Quantitative Analysis of Deep Leaf: a Plant Disease Detector on the Smart Edge

Fabrizio De Vita, Giorgio Nocera, Dario Bruneo, Valeria Tomaselli, Davide Giacalone, Sajal K. Das

Abstract—Diagnosis of plant health conditions is gaining significant attention in smart agriculture. Timely recognition of early symptoms of a disease can help avoid the spread of epidemics on the plantations. In this regard, most of the existing solutions use some AI techniques on smart edge devices (IoTs or intelligent Cyber Physical Systems), typically equipped with a hardware like sensors and actuators. However, the resource constraints on such devices like energy (power), memory and computation capability, make the execution of complex operations and AI algorithms (neural network models) for disease detection quite challenging. To this end, compression and quantization techniques offer viable solutions to reduce the memory footprint of neural networks while maximizing performance on the constrained devices. In this paper, we realized a real intelligent CPS on top of which we implemented an AI application, called Deep Leaf, running on a microcontroller of the STM32 family, to detect coffee plant diseases with the help of a Quantized Convolutional Neural Network (Q-CNN) model. We present a quantitative analysis of Deep Leaf by comparing five different deep learning models: a 32-bit floating point model, a compressed model, and three different types of quantized models exhibiting differences in terms of accuracy, memory utilization, average inference time, and energy consumption. Experimental results show that the proposed Deep Leaf detector is able to correctly classify the plant health condition with an accuracy of 96%, thus demonstrating the feasibility of our approach on a Smart Edge platform.

Index Terms—Q-CNN, edge computing, intelligent CPS, plant disease, smart agriculture

I. INTRODUCTION

In an era in which the number of devices connected to the Internet is in the order of 20 billions, the management and processing of data generated from such devices has become extremely challenging. To this end, Cloud computing has been an effective paradigm that offloads the data to remote servers providing large storage and high power processing services accessible by any device with internet connectivity [1]. However, with the advancement of IoT technology and Internet of Things (IoTs), emerging systems and applications have new requirements that the Cloud is usually unable to handle. Such systems work both as sensors and actuators, provided with two interfaces that enable them to interact with each other as well as the surrounding physical world, leading to Cyber Physical Systems (CPS) [2].

Indeed, the IoTs have significant impacts on our daily lives, as they have changed the way we interact with the physical world. While the market and economic potential of CPS and IoT applications are increasingly rapidly, artificial intelligence (AI) is playing an important role in implementing various applications on the smart edge. Machine learning helps build Intelligent Cyber Physical Systems (ICPS), allowing them to make context-aware autonomous decisions [3]–[5].

Although Cloud computing is useful for realizing intelligent CPSs, it also poses some limitations especially in terms of latency and security. This has led to a paradigm shift of the computations from the Internet to the location where the data are collected. This emerging paradigm is called Edge computing in which the processing is done on the devices “close” to the data [6]. However, moving the computations from the Cloud to the Edge comes with a new set of challenges due to the hardware constraints on these devices. Given that, these devices are designed to be energy efficient because of their always-on feature that allows them to continuously “listen” to the surrounding environment, such a feature limits the computing power of the device, thus making the execution of complex operations or algorithms (e.g., neural network models) extremely challenging. Recently, sophisticated compression and quantization models have been proposed that aim to run on constrained hardware like edge devices, while maintaining good performance [7].

In this paper, we present a quantitative analysis of a real working ICPS, on top of which we implemented an AI application for detecting coffee plant diseases in smart agriculture. The application, called Deep Leaf, utilizes Quantized Convolutional Neural Network (Q-CNN) model running on a microcontroller of the STM32 family. We use the X-CUBE-AI tool provided by STMicroelectronics to implement five different deep learning models, namely a 32-bit floating point model, a compressed model, a quantized model using Tensor Flow Lite (TFLite) converter, a quantized model using an integer representation, and yet another quantized model using a fixed point (Q-format or Qm,n) representation. We conduct extensive experiments to quantitatively analyze the performance of these deep learning models in terms of their accuracy, inference time, memory utilization, and energy consumption.

The rest of the paper is organized as follows. Related works in Section II highlight the differences with our approach.
Section III describes the hardware and AI tool used to execute the compression and quantization models. Section IV provides a detailed description of Deep Leaf detector that we implemented for quantitative analysis. Section V presents experimental results from testing the five deep learning models. Section VI concludes the paper with directions of future works.

II. RELATED WORKS

Timely and accurate diagnosis of plant diseases is important in (smart) agriculture for control of disease spread and high yield of crops. In [8], a survey is presented on different plant diseases along with their detection techniques. This section reviews existing AI applications on the smart edge and summarizes their differences with our novel approach.

In [9], the authors proposed a plant disease detector based on a Random Forest (RF) classifier. They adopted a Histogram of an Oriented Gradient for extracting features, which are then used to train the classifier. The accuracy of this classifier is very low (about 70%), whereas our proposed model based on the Q-CNN results in much higher accuracy of 96%.

In [10] is presented a plant disease detection system based on a Convolutional Neural Network (CNN) and a Learning Vector Quantization (LVQ) algorithm, which is able to reach a very good accuracy of 90%. Since the detector is not deployed on an ICPS, it is not suitable for efficient implementation on a Smart Edge. Our main contribution is to demonstrate how compression and quantization techniques can reduce the complexity of our Q-CNN model and run on a microcontroller with limited hardware capabilities.

Considering that computations are getting closer to the edge devices, the authors in [11], [12] utilized quantization techniques on neural networks for deploying energy efficient AI applications on the Smart Edge. In [13], a pruning technique based framework is proposed to compress and accelerate CNN models during both the training and inference processes. Experimental results show very good performance in terms of accuracy and inference time. However, it may not always be possible to run compression techniques on a microcontroller due to low available RAM and flash. In contrast, we analyze quantitatively the differences between a compressed model and a quantized model putting in evidence advantages of the latter in terms of inference time and energy consumption.

The authors in [14] proposed a Keyword Spotting System for a speech recognition application run on a Cortex-M7 based STM32F746G-disco development board. Due to hardware constraints, they adopted a quantization technique based on ARM CMSIS-NN kernels [7] to reduce memory footprint of the proposed Deep Neural Network (DNN) models while improving energy efficiency. Unlike our approach, this work provides a comparison of only accuracy between several DNN models and their corresponding quantized versions.

In [15], an embedded application is presented for acoustic event detection based on compact Recurrent Neural Networks (RNN). A fixed point 8-bit quantization using the ARM CMSIS-NN kernels is adopted. Instead, in this paper we use the STMicroelectronics X-CUBE-AI tool to deploy a quantized DNN directly into a microcontroller. Such a software (described in the next section) allows us to directly transfer a pre-trained floating point model into an ST board, providing a set of functionalities to validate and test the converted model.

III. ICPS PLATFORM AND MODEL CONVERSION METHODS

This section describes the STMicroelectronics based hardware and software platform used in realizing a real ICPS (intelligent CPS) environment. The platform will be used to conduct experiments with Deep Leaf detector.

A. STM32 Discovery Board

The STM32F746GDISCOVERY kit is a complete demonstration and development platform for STMicroelectronics Arm® Cortex®-M7 core-based STM32F746NG microcontroller. It includes 1 Mbyte of Flash memory and 340 KBytes of RAM in BGA216 package. The board includes complete peripherals for developing user applications before porting on the custom boards. The Discovery kit enables a diversity of applications by taking advantage of audio, multi-sensor support, AI, graphics, security, video and high-speed connectivity features. Fig. 1 shows the front and back sides of the board.

![STM32F746NG-disco board](a) Front. (b) Back. Fig. 1: STM32F746G-disco board.

The STM32F746NG microcontroller features include on board peripherals: a 4.3” RGB 480x272 color LCD-TFT with capacitive touch screen, two ST-MEMS digital microphones, 128-Mbit Quad-SPI Flash memory, 128-Mbit SDRAM (64 Mbits accessible), two user and reset push-buttons. It also includes connectors to expand the board capabilities: Camera, microSD™ card, Audio line in and out jacks, 2×USB with Micro-AB, ARDUINO Uno V3 expansion connectors.

The on-board ST-Link/V2-1 programmer enables virtual COM port, mass storage and debug port functions. With the power supply, the board can be powered from 3.6 to 12 volts, thus facilitating its integration into existing circuitry.

B. X-CUBE-AI Tool

The STM32F746NG microcontroller features and the Discovery kit are configurable from STM32CubeMX tool that allows to manage an expansion package, called X-CUBE-AI. It has the capability to automatically convert a pre-trained neural network, and then integrate an optimized library into the user’s project. The expansion package also provides metrics to validate the neural network model generated on both PCs and STM32 devices, without having to develop ad-hoc C codes. The X-CUBE-AI significantly extends the feature-set

1[https://www.st.com/en/embedded-software/x-cube-ai.html](https://www.st.com/en/embedded-software/x-cube-ai.html)
of STM32CubeMX by importing an artificial neural network trained with many of the most popular libraries freely available today, such as Keras, TensorFlow Lite, ONNX, Caffe, Lasagne, or ConvnetJS. This tool supports 8-bit quantization of the Keras models and Tensorflow Lite quantized networks. It also allows to import larger networks than STM32 microcontroller resources, storing the weights on external Flash memory, and activation buffers on external RAM modules.

Integrating a neural network into an embedded system is challenging due to memory occupancy. If the model can fit on the embedded memory of a processor, huge energy savings is achieved. This motivates the development of a fully embedded memory implementation of CNN. Therefore, in addition to quantization, STM32Cube.AI also offers a weighted compression strategy that can be exploited to build a large model to fit in the limited STM32 MCU memory. More details on the compression and quantization techniques exploited by X-CUBE-AI, will be presented in the next two sections.

C. Compression Technique

X-CUBE-AI enables the user to compress the weights in case a network cannot fit in the selected Micro Controlling Unit (MCU) due to higher flash requirements. The implemented weights and bias of the compression can be of 3 different types: “none” implying the target-factor of 1 (i.e., no compression applied), x4 and x8 corresponding to target-factors of 4 and 8, respectively. Compression is only applicable to dense layers, which is motivated by the fact that most of the weights of a neural network are in the fully connected layers. Each dense layer is taken into account individually and the weights in it are clustered using the K-means algorithm, where the number of centroids is defined as a function of the compression factor we want to achieve. Assuming that the uncompressed weights are represented in single precision floating point, the number of centroids for a given target-factor, can be calculated as:  \( n_{\text{centroids}} = 2^{32}/(\text{target-factor}) \)

The compression technique represents each weight with the value of closest centroid obtained through K-means algorithm.

D. Quantization Techniques

Besides a native TensorFlow Lite quantized model, X-CUBE-AI has the capability to import a Keras-based quantized model, opportunely reshaped by the Command Line Interface (console level) provided by X-CUBE-AI. When a quantized model is loaded, the code generator quantizes weights and bias, and the associated activation from floating point to 8-bit precision. These are mapped on the optimized and specialized C implementation of the supported kernels. The objective of this technique is to reduce the model size while also improving the CPU and hardware accelerator latency (including power consumption) with little degradation in the model accuracy.

A TFLite model is a self-contained file that is able to handle floating-point and/or quantized model (weight/bias are already converted). This is not the case for Keras models: Keras h5 file does not support quantized params or meta information. Hence a reshaped Keras model file *.h5 and a proprietary tensor format configuration (json) file are requested by the tool. For Keras-based model, due to the tensor format configuration file, the code generator is able to quantize (convert) the weights/bias and associated activation from floating-point to integer precision. To generate the reshaped Keras model file and associated tensor format configuration file, the stm32ai application integrates a complete Keras post-training quantization process (command). The Keras post-training quantization process goes through a flow depicted in Fig. 2.

In particular, the quantization flow includes three steps: (i) Reshape the model, (ii) Quantize the model, and (iii) Predict with emulated quantized model. (For more details, see [16].) By default, X-CUBE-AI supports the standard 32-bit floating point arithmetic for the most imported models. For quantized models, integer or less-precision arithmetic, no standard is defined. X-CUBE-AI supports two integer-base arithmetic: \( \text{Qm,n} \) and Integer. \( \text{Qm,n} \) arithmetic is a signed fixed-point number (in two’s complement format) where the number of fractional bits \( n \) and the number of integer bits \( m \) are specified with a constant and fixed resolution. For the integer arithmetic, each real number \( r \) is represented as a function of the quantized value \( q \), a scale factor (arbitrary positive real number), and a zero_point parameter.

IV. DEEP LEAF DETECTOR

Being able to perform an initial analysis on plants suspected of being infected by a disease, is paramount to save them and prevent the spread of epidemics on the plantations. Since continuous internet connection may not always be guaranteed and the number of plants to analyze could be very large, the Cloud computing paradigm may require huge amount of time for diagnosis of disease. Therefore, there is a need to implement a system capable of performing on-site early analysis of plants health condition, thus supporting the human operator during the diagnosis process.

However, the deployment of applications on the Edge constrained devices could be challenging, especially for AI applications which mostly require a lot of computing power. Here the challenge includes the deployment of a software that can be run on the Edge with energy efficiency, while maintaining the system performance. As explained in Section

**Fig. 2: Quantization steps.**
III, different techniques can be adopted to reduce the complexity of an application such that it can be run on a resource constrained device. Depending on the available hardware and on the application context, an approach may often be preferred over others. The primary aim of our work is to provide a detailed quantitative analysis highlighting the advantages and disadvantages of various approaches.

This section first describes Deep Leaf, an AI application implemented on a real ICPS that we built. Specifically, the application is able to detect the main biotic diseases that typically affect the coffee trees by analyzing the pictures of leaves. Fig. 3 depicts the prototype we created using an STM32F746G-disco board (see Section III) connected to an STM2F4DISCAM module, both embedded in a polymethylmethacrylate box equipped with some LEDs which are used to light the coffee leaf while the camera module takes the picture.

A. Dataset Description

The dataset for our application is made up of pictures of healthy and diseased coffee leaves which are affected by the main biotic stresses of the coffee tree. Fig. 4 shows the healthy (Fig. 4a) and diseased leaves: miner (Fig. 4b), brown leaf spot generically called “phoma” (Fig. 4c), and rust (Fig. 4d). From the original dataset, we extracted a total of 1,290 images of which 274 images belonged to the healthy class, 343 images to the miner class, 350 images to the phoma class, and 353 images to the rust class. All images were 512x256 pixels with a ratio of 2:1. However, the camera module attached to the board takes pictures with a squared ratio. Therefore, we resized all the images into a square format of 224x224 pixels (ratio 1:1) avoiding stretching and cropping.

The images were divided into training, validation and test datasets by using the ratio 80%, 10%, and 10%, respectively. However, after the division, the total number of samples for the training and validation datasets were too low (about 1,021 and 134 samples respectively). To overcome this issue that would have caused over-fitting of our model, we applied the data augmentation technique to our images. Such a technique allows to generate new instances of data from the existing one by applying different random transformations such as movements, rotations, flips, zooms, and skews [17]. Data augmentation also allows us to increase the number of training samples, thereby making the model more robust.

Another problem we faced, was related to the quality of the camera module. Such a module has a resolution lower than a smartphone camera so its pictures are noisy and have lower quality as compared to the ones available in the dataset. Therefore, in order to force the model to be more tolerant to low lighting conditions and noise, a vignette effect was added on half of the images – both on the training and validation datasets. Moreover, for each image in these two datasets, it was randomly decided whether to apply a Gaussian noise, Poisson noise, sparkle noise, or to maintain the image without noise.

At the end of data augmentation operations, we reached a total of 23,462 images for the training dataset and 3,057 images for validation dataset. For better visualization, Table I summarizes the number of samples for each set (training, validation, and test) after the data augmentation process.

| Class  | Training | Validation | Test | Total images |
|--------|----------|------------|------|--------------|
| Healthy| 5,055    | 621        | 27   | 5,703        |
| Miner  | 5,932    | 785        | 32   | 6,609        |
| Phoma  | 6,297    | 781        | 34   | 7,112        |
| Rust   | 6,179    | 920        | 40   | 7,139        |
| Total  | 23,462   | 3,057      | 133  | 26,652       |

B. CNN Description

The task of detecting plant diseases is accomplished via a supervised multiclass classification problem with four classes (e.g., healthy, miner, phoma, and rust). In particular, the deep learning technique we used to implement the classifier is a CNN whose complete structure is depicted in Fig. 5.
From a topological point of view, a CNN is composed by two parts. The first part, consisting of the convolutional and the max-pooling layers, is responsible for the feature extraction process, through which the network is able to learn the internal structure of the input images (i.e., lines, edges, shapes, etc.). The second part of the CNN makes the actual classification (or detection in this case). It is usually composed of a sequence of fully connected layers that take as input a vector containing the features extracted by the network in the previous stage, and that are terminated by a softmax layer which outputs the most probable class associated with the input image. Analyzing Fig. 5, it is possible to observe these two distinct parts in our CNN. The feature extraction part starts from the input layer and contains all the layers until the last convolutional one (i.e., 7). The classification part instead contains the fully connected layers until the output one.

Starting from the input layer 1 which takes as input 224x224 pixels RGB images, the CNN we designed consists of 4 convolutional layers (i.e., 2, 4, 5, and 7) containing 32, 64, 128, and 64 filters, a map size of 8x8, 3x3, 3x3, and 3x3, respectively, with strides of size 4x4 and 2x2 applied only to the first two convolutional layers. With respect to the pooling layers (i.e., 3 and 6), we adopted the max-pooling with a map and strides of size 2x2.

The classifier network part is composed by two fully connected layers (i.e., 8, 9) containing 128 and 32 neurons, respectively. Moreover, after each fully connected layer we used two dropout layers with 0.2 and 0.5 drop rates to prevent network over-fitting. Finally, for the output layer 10, we used a softmax function made up of four output neurons where each one represents one of the classes we want to classify.

For better visualization, Table II summarizes the model parameters used in the design of the CNN. Specifically, we used Rectified Linear Unit (ReLU) as activation function for each layer, the Stochastic Gradient Descent (SGD) as optimizer with a learning rate of 0.01, and set the maximum number of training epochs to 150. Finally, to prevent over-fitting and under-fitting, we used an early stopping technique setting with a patience term equal to 30.

| CNN model architecture | No. of layers | Optimizer | SGD |
|------------------------|--------------|-----------|-----|
| Activation function    | ReLU         | Learning rate | 0.01 |
| Pooling method         | Max-pooling  | Patience   | 30  |
| Dropout rate           | [0.5, 0.2]   | Max training epochs | 150 |

C. Deep Leaf Workflow

This subsection presents how Deep Leaf works and is able to detect the health condition of a coffee leaf. At the start of the (microcontroller) board, the prototype asks the user to press “red button” (Fig. 6a) under the display to activate the camera module and LEDs that help better focus the subject (i.e., the leaf). Then, the user can insert a leaf inside the box using a slot accessible from the back side and verify its positioning with respect to the camera frame through the display. When the leaf is inside the frame, the user can press on the LCD touch screen to take a picture of the leaf. After this phase, the image is passed as input to the CNN that will perform the inference process. Finally, the classification output is shown to the user in a text format with an icon indicating if the leaf is healthy or not, as shown in Fig. 6b.

V. EXPERIMENTAL RESULTS

This section provides a quantitative analysis of experimental results and comparisons of five different models of Deep Leaf, a plant disease detection application. In particular, we compare the original 32-bit floating point model, its compressed version, the quantized model using TFLite (Tensor Flow Lite) converter, the quantized version using the fixed point Qm,n format, and the quantized model using the integer format, as discussed in Section III. For each model we measure the accuracy, precision, and recall; and compare the compressed and quantized models in terms of flash and RAM utilization, average inference time, and average energy consumption. The results are obtained from our testbed using STM32F746G-disco board, the X-CUBE-AI tool for deploying models on the microcontroller, and the test dataset in Table I.

A. 32-bit Floating Point Model

This model achieved an accuracy of 96.24% on the test dataset. For better understanding of the results, we computed the confusion matrix (Fig. 7a), an n-by-n table (where n is the number of classes) that measures the performance of a machine learning algorithm. It contains the following values: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN), where the first two refer to the number of correctly classified samples while the last two refer to the number of samples misclassified by the algorithm.

Besides providing the performance summary, the confusion matrix helps extract three important indices, namely: precision, recall, and F1-score. The precision is computed as the ratio of TP to the total number of samples classified as positive by the algorithm: \( \text{precision} = \frac{TP}{TP + FP} \). This index tells us how precise the algorithm is in determining correct class.

The recall is the ratio of TP to the total number of samples that are actually positive: \( \text{recall} = \frac{TP}{TP + FN} \). This index provides the sensitivity of the algorithm.

Due to the variability of these indices, it may be difficult to determine the performance of a machine learning algorithm vis-a-vis others. For this purpose, the F1-score estimates how good a model is, returning a value between 0 and 1. This index is defined as: \( F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \), where 1 (respectively,
TABLE III: Precision, recall and F1-score of the 32-bit floating point, Compressed, TFlite quantized, and Integer quantized models.

| Class | Precision | Recall | F1-score | Support |
|-------|-----------|--------|----------|---------|
| Health | 0.96 | 1.00 | 0.98 | 27 |
| Miner | 0.94 | 0.94 | 0.94 | 32 |
| Phoma | 0.90 | 0.94 | 0.97 | 34 |
| Rust | 0.95 | 0.97 | 0.96 | 40 |

Then, the Keras model was exported in a **.h5** file that occupies 2.1 MBytes on the disk. As described in Section III-A, the STM32F746G-disco board is equipped with 1 MByte of flash memory; hence the model can not run efficiently on the microcontroller. To deploy the model on the board, we used compression and quantization techniques as in Section III.

The compression model used a higher compression factor (i.e., x8) in order to reduce the flash memory occupancy to 729.49 KBytes. However, the model could not still run on the microcontroller because the RAM required (619.78 KBytes) was too high while the board supports only 340 KBytes. To overcome this issue, we configured STM32F746G-disco to provide access to the external SDRAM.

B. Compressed Model

We evaluated the performance of the compressed model by computing the confusion matrix and extracting its metrics. The results obtained on the test dataset are the same as reported in Fig. 7a. Since the precision, recall, and consequently the F1-score depend on the confusion matrix values, these quantities coincide with the values reported in Table III. The health class representing healthy leaves produced very high values (close to 1) for precision, recall, and F1-score. This means that the model was able to correctly classify leaves belonging to this class. For other classes, such as miner representing a disease where larvae eat the leaf tissue, phoma and rust representing two diseases where a fungus deteriorates the leaf, the Deep Leaf detector was able to correctly detect them with a precision of 0.94, 1.00, and 0.95, respectively; a recall of 0.94, 0.94, and 0.97; and F1-score of 0.94, 0.97, and 0.96 respectively.

As regards to the inference time, by running the model on the test dataset, each inference on the average lasted 1,022.81 ms with an average of 220,927,803 CPU cycles and 31,593,500 of Multiply-and-Accumulate (MACC) operations considering a CPU clock speed of 216 MHz. In this context, a useful metric is represented by the ratio cycles/MACC, which highlights the efficiency of the model implementation (i.e., the smaller this ratio the more efficient is the implementation). For the compressed model, the cycles/MACC is 6.99.

The STM32CubeMX software also estimates the average energy consumption of a board depending on its setup. More precisely, by properly setting the board configuration in terms of clock frequency and active peripherals, it is possible to estimate the average consumption of the current. The peripherals enabled by our application include: Digital CaMera Interface (DCMI) to communicate with the camera module, Direct Memory Access (DMA) and Direct Memory Access 2D (DMA2D) for image manipulation, LCD-TFT Display Controller (LTDC) to control the display, and a timer (TM1).

For the compressed model, the external SDRAM and Quad Serial Peripheral Interface (QSPI) are necessary for communication. Here the average absorbed current per millisecond (denoted as $i$) was equal to 305.67 mA. Considering a supply voltage ($V_{dd}$) equal to 3.3 V and an average inference time ($t$) of 1022.81 ms, the average energy consumption ($E$) for each inference is computed as: $E = i \cdot V_{dd} \cdot t$.

Thus, the compressed model has an average energy consumption of 1,031.72 mJ for each inference. If the board is powered with a configuration of 3 Ni-MH(A2500) batteries (arranged in series), each with a capacity of 2,500 mA and a nominal voltage of 1.2 V, the expected lifetime of batteries would be 2 days and 14 hours considering a scenario in which one inference is performed per minute.

C. TFLite Quantized Model

The next model we analyze is obtained by adopting a quantization technique that uses TFLite converter. It reduces the flash occupancy to 251.79 KBytes and the RAM to 33.25 KBytes, and hence allows the model to run on the board without external SDRAM. The confusion matrix coincides with the ones computed for the floating point and compressed models. This is significant because the TFLite Quantized model exhibits the same performance as the other two models in terms of accuracy, precision, recall, and F1-score.

In this model, each inference took an average of 364.44 ms (about a quarter of the compressed model) with an average of 78,719,324 CPU cycles and 31,469,676 of MACC operations. In terms of cycles/MACC, we obtained very good results with a value of 2.50 which is about 3 times lower than the one obtained by the compressed model.

The TFLite model is very energy efficient; in fact it was not necessary to enable QSPI and SDRAM peripherals which have a huge impact on the absorbed current. For this new configuration and assuming the same voltage $V_{dd}$, the average absorbed current per millisecond ($i$) was 135.66 mA. Thus, the
average energy consumption for each inference was 163.15 mJ (about 6.5 times less than the compressed model).

### D. Qm,n Format Quantized Model

The quantization using Qm,n representation format is one of the techniques supported by the X-CUBE-AI tool. Such a quantization generated a model occupying 250.44 KBytes on flash and 32.35 KBytes on RAM. In terms of performance, this model exhibited a slightly reduced accuracy of 95%. This is because the Qm,n representation format results in a higher loss of information during DNN conversion than other quantization schemes, and consequently degradation of the performance.

Fig. 7b depicts the confusion matrix for the Qm,n quantized model. As compared to Fig. 7a, this new model misclassified 2 instances of the miner class with a consequent reduction of the precision, recall, and F1-score (see Table IV), which are still very good with values ranging between 0.93 and 1.0. More precisely, for the health, phoma and rust classes the values are comparable with the ones obtained in Table III. Most of the impact is caused by the miner class having a reduced recall value below 0.90. This also caused a decrease in the corresponding F1-score and the overall accuracy.

#### TABLE IV: Precision, recall and F1-score of the classifier for Qm,n quantized model.

| Class | Precision | Recall | F1-score | Support |
|-------|-----------|--------|----------|---------|
| Health | 0.93      | 1.00   | 0.96     | 27      |
| Miner  | 0.93      | 0.88   | 0.90     | 32      |
| Phoma  | 1.00      | 0.94   | 0.97     | 34      |
| Rust   | 0.93      | 0.95   | 0.97     | 40      |

As for the inference time, we registered an average value of 299.60 ms with an average number of 64,712,955 CPU cycles, 31,593,508 MACC operations, and a cycles/MACC ratio of 2.05 which has the lowest value among the five models considered in this quantitative analysis. Consider the same hardware configurations as the TFLite quantized model. Assume the expected average current consumption is 135.66 mA, the voltage $V_{dd}$ is 3.3 V, and the average inference time lasted is 299.60 ms. Then the average energy consumption for each inference is 134.12 mJ, which is again the lowest.

#### E. Integer Format Quantized Model

The final model also used the X-CUBE-AI tool but selected a quantization technique with an integer representation format. This model occupied 251.79 KBytes on flash and 32.60 KBytes on RAM. The model obtained has the same accuracy and same confusion matrix as the original floating point model.

The model performed each inference on the test dataset with an average of 360.41 ms, requiring an average of 77,848,890 CPU cycles, 31,409,668 MACC operations, and a cycles/MACC ratio equal to 2.47. Assuming again the same hardware configuration as the TFLite and Qm,n quantized models, the integer format quantized model reported an average energy consumption 161.34 mJ for each inference.

Table V summarizes the results for each model. The symbol “...” for the floating point model indicates an empty value due to the impossibility of directly running this model on the board.

#### F. Lessons Learned

A closer look at Table V and Fig. 8, allows us to gain interesting insights on the two techniques explained above. In general, the compression technique produces a model that achieves the largest values of flash and RAM occupancy, inference time, and cycles/MACC. In our Deep Leaf application context, although the compressed model yielded the same accuracy, precision, recall, and F1-score as the original floating point model, the compressed model is not practical for Edge computing, where the devices are battery driven. This is because the compressed model needs to use of SDRAM module which has very high energy consumption. On the contrary, the quantized models using the TFLite converter and X-CUBE-AI tool exhibit very good results and heavily reduce the flash and RAM occupancy. Specifically, the TFLite quantized model and the the integer format quantized model attain the same performance as the original floating point model, but with lower values of inference time and energy consumption as compared to the compressed model.

These results demonstrate the effectiveness of the quantization techniques over the compressed one when working with the DNNs consisting of a large number of layers. However, the performance of the quantization is heavily influenced by the type of the representation format and the tuning of the parameters (i.e., number of bits, quantization steps, number of intervals, etc.). In this sense, they require the implementation of efficient algorithms which can take significant time to find the best set of quantization parameters. For example, the X-CUBE-AI tool took about 5 hours to quantize our model, and, in general, the required time increases with the complexity of the network. For our Deep Leaf application, the Qm,n format quantized model resulted in the best trade-off in terms of accuracy, flash occupancy and energy consumption as showed by the plot, as illustrated in Fig. 9 that depicts the 3D scatter plot of the models with respect to these parameters.

#### VI. CONCLUSIONS

We designed an intelligent CPS on top of which we implemented Deep Leaf, an AI application for disease detection on coffee leaves. The application is deployed on an STM32F746G-disco board powered by a microcontroller of the STM32 family, using X-CUBE-AI tool developed by STMicroelectronics. This paper presented a quantitative analysis of five different models and compared the performance of the proposed classifier in terms of accuracy (precision, recall, F1-score), flash and RAM occupation, average inference time, and energy consumption. Our experimental results demonstrated that the quantization techniques outperformed the compression method in every metric we measured, by careful selection and tuning of the parameters which may be time consuming due to complexity of the models. Future works will be devoted to the improvement of the application performance to detect multiple diseases and to the analysis of more sophisticated compression and quantization techniques.
### TABLE V: Measurement on models.

| Model               | Flash (KB) | RAM (KB) | AVG Inf. Time (ms) | Accuracy | AVG Energy Cons. (mJ) | Cycles/MACC |
|---------------------|------------|----------|--------------------|----------|-----------------------|-------------|
| Floating Point      | -          | -        | -                  | -        | -                     | -           |
| Compressed          | 729.49     | 619.78   | 1,022.81           | 0.96     | 1,031.72              | 6.99        |
| Quantized TFLite    | 251.79     | 33.25    | 364.44             | 0.96     | 163.15                | 2.50        |
| Quantized Integer Format | 251.79 | 32.60   | 360.41             | 0.96     | 161.34                | 2.47        |
| Quantized Qm,n Format | 250.44 | 32.35   | 299.60             | 0.95     | 134.12                | 2.05        |

![Fig. 8: Histogram plot that compares the compressed and quantized models.](image)

![Fig. 9: 3D scatter plot of the models putting in relationship flash occupation, accuracy and average energy consumption.](image)

### REFERENCES

[1] D. Bruneo et al., “I/Ocloud: Adding an IoT Dimension to Cloud Infrastructures,” *Computer*, vol. 51, no. 1, pp. 57–65, January 2018.

[2] J. Gubbi et al., “Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions,” *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[3] F. De Vita et al., “Using deep reinforcement learning for application relocation in multi-access edge computing,” *IEEE Communications Standards Magazine*, vol. 3, no. 3, pp. 71–78, Sep. 2019.

[4] D. Cogliati et al., “Intelligent cyber-physical systems for industry 4.0,” in Int’l Conf. on Artificial Intelligence for Industries, 2018, pp. 19–22.

[5] R. Varenne, J. M. Delorme, E. Plebani, D. Pau, and V. Tomaselli, “Intelligent recognition of tcp intrusions for embedded micro-controllers,” in *New Trends in Image Analysis and Processing – ICIAP 2019*, M. Cristani, A. Prati, O. Lant, S. Messelodi, and N. Sebe, Eds. Cham: Springer International Publishing, 2019, pp. 361–373.

[6] P. Bellavista et al., “A Survey on Fog Computing for the Internet of Things,” *Pervasive and Mobile Computing*, vol. 52, pp. 71–99, 2019.

[7] L. Lai, N. Suda, and V. Chandra, “Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus,” *arXiv* preprint, 2018.

[8] S. S. Kumar and B. K. Raghavendra, “Diseases detection of various plant leaf using image processing techniques: A review,” in 5th Int’l Conf. on Advanced Computing Communication Systems, 2019, pp. 313–316.

[9] S. Ramesh et al., “Plant disease detection using machine learning,” in *2018 International Conference on Design Innovations for ICS Compute Communicate Control (ICDSC)*, April 2018, pp. 41–45.

[10] M. Sardogan et al., “Plant leaf disease detection and classification based on cnn with lvq algorithm,” in 2018 3rd International Conference on Computer Science and Engineering (UBMK), Sep. 2018, pp. 382–385.

[11] M. Rusci et al., “Work-in-progress: Quantized nns as the definitive solution for inference on low-power arm mcus,” in *International Conf. on Hardware/Software Codesign and System Synthesis*, 2018, pp. 1–2.

[12] Y. Guo, “A survey on methods and theories of quantized neural networks,” *arXiv* vol. abs/1808.04752, 2018.

[13] J. Luo et al., “Thinet: A filter level pruning method for deep neural network compression,” in *IEEE International Conference on Computer Vision*, Oct 2017, pp. 5068–5076.

[14] Y. Zhang et al., “Hello edge: Keyword spotting on microcontrollers,” *arXiv* preprint, 2017.

[15] Cerutti et al., “Compact recurrent neural networks for acoustic event detection on low-energy low-complexity platforms,” *IEEE Journal of Selected Topics in Signal Processing*, pp. 1–1, 01 2020.

[16] “X-Cube-AI User manual.” [Online]. Available: https://www.st.com/content/ccc/resource/technical/document/

[17] A. Mikołajczyk and M. Grochowski, “Data augmentation for improving deep learning in image classification problem,” in *2018 international interdisciplinary PhD workshop (IIPhDW)*. IEEE, 2018, pp. 117–122.