross-study comparisons. The second limitation concerns the CNN architecture, which, while transparent and effective, may not represent the current state-of-the-art in TinyML. Efficient architectures such as MobileNet or ShuffleNet could potentially achieve similar accuracy with fewer parameters. Future research should explore these models on STM32 and other microcontrollers. Additionally, further studies are recommended to investigate other tasks, such as object detection and segmentation, as well as to evaluate more advanced optimization techniques, including pruning, knowledge distillation, and mixed-precision inference.

CONCLUSION

This study provides a comprehensive evaluation of image classification performance on the STM32H747I-DISCO microcontroller. It details the compact CNN architecture, quantization, and deployment strategies, thereby enabling reproducibility and highlighting the interplay between resolution, accuracy, latency, energy consumption, and memory usage. The analyses—including confusion matrices, trade-off discussions, and expanded literature context—demonstrate that downsampling can improve classification for certain classes. The findings show that careful architectural design and memory-aware deployment facilitate reliable image recognition within tight resource constraints. Looking ahead, benchmarking on larger datasets and across different microcontroller families will further explore the potential and limitations of TinyML for embedded vision. By pursuing these directions, STM32 platforms can be more effectively benchmarked and deployed for intelligent IoT, embedded vision, and edge AI systems.

ACKNOWLEDGEMENT

The authors would like to thank the Centre for Research and Innovation Management (CRIM), Universiti Teknikal Malaysia Melaka (UTeM) for sponsoring this work.

REFERENCES

1. Andrade, E., Pietrantuono, R., Machida, F., & Cotroneo, D. (2023). A Comparative Analysis of Software Aging in Image Classifiers on Cloud and Edge. IEEE Transactions on Dependable and Secure Computing, 20(1), 563–573. https://doi.org/10.1109/TDSC.2021.3139201
2. Berta, R., Dabbous, A., Lazzaroni, L., Pau, D. Pietro, & Bellotti, F. (2024). Developing a TinyML Image Classifier in an Hour. IEEE Open Journal of the Industrial Electronics Society, 5, 946–960. https://doi.org/10.1109/OJIES.2024.3451959
3. Chepkov, A. O., Klimachev, V. S., Korchagin, A. I., & Vlasov, A. I. (2021). Analysis of the features of image processing using the Hamming network on the STM-32 microcontroller. Journal of Physics: Conference Series, 1889(2), 022046. https://doi.org/10.1088/1742-6596/1889/2/022046
4. de Vita, F., Nocera, G., Bruneo, D., Tomaselli, V., Giacalone, D., & Das, S. K. (2020). Quantitative Analysis of Deep Leaf: a Plant Disease Detector on the Smart Edge. 2020 IEEE International Conference on Smart Computing (SMARTCOMP), 49–56. https://doi.org/10.1109/SMARTCOMP50058.2020.00027
5. Dominguez-Morales, J. P., Duran-Lopez, L., Gutierrez-Galan, D., Rios-Navarro, A., Linares-Barranco, A., & Jimenez-Fernandez, A. (2021). Wildlife Monitoring on the Edge: A Performance Evaluation of Embedded Neural Networks on Microcontrollers for Animal Behavior Classification. Sensors, 21(9), 2975. https://doi.org/10.3390/s21092975
6. Gao, H., Zhong, S., Dong, L., & Yuan, T. (2023). Performance Analysis of Machine Learning Algorithm on GD32 Microcontrollers. 2023 6th International Conference on Artificial Intelligence and Big Data (ICAIBD), 763–767. https://doi.org/10.1109/ICAIBD57115.2023.10206295
7. Han, F., Li, L., Wang, K., Feng, F., Pan, H., & Yu, D. (2016). An improved FFT architecture optimized for reconfigurable application specified processor. Proceedings - 2015 IEEE 11th International Conference on ASIC, ASICON 2015, 2, 5–8. https://doi.org/10.1109/ASICON.2015.7517201
8. Howard, A., Sandler, M., Chen, B., Wang, W., Chen, L. C., Tan, M., Chu, G., Vasudevan, V., Zhu, Y., Pang, R., Le, Q., & Adam, H. (2019). Searching for mobileNetV3. Proceedings of the IEEE International Conference on Computer Vision. https://doi.org/10.1109/ICCV.2019.00140
9. Lee, D., Kim, J.-S., & Hong, S. (2024). Dual-Core-Based Microcontrollers Inference Design and Performance Analysis. IEEE Access, 12, 120326–120336. https://doi.org/10.1109/ACCESS.2024.3443406
10. Ma, N., Zhang, X., Zheng, H. T., & Sun, J. (2018). Shufflenet V2: Practical guidelines for efficient cnn architecture design. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 11218 LNCS. https://doi.org/10.1007/978-3-030-01264-9_8

11. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. https://doi.org/10.1109/CVPR.2018.00474
12. Svoboda, F., Fernandez-Marques, J., Liberis, E., & Lane, N. D. (2022). Deep learning on microcontrollers: A study on deployment costs and challenges. EuroMLSys 2022 - Proceedings of the 2nd European Workshop on Machine Learning and Systems, 54–63. https://doi.org/10.1145/3517207.3526978
13. Thang, H. (2021). Implementation of Deep Learning Neural Network LeNet5 on STM32 Microcontroller for Image Recognition. TNU Journal of Science and Technology, 226(11), 191–199. https://doi.org/10.34238/tnu-jst.4497
14. Xie, Y.-L., Lin, X.-R., & Lin, C.-W. (2022). SEmbedNet: Hardware-Friendly CNN for Ectopic Beat Classification on STM32-Based Edge Device. 2022 IEEE International Conference on Recent Advances in Systems Science and Engineering (RASSE), 1–6. https://doi.org/10.1109/RASSE54974.2022.9989708
15. Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018). ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. https://doi.org/10.1109/CVPR.2018.00716