Tiny Machine Learning for High Accuracy Product Quality Inspection

Andrea Albanese, *Student Member, IEEE,* Matteo Nardello, *Member, IEEE,*
Gianluca Fiacco, and Davide Brunelli, *Senior Member, IEEE*

I. Introduction

The Industrial Internet of Things (I-IoT) is becoming essential for companies to optimize their production processes. Adding distributed intelligence along the production lines can lead to consistent cost savings. Machine learning can be a crucial tool for supervising products and production processes [1]. It is possible to generate models that can quickly inspect the quality of a product (e.g., the presence or absence of a correct label on a bottle or the classification of defects on the surface) with statistical methods and databases, thus avoiding possible waste in production lines [2]. So far, cloud-based architectures have been used for many industrial inspection applications by exploiting servers with unlimited computing resources to carry out complex data processing. The cloud computing approach usually employs with high-resolution industrial cameras providing high-quality images for accurate analysis. However, even though they provide superior service quality, they are expensive solutions in terms of cost and performance [3].

This approach is also highly conditioned by the network quality, which can introduce a significant latency or even lead to service degradation if the connection is lost. Also, scalability is affected as more sensors generate more pressure on the cloud infrastructure (i.e., bandwidth and data storage requirements). A solution can be represented by low-cost intelligent electronic devices, as such embedded platforms can guarantee a reduced development cost without affecting the quality of service [4]. Using the computational capacity that embedded systems have achieved, it is possible to design complex visual inspection systems – powered by deep learning algorithms – directly on intelligent sensors and actuators, creating the so-called Tiny Machine Learning (tinyML). TinyML expands the so-called *edge computing* paradigm bringing complex data processing closer to the data source. This approach improves the application's responsiveness and efficiency, and reduces the amount of data transmitted [5]. Thus, cloud computing limitations associated with data throughput and costs are avoided. In-pixel processing architectures [6] could be used in the near future to further reduce the amount of data exchange between the sensor and the nearby microcontroller. Also, event cameras can improve system's efficiency by transmitting only detected conditions in the image (i.e., events in a scene). However, these solutions are still under development and unavailable on the market.

This paper presents the design, implementation, and evaluation of an automatic visual inspection I-IoT system based on image data processing with tinyML, designed for large-scale product quality inspection. We addressed the specific requirements of a company leader in the production of plastic parts through the injection molding process. They provided the dataset used to develop the deep learning models, the specifications about the responsiveness of the visual inspection system, and the list of possible anomalies during the process. For this reason, this work consists of a unique scenario with real case problems. The system is currently used in a dozen of the company's machines for long-time and large-scale testing. Two convolutional neural networks (CNNs) for image classification were trained, tested, and compared, namely MobileNetV2 [7] and SqueezeNet [8]. Then, the CNN models were compressed and deployed in the target platform, namely OpenMV Cam H7 Plus, for image processing and neural classification directly on the microcontroller unit (MCU). Results highlight the perfect fit for this use case due to their optimized structure for resource-constrained environments.

In particular, the main contributions of this paper are:

- The design of a cyber-physical system for product quality inspection, capable of detecting the defects of plastic molded objects (Figure 1).
- The design, optimization, and deployment of tiny neural networks (NNs) for object classification on resource-constrained cameras.
- The creation of a custom dataset used to train and test different NN architectures.
- The evaluation and comparison of performances between the two tiny NNs, namely MobileNetV2 and SqueezeNet.
- The system characterization by examining its execution time and energy consumption during image preprocessing and classification.

The rest of the paper is organized as follows: in Section II related work is discussed. Section III describes the system architecture, while Section IV presents the developed tiny NNs and the used optimization algorithms. In Section V the overall system's implementation is discussed, highlighting the pre-

A. Albanese, M. Nardello, G. Fiacco and D. Brunelli are with the Department of Industrial Engineering, University of Trento, Italy. e-mail: {name.surname}@unitn.it

This work was supported by STMicroelectronics, Italy, and Rosa Micro srl, Ceggia, Italy.

Funded by the European Union under NextGenerationEU, through the project "INEST- Interconnected Nord-Est Innovation". Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or The European Research Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

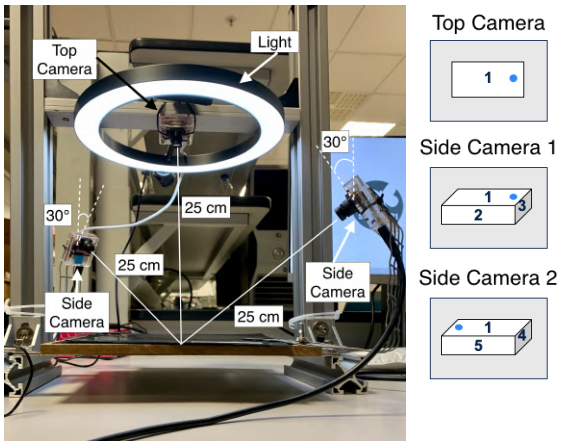


Fig. 1: Picture of the system's setup. The top camera is in the center of the ring light. Each camera is located at a distance of 25 cm from the working plane to ensure in-focus objects. Side cameras have an orientation angle of 30° with respect to the vertical axes.

processing algorithm, the dataset acquisition, the NN training, and the deployment in the OpenMV Cam H7 Plus. Section VI discusses the results acquired and the system characterization. Finally, Section VII closes this paper with some conclusions and future work.

II. Related Work

A. Edge computing

Edge devices with ML capabilities are recently gaining interest in designing intelligent IoT infrastructures that limit data exchange to a few bytes. AI edge processing focuses on moving the inference part of the AI workflow on the device by keeping data locally to improve latency and bandwidth [9]. The authors in [10] discuss the importance of bandwidth reduction. They present a deep-learning base algorithm for filtering surveillance videos, to transmit and store only the meaningful footage. Other applications – such as autonomous driving – have tight latency constraints, while others like voice/speech and face recognition need to consider users' privacy. Keeping AI processing on the edge device circumvents privacy concerns while avoiding bandwidth, latency, and cost issues [11].

B. Industrial quality inspection

The main challenges in using ML algorithms in industrial manufacturing environments are the limited processing capabilities, the use of big data, memory and energy constraints, and, sometimes, real-time processing [12]. Recent technologies for industrial visual inspection are based on line-scan cameras, spectrometers, or high-resolution cameras. These systems are expensive and require a significant amount of time to inspect one piece [13]. However, companies that want to offer high-quality products and optimize their production processes or costs need comprehensive and reliable quality inspection tools. In [14], a framework for the detection of glass bottle bottom defects is implemented using a combination of the visual attention model and wavelet transformation. The system achieves a recall of around 92%, requiring only 535 ms of computational time on a low-performance laptop CPU. The possible quality improvements in industrial processes are also highlighted in [15]. By exploiting a machine learning vision-based classification model, the authors highlight how the histogram-based droplet detection and micrograph classification approach can be exploited to determine when the emulsification process is completed automatically.

However, inspection systems are usually deployed in unreachable positions, making maintenance difficult. Thus, such systems must be standalone and used with the "deploy and forget" approach. Also, energy resources play a fundamental role in ensuring the system's reliability. For those reasons, research in machine learning has increasingly shifted to move data evaluation where it is generated, reducing and optimizing the used resources. In [12], the authors propose a quality inspection system that uses supervised ML algorithms in edge devices. This work supports manufacturing companies with a predictive model-based quality inspection system to predict the final product quality based on the recorded parameter of the process.

On the other hand, deep learning methods are data-hungry: they need a large amount of annotated data which is labor-intensive and time-consuming. As a workaround to these limitations, the authors in [16] have proposed a segmentation-aggregation framework to train object detectors from annotated visual data for automatic industrial visual inspection. They have limited the data annotation to label the image class and avoid the expensive task of the bounding box coordinate annotation. For this purpose, developing an accurate dataset for ML training is crucial to obtaining an accurate and efficient system.

C. Deep learning architecture and optimization techniques

In the last few years, fostered by new-generation microcontrollers, multiple neural network architectures were developed for resource-constrained devices [17]. Thanks to the study of innovative pruning and quantization techniques [18], it is possible to drastically reduce the ML model complexity while maintaining the same prediction accuracy. By combining those traditional techniques with more recent Neural Architecture Search (NAS) [19], and Federated Learning [20] approaches, it is possible to deploy AI-based systems in MCUs with impressive low energy consumption and high accuracy [21]. In the literature, we can already find multiple implementations, like MobileNetV2 [7] and SqueezeNet [8] and, more recently, new cutting-edge deep architecture, namely MobileNetV3 small [22], EfficientNet [23], and MCU Net V1 and V2 [21], [24]. In our proposed implementation, we have preferred to stick with the well-investigated MobileNetV2 and SqueezeNet as the preliminary results obtained satisfied the requirements of this application. We have thus preferred focusing on the deployment and long-term evaluation of the whole system in a real industrial large-scale test to assess the real performance

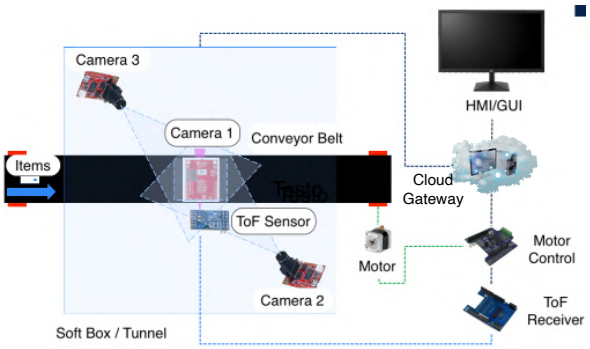


Fig. 2: System architecture. It consists of a conveyor belt, a ToF sensor that detects the presence of an object, three MCU-based cameras responsible for image processing and classification, and the cloud gateway which retrieves or rejects the piece according to the classification results.

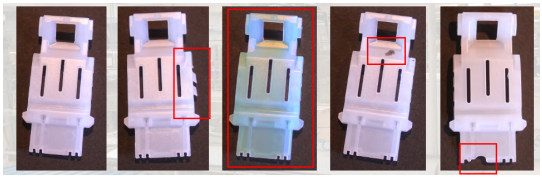


Fig. 3: Possible defects of the objects. From left: conforming object, deformed object, polluted object, object with stains, and incomplete object.

of the system. This has allowed us to optimize the selected network further and achieve the results presented in Section VI.

III. System Architecture

The architecture of the visual inspection system is shown in Figure 2. The workflow consists of positioning the produced items on a conveyor belt and moving them until they reach three cameras that acquire pictures of the items from different perspectives. A ToF (Time of Flight) sensor is placed on the side of the belt and alerts the cloud gateway that the object has arrived at the exact location. The gateway stops the belt, and the image classification phase begins. By using deep neural networks (DNN) and computer vision algorithms, the MCU on the camera classifies objects, like those in Figure 3, as conformant, with shape problems, or with color anomalies. Inferences and detection results are sent to the gateway, which controls and rejects the part if it does not meet the requirements. In detail, this process can be divided into four steps.

A. Item Upload

The first phase consists of sliding the object on the conveyor belt. The rotation of the object during the feeding has no constraints, so the components can assume a random rotation. The conveyor belt color was carefully chosen as ”matte black” to avoid unwanted light reflections that can generate image distortions and lead to wrong classifications.

B. Item Movement

The object slides on the conveyor belt, which is moved by a stepper motor positioned at the roller. The motor is controlled by the M4 core of the STM32MP1 MPU through the X-NUCLEO-IHM03M1 expansion board. Moreover, an X-NUCLEO-6180XA1 expansion board detects the presence of the items on the belt through a ToF sensor. The X-NUCLEO-6180XA1 also measures the ambient light, used to derive the light conditions when tagging and classifying items. Both expansion boards are connected to the cloud gateway.

C. Object Classification

Object classification is carried out with an edge computing approach by the arrangement of three OpenMV H7 Plus cameras, as shown in Figure 1. Two different models are used for the classification of shape and color anomalies. The position and the number of the cameras were chosen based on the types of defects to detect. Anomalies mainly occur only on 5 of the 6 faces of the object. This means that the field of view (FoV) of a single camera cannot cover all the object’s faces. The three cameras are placed in three different locations to ensure that all faces are within the cameras’ FoV. A ”Top camera” is positioned orthogonally to the belt and focuses on the 2D plane of the object. The other two cameras, called ”side cameras”, are placed in a specular position with an orientation angle of 30° with respect to the vertical axes. They are placed on either side of the object and expand the field of view on the remaining four faces. The distance of the lens from the object is 25 cm to ensure focus objects. Moreover, the 20 degrees FoV lens guarantees the inclusion of an item within a picture with some extra space around it to compensate for possible delays when stopping the belt. The three cameras take three different images of the component on the belt. These images are the input of DNNs deployed on the MCU responsible for classifying shape and color imperfections. The ”shape-defect” anomalies occur on the perimeter of the object. It follows that the top camera is best suited to select this type of nonconformity. The side cameras are used to detect color-defected objects.

D. Post-processing

The cloud gateway, according to the value obtained from the three cameras, enables the conveyor belt motor when the objects are conformant. On the contrary, if the result of the prediction is a non-conforming object, it proceeds to eliminate the component by activating a plunger.

IV. Tiny Neural Networks

A. Network Architecture and Hardware Requirements

Most DNN architectures require high computational capacity, focusing the deployment on specific high-performance computational units. In this application, the available resources are limited because the target board is an MCU. This lead to a challenge in researching and optimizing DNNs. Two of the best-performing DNNs specifically designed for embedded systems are chosen, namely MobileNetV2 [7] and SqueezeNet [8].

TABLE I: Topology of MobileNetV2. "n" denotes the replicas of the same layer, "c" the number of output channels, "s" the stride, and "t" the expansion factor [7].

Input	Operator	t	c	n	s
$224 \times 224 \times 3$	$conv2d$	-	32	1	2
$112 \times 112 \times 32$	$bottleneck$	1	16	1	1
$112 \times 112 \times 16$	$bottleneck$	6	24	2	2
$56 \times 56 \times 24$	$bottleneck$	6	32	3	2
$28 \times 28 \times 32$	$bottleneck$	6	64	4	2
$14 \times 14 \times 64$	$bottleneck$	6	96	3	1
$14 \times 14 \times 96$	$bottleneck$	6	160	3	2
$7 \times 7 \times 160$	$bottleneck$	6	320	1	1
$7 \times 7 \times 320$	$conv2d 1 \times 1$	-	1280	1	1
$7 \times 7 \times 1280$	$avgpool 7 \times 7$	-	-	1	-
$1 \times 1 \times 1280$	$conv2d 1 \times 1$	-	k	-	-

1) MobileNetV2: MobileNetV2 is an upgrade of the less recent MobileNet. The idea consists of using a new technique of convolution called depthwise separable convolution. It consists of replacing a full convolution operator with a factorized version that splits convolution into two separate layers: a depthwise convolution, and a pointwise convolution. MobileNetV2 also introduces two new types of calculations: the inverted residuals, and the linear bottleneck. They modify the network topology and obtain better performance and benchmarks. These operations compose the new block which includes three convolution layers. The first layer is a pointwise convolution that expands the low-dimensional feature map to a higher-dimensional space, followed by a depthwise convolution that implements spatial filtering of the higher-dimensional tensor with a ReLu activation function. Then, another pointwise convolution is added to the last feature map to lower the dimensional space. It is followed by a linear activation function to avoid the loss of information caused by the projection of the last convolution. The overall topology is defined in Table I [7].

2) SqueezeNet: SqueezeNet is an NN designed for systems with low computational capacity. The developers goal was to create a network that maintains competitive accuracy by reducing the number of parameters. These results are achieved with three strategies:

- Replacing 3x3 filters with 1x1 filters reduces the network size. This setting involves 9x fewer parameters and permits capturing correlations amongst near pixels.
- Reduce the number of inputs for the remaining 3x3 filters by feeding "squeeze" layers into "expand" layers as shown in Figure 4 (called "Fire Module").
- Downsample late in the network so that convolution layers have large activation maps, thus increasing the network accuracy.

"Squeeze" layers are convolution layers that are made up of only 1x1 filters; "expand" layers are convolution layers with a mix of 1x1 and 3x3 filters. SqueezeNet starts with a convolution layer, followed by 8 fire modules, and ends with a final convolution layer. The number of filters in the fire modules increases progressively throughout the network [8].

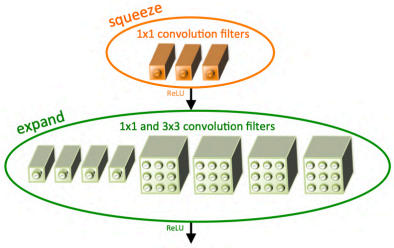


Fig. 4: SqueezeNet micro-architectural view of convolution filters and fire module [8].

B. Model Compression

Model compression is a fundamental step to deploy deep learning models in resource-constrained devices. In this application, three compression techniques were used to make possible the deployment of MobileNetV2 and SqueezeNet on the OpenMV Cam H7 plus. Even though two datasets are used to compose the visual inspection system for color and shape anomalies, they share the same model architecture and input image size; therefore, the number of parameters and the required resources are the same.

1) Parameter minimization: DNN's parameters are the sum of the weights and biases of each layer. Convolution or fully connected layers present a high number of parameters. Therefore, the number of parameters is proportional to the number of convolution layers and affects the final model dimensions. It is possible to reduce the number of layers to obtain a lighter model. However, this operation can reduce the overall accuracy, especially if the network becomes too shallow. Starting from this idea, MobileNetV2's blocks are reduced from 17 to 14, and the fire modules of SqueezeNet are reduced from 8 to 5. The effect of parameter optimization is presented in the last two rows of Table IV.

2) Pruning: after training the DNN models with the optimized architectures, pruning is applied to optimize the model complexity further. It permits cutting off weights irrelevant for prediction purposes (e.g., weights close to zero). We used the "Polynomial Decay" method to apply sparsity to the DNN. This method uses a range of sparsity values to mask weights, starting from those with less significant values and increasing them until the final sparsity is reached. In this case, both DNNs were pruned with a sparsity range of [20, 50] expressed as a percentage of removed weights, obtaining a model composed of 50% of the original parameters. This threshold was chosen to not lose accuracy. The result of the pruning operation is shown in Table II.

3) Quantization: quantization aims to reduce the NN model size by replacing the 32-bit float model with an 8-bit representation. This operation reduces size, storage, and memory

TABLE II: Number of parameters before pruning and number of non-zero parameters (NNZ) after pruning.

	Param. before pruning	Sparsity	NNZ after pruning
MobileNetV2	234 914	50%	117 457
SqueezeNet	120 930	50%	60 456

peak usage during inference. Quantization is essential to deploy DL models in MCU and improve hardware acceleration latency and power efficiency. A model quantized with 8-bit representation results in a model 4x smaller and 1.5x-4x faster in computations. In this paper, weights, activations, and network inputs and outputs were quantized. As a result, a "full-integer" model is obtained.

V. System Implementation

A. Image Pre-processing

The pre-processing algorithm analyzes the images in the camera MCU to highlight the relevant features useful for classification purposes. In this use case, the object arrives on a dark belt; thus, algorithms for background removal are used to avoid light reflection problems. Moreover, the object to classify may not be placed exactly in the center of the acquisition window. This means that clipping can cause the elimination of fundamental data for classification. For this reason, the developed algorithm considers the component's position within the image window and therefore avoids incorrect clipping. Three well-known computer vision algorithms are used: the "Canny algorithm", "Blob detection", and "Otsu's method". The pre-processing algorithm works as follows:

- Capture an image and create a copy.
- Conversion of the image from RGB565 to gray-scale.
- Canny algorithm to find the component contour.
- Search for blobs inside the Canny image.
- Blobs merging.
- Blob center computation.
- Check the position of the center to avoid wrong cropping.
- Image crop by taking as reference the obtained center.

The images are acquired by setting the sensor size to QVGA resolution (i.e., 320×240). After the Canny algorithm, we binary the image in the range [0,1] in which the pixels with value 1 are the contours of the plastic component. The Blob detection algorithm obtains different blobs of 1-pixel size referred to each pixel with value 1. The blobs fusion represents the four corners of the image, taking into account their outermost positions. Once a single blob is obtained, its center is calculated as a reference point for the crop to avoid clipping. Then, images are resized into 160×160 size, and the background is removed. The resulting images include only the object and straighten every other pixel regarding the background as a value of 0 in the RGB range. At this point, Otsu's method is used to find an optimal threshold to execute background removal.

B. Dataset Acquisition

The collection of a robust dataset is a key step in DL algorithm development. The three-camera arrangement uses two different DNN models, thus, it is necessary to collect two separate datasets: one for the top camera and one for the side cameras. The two datasets are composed of images pre-processed with the algorithm presented in Section V-A.

Figure 5 shows the results obtained by the pre-processing algorithms in the top camera and side cameras, respectively. In particular, Figure 5a is related to the pre-processing result from the top camera of a "conformant" component. Figure 5b shows a "color defected" component obtained after pre-processing. In this case, the background pixels and the stained part of the component have a value of 0 in the RGB channels. The rest of the image is in the RGB color space, where each pixel can take any value in the range [0,255]. Figure 5c shows a "conformant" component; therefore, despite the previous case, no pixel value about the component is set to 0, but only the background pixels.

(a) (b) (c)

Fig. 5: (a) pre-processing on the top camera showing a "conformant" component. (b) pre-processing done on the side camera showing a "color defected" component. (c) pre-processing done on the side camera showing a "conformant" component.

The pre-processed images are sent to the cloud gateway responsible for collecting the dataset. In this way, the camera does not keep images in memory but sends them to the gateway through the Remote Procedure Call (RPC) library embedded in Micro-Python. This allows the camera to connect to another device and execute remote procedure calls on the camera. The complete dataset was acquired with the setup shown in Figure 1. Each component was placed within the field of view of the three cameras, with different orientations and a minimum of random rotation to extend the heterogeneity of the dataset. Then, through a GUI, each image is tagged by selecting the class of the object.

More than 500 images were acquired for each camera in a balanced manner (i.e., each class includes the same number of images). Moreover, data augmentation was performed to increase the dataset size. Rotation and translation are used as image transformations for better generalizing the DNN's inputs. This operation is necessary to increase the number of images and replicate the real scenario where components can be placed in different positions and with minimal rotations in the camera field of view. The dataset is augmented by rotating the image in a range of [10, -10] degrees. However, this process is only done concerning the datum of the top camera because the two side cameras capture an image of the object laterally, therefore in a perspective way. A rotation of the image can cause distortion, thus, a wrong dataset optimization.

Given that the component can assume various positions, also image translation is used as image transformation. Considering that we are using square images with a rectangular object, the translation along the image width was set to 4% of the total (i.e., a random translation in the range [-9, 9] pixels). The translation in amplitude is 10% of the total (i.e., a random translation in the range [-22, 22] pixels along the height). The dataset size is summarized in Table III and is organized in

TABLE III: Overview of the dataset size for the top camera and the side cameras after data augmentation.

160x160px	Top Camera		Side Camera	
	Conformant	Shape Defected	Conformant	Color Defected
Train	3709	4068	2500	2409
Validation	1045	1145	691	672
Test	863		544	
Total	10830		6816	

TABLE IV: NN training parameters.

	Network Architecture	
	SqueezeNet	**MobileNetV2**
Batch size	32	32
Initial learning rate	10^{-5}	10^{-5}
Input image	RGB 160×160	RGB 160×160
Optimizer	Adam	Adam
Loss function	Binary cross-entropy	Binary cross-entropy
Source framework	Tensorflow	Tensorflow
♯ of parameters - base model	723 522	412 770
♯ of parameters - modified model	337 090	176 386

sub-folders to simplify the label extraction.

C. Training

The training of MobileNetV2 (alpha parameter equal to 0.35) and SqueezeNet is carried out with the parameters shown in Table IV. Model weights are initialized with random values. The number of epochs was set by using the *early stopping callback by validation accuracy* approach. This technique permits stopping the training session when the validation accuracy is below a certain threshold (i.e., 99.5%) to be less prone to network overfitting. The model evaluation is presented in Section VI-A.

VI. Results and Evaluation

A. Tiny Neural Network Evaluation

1) Top Camera: Figures 6a and 6b show the training results and validation accuracy of both architectures for the top camera dataset. To reach the desired validation accuracy value, MobileNetV2 and SqueezeNet need 25 and 38 epochs, respectively.

2) Side Cameras: Figures 6c and 6d show the results of the training session for the side camera dataset evaluated with MobileNetV2 and SqueezeNet, respectively. In this case, MobileNetV2 needs only 12 epochs to reach the desired validation accuracy, while SqueezeNet does not reach the set validation accuracy and uses the maximum number of epochs to complete the train (i.e., 100). Figure 6c shows that the validation loss value is low from the first epoch, which implies a rapid growth of the training accuracy. On the other hand, the plot in Figure 6d shows different negative peaks

TABLE V: Resources needed by the developed models to perform inference on OpenMV Cam H7 Plus.

160x160px	Float32 model (opt. parameters)		Optimized model (pruning, quantization)		Compress Factor
	Flash (KB)	RAM (KB)	Flash (KB)	RAM (KB)	
MobileNetV2	635.29	1490	172.26	381.22	3.8 ×
SqueezeNet	1290	1780	334.35	455.25	3.9 ×

TABLE VI: Comparison of MobileNetV2 and SqueezeNet performance for the top camera. ”Float 32” refers to the model with optimized parameters, while ”Optimized” refers to compressed models with pruning and quantization.

160x160px	MobileNetV2			SqueezeNet		
	Float32	Optimized	Δ	Float32	Optimized	Δ
Accuracy	99.5%	98.9%	0.6%	98.6%	98.4%	0.2%
Precision	99%	98%	1%	99%	99%	0
Recall	100%	100%	0	98%	98%	0
F-score	99.5%	99%	0.5%	98%	98%	0

in the validation accuracy due to the wrong weight update during backpropagation. In this case, parameter optimization has influenced the performance of SqueezeNet.

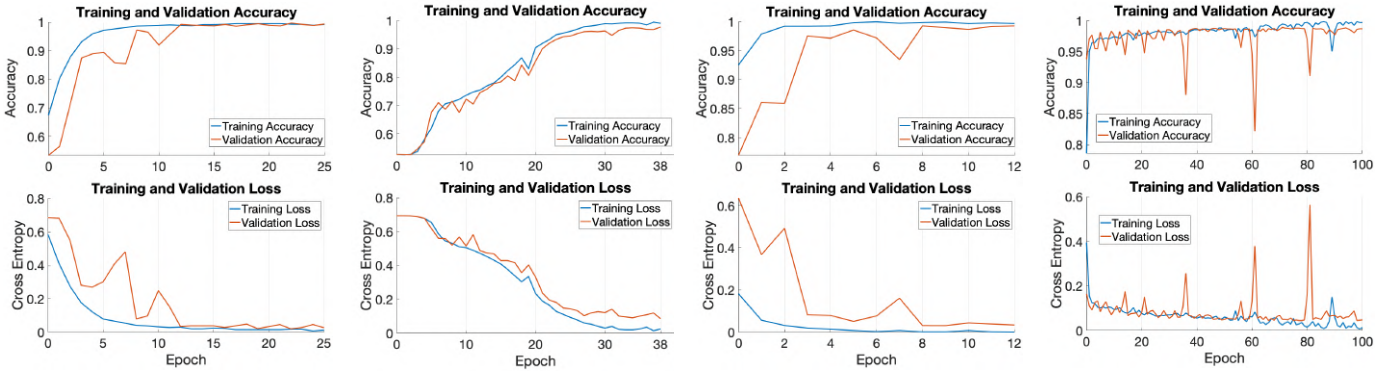
B. Top Camera Test

The top camera aims to classify ”conformant” and ”shape-defected” objects. MobileNetV2 and SqueezeNet are evaluated by considering accuracy, precision, recall, and f-score, as well as the loss in accuracy throughout the optimization operations. The test is conducted with 863 images, where 412 of them are ”conformant” and 451 are ”shape-defected” images. Their results are summarized in Table VI. Even though the models are highly compressed, the loss in performance is negligible.

Figures 7a and 7b show the Grad-CAM heatmap of MobileNetV2 and SqueezeNet, respectively. Grad-CAM is a tool to reveal zones where the network extracts features for classification. Here, the yellow circle highlights the shape anomalies (i.e., a missing pin), while the red circle highlights the region where most of the features are extracted by the deep learning model. In this case, the DL model extracts most of the features where the imperfection is located to produce the classification result. The MobileNetV2 heatmap in Figure 7a shows that only the image portion that includes the object, especially the right side, is used for feature extraction. This leads to a more generalized CNN, which processes almost the whole object to classify shape defects. On the other hand, the SqueezeNet heatmap in Figure 7b highlights only the image part related to the anomaly (upper right corner). It means that the model is specialized for this type of imperfection and cannot generalize as MobileNetV2 does. Furthermore, it is crucial to minimize false negatives (FNs) in an industrial visual inspection system to avoid missing detection of defects. As shown in Table VI, MobileNetV2 outperforms SqueezeNet achieving a recall of 100% (i.e., any FN is predicted during the test).

C. Side Cameras Test

Side cameras classify ”conformant” and ”color-defect” objects. The same evaluation for the top camera was conducted, but with 544 images. Table VIII summarizes the results. In this case, the comparison reveals good performance also with a different dataset, and the loss in accuracy due to the optimization process is negligible. Figures 7c and 7d show the Grad-CAM heatmap of a color-defected component of MobileNetV2 and SqueezeNet, respectively. Here, the yellow circle highlights the color anomalies (i.e., a stain), while the red circle highlights the region where the deep learning model extracts most of



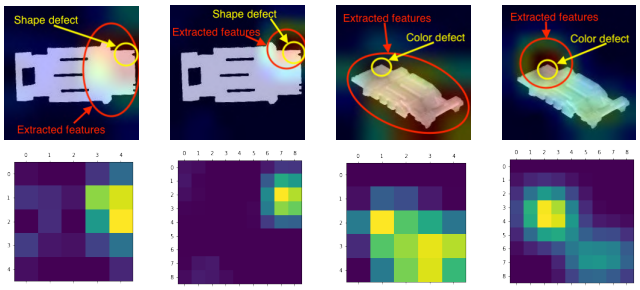
(a) Training and validation accuracy of the MobileNetV2 architecture for the top camera.

(b) Training and validation accuracy of the SqueezeNet architecture for the top camera.

(c) Training and validation accuracy of the MobileNetV2 architecture for the side camera.

(d) Training and validation accuracy of the SqueezeNet architecture for the side camera.

Fig. 6: Training and validation accuracy for both networks and cameras.



(a) (b) (c) (d)

Fig. 7: Grad-CAM heatmap of MobileNetV2 and SqueezeNet for the top camera ((a), and (b)), and side camera ((c), and (d)), respectively.

the features. It is visible that feature extraction involves the portion of the image where the defect is located. However, Figure 7d confirms the worsening of the performance of SqueezeNet. It highlights that most of the features useful for classification are extracted from the region in the upper-left corner, while features in the bottom-right corner are less useful for classification purposes. This phenomenon does not occur in the MobileNetV2 case (Figure 7c), where the whole region that covers the object is used to extract features to classify color anomalies. It means that MobileNetV2 can better generalize color anomalies than SqueezeNet. Furthermore, MobileNetV2 outperforms Squeezenet by analyzing the FNs, achieving a recall of 99% after the optimization process (i.e., only 1% of the samples were classified as FN during the test).

D. Inference on Cameras

The developed NN models were deployed in the setup shown in Figure 1. Their performance was evaluated by considering the execution time and the energy consumption of the two main tasks: pre-processing and classification. The OpenMV Cam H7 Plus consumes 0.8 W in active mode. The result is summarized in Table VII. The pre-processing time remains the same for both architectures because this task does not use DNN models. However, the top camera pre-processing time is slightly higher than side cameras because of the image binarization operation. MobileNetV2 outperforms SqueezeNet for both top camera and side cameras, considering the classification time and the energy consumption. As a result, MobileNetV2 needs only 240 ms and 233 ms for the top camera and the side camera, respectively, to process one image and classify the object. Consequently, it consumes 192 mJ and 186 mJ for the top camera and side camera processing, respectively. Moreover, the industrial molding machines, supported by the developed system, take about 20 seconds to produce two moles of 8 pieces. Thus, 240 ms to classify one piece is enough to guarantee the continuous and smooth operation of the production lines.

VII. Conclusions

High-accuracy product quality inspection is a fundamental step in any manufacturing process. It permits to boost in production yield and reduces production costs. By using machine learning techniques, developers can automate and make the process real-time. This paper presents the development and study of an innovative sensor system for automatically inspecting on-edge the quality of objects in large-scale production. The system exploits three smart cameras trained to detect and classify different anomalies in the components. Two different DNN models – namely the MobileNetV2 and the SqueezeNet – were trained and assessed, showing an accuracy of 99% and 98%, respectively. Thanks to the learning model optimization, the system can achieve respectively 5 FPS and 2 FPS for the two learning models while executing the evaluation on the edge of resource-constrained smart cameras. Future work will investigate new and innovative training techniques, like NAS, and enhance some features by integrating continuous learning functionalities to evolve the model automatically and extend the set of anomalies detectable by the system. Moreover, system's efficiency will be improved by using cameras to detect only relevant information directly from the imager, avoiding useless processing of uninteresting pixels.

TABLE VII: Execution time and energy consumption of the pre-processing and classification tasks for the top camera and the side cameras.

160x160px	**Top Camera** binarization=True				**Side Camera** binarization=False			
	Pre-processing (ms)	Classification (ms)	Total (ms)	Total Energy (mJ)	Pre-processing (ms)	Classification (ms)	Total (ms)	Total Energy (mJ)
MobileNetV2	125	115	240	**192**	118	115	233	**186**
SqueezeNet	125	390	515	412	118	390	508	406

TABLE VIII: Comparison of MobileNetV2 and SqueezeNet performance for the side cameras. "Float 32" refers to a model with optimized parameters, while "Optimized" refers to compressed models with pruning and quantization.

160x160px	**MobileNetV2**			**SqueezeNet**		
	Float32	Optimized	Δ	Float32	Optimized	Δ
Accuracy	100%	99.6%	0.6%	98.4%	98.2%	0.2%
Precision	100%	100%	0	100%	100%	0
Recall	100%	99%	1%	96%	96%	0
F-score	100%	99.5%	0.5%	98%	98%	0

REFERENCES

[1] H. Chopra, H. Singh, M. S. Bamrah, F. Mahbubani, A. Verma, N. Hooda, P. S. Rana, R. K. Singla, and A. K. Singh, "Efficient fruit grading system using spectrophotometry and machine learning approaches," *IEEE Sensors Journal*, vol. 21, no. 14, pp. 16 162–16 169, 2021.

[2] J. Tao, Y. Zhu, F. Jiang, H. Liu, and H. Liu, "Rolling surface defect inspection for drum-shaped rollers based on deep learning," *IEEE Sensors Journal*, vol. 22, no. 9, pp. 8693–8700, 2022.

[3] Z. Wang, J. Bai, X. Zhang, X. Qin, X. Tan, and Y. Zhao, "Base detection research of drilling robot system by using visual inspection," *Journal of Robotics*, vol. 2018, 2018.

[4] D. L. Dutta and S. Bharali, "Tinyml meets IoT: A comprehensive survey," *Internet of Things*, vol. 16, p. 100461, 2021.

[5] R. Sanchez-Iborra and A. F. Skarmeta, "Tinyml-enabled frugal smart objects: Challenges and opportunities," *IEEE Circuits and Systems Magazine*, vol. 20, no. 3, pp. 4–18, 2020.

[6] Y. Zou, M. Gottardi, M. Lecca, and M. Perenzoni, "A low-power vga vision sensor with embedded event detection for outdoor edge applications," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 11, pp. 3112–3121, 2020.

[7] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[8] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[9] L. Tsutsui da Silva, V. M. A. Souza, and G. E. A. P. A. Batista, "An open-source tool for classification models in resource-constrained hardware," *IEEE Sensors Journal*, vol. 22, no. 1, pp. 544–554, 2022.

[10] K. Muhammad, T. Hussain, J. Del Ser, V. Palade, and V. H. C. de Albuquerque, "Deepres: A deep learning-based video summarization strategy for resource-constrained industrial surveillance scenarios," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 5938–5947, 2020.

[11] A. Albanese, M. Nardello, and D. Brunelli, "Low-power deep learning edge computing platform for resource constrained lightweight compact uavs," *Sustainable Computing: Informatics and Systems*, vol. 34, p. 100725, 2022.

[12] J. Schmitt, J. Bönig, T. Borggräfe, G. Beitinger, and J. Deuse, "Predictive model-based quality inspection using machine learning and edge cloud computing," *Advanced engineering informatics*, vol. 45, p. 101101, 2020.

[13] M. A. Ali and A. K. Lun, "A cascading fuzzy logic with image processing algorithm–based defect detection for automatic visual inspection of industrial cylindrical object's surface," *The International Journal of Advanced Manufacturing Technology*, vol. 102, no. 1, pp. 81–94, 2019.

[14] X. Zhou, Y. Wang, Q. Zhu, J. Mao, C. Xiao, X. Lu, and H. Zhang, "A surface defect detection framework for glass bottle bottom using visual attention model and wavelet transform," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 4, pp. 2189–2201, 2020.

[15] S. Unnikrishnan, J. Donovan, R. Macpherson, and D. Tormey, "An integrated histogram-based vision and machine-learning classification model for industrial emulsion processing," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 5948–5955, 2020.

[16] C. Ge, J. Wang, J. Wang, Q. Qi, H. Sun, and J. Liao, "Towards automatic visual inspection: A weakly supervised learning method for industrial applicable object detection," *Computers in Industry*, vol. 121, p. 103232, 2020.

[17] M. G. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," *ACM Comput. Surv.*, vol. 54, no. 8, oct 2021.

[18] R. Mishra, H. P. Gupta, and T. Dutta, "A survey on deep neural network compression: Challenges, overview, and solutions," *arXiv preprint arXiv:2010.03954*, 2020.

[19] P. Ren, Y. Xiao, X. Chang, P.-y. Huang, Z. Li, X. Chen, and X. Wang, "A comprehensive survey of neural architecture search: Challenges and solutions," *ACM Comput. Surv.*, vol. 54, no. 4, may 2021.

[20] A. Imteaj, U. Thakker, S. Wang, J. Li, and M. H. Amini, "A survey on federated learning for resource-constrained iot devices," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 1–24, 2022.

[21] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "Mcunet: Tiny deep learning on iot devices," *arXiv preprint arXiv:2007.10319*, 2020.

[22] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1314–1324.

[23] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[24] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, "Mcunetv2: Memory-efficient patch-based inference for tiny deep learning," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021.