Defect Detection in Printed Circuit Boards Based on EdgeML and Computer Vision

Felipe G. F. Rocha[1], Hyago V. L. B. Silva[1], Rodrigo B. Vimieiro[2], and Felipe A. P. Figueiredo[1]
[1]*Wireless and Artificial Intelligence Laboratory, WAI Lab. - Inatel - Santa Rita do Sapucaí, Brazil*
[2]*São Carlos School of Engineering of the University of São Paulo - São Carlos, Brazil*
email: [feliperocha, hyago.silva]@mtel.inatel.br, rodrigo.vimieiro@usp.br, felipe.figueiredo@inatel.br

***Abstract*—The production of electronic boards is a common activity in the industrial environment, and ensuring their quality is essential for obtaining reliable products. This work presents a comparative study of the performance of three machine learning architectures: YoloV8, FOMO, and MobileNet. The dataset was created based on four classes for assembly fault detection. The model that achieved the best results was FOMO, with precision, recall, and F1-Score above 95%, as well as processing 10.5 frames per second and having a total size of 152 kB.**

***Index Terms*—Quality Control, PCB, Machine Learning, Deep Learning.**

I. Introduction

In the industrial environment, producing high-quality Printed Circuit Boards (PCBs) is essential to ensure that a reliable product reaches the end customer [1]. The quality control department aims to ensure and enforce compliance at each stage of the industrial process, in accordance with pre-established standards. The department is responsible for conducting functionality tests and visual inspections of products by sampling, a frequently manual task that relies on the employee's focus and interpretation. This can lead to human errors or undetected defects that fall outside the sampling [2]. The integration of Industry 4.0 technologies, such as the Internet of Things (IoT), artificial intelligence (AI), and cloud computing, plays an important role in optimizing and ensuring reliability in processes [3]. The technological capability of machine learning models to process and analyze large volumes of data and recognize patterns makes it possible to distinguish between defective and non-defective PCBs accurately, detect an unmounted or incorrectly mounted component, or even identify defects in traces, such as open circuits or short circuits. This technology makes sample-based inspections unnecessary, as every produced board can be individually analyzed. This work aims to investigate different convolutional neural network architectures to characterize assembly defects in PCBs within an industrial process.

This work was partially funded by Intelbras (Pró-Educar Program), by CNPq (Grant Nos. 403612/2020-9, 311470/2021-1, and 403827/2021-3), by Minas Gerais Research Foundation (FAPEMIG) (Grant Nos. APQ-00810-21, APQ-03162-24, and PPE-00124-23), and by the projects XGM-AFCCT-2024-2-5-1, and XGM-AFCCT-2024-9-1-1 supported by xGMobile – EMBRAPII-Inatel Competence Center on 5G and 6G Networks, with financial resources from the PPI IoT/Manufatura 4.0 from MCTI grant number 052/2023, signed with EMBRAPII.

II. Related Works

In Githu's work [4], machine learning (ML) concepts are used to identify manufacturing defects in printed circuit boards. In contrast, this article specifically focuses on identifying faults during the assembly of electronic components, a distinct area of application. Githu's project used a Raspberry Pi 4 and an 8-megapixel camera module to run three models: MobileNetV2 SSD FPN-Lite 320x320 [5], Edge Impulse's FOMO [6], and YOLOv5 [7], with FOMO being chosen for its superior performance in detecting short circuits, open circuits, and missing holes. Differently, our study considers YOLOv8, a more advanced model, along with FOMO e MobileNetv2 SSD FPN-Lite 320x320. Additionally, we apply weight quantization to the studied models and assess its impact on their performance, something not explored in Githu's work. Quantization allows optimization of both accuracy and performance on more limited hardware. Moreover, a significant contribution of our work is the creation of a specific dataset for detecting assembly faults in PCBs.

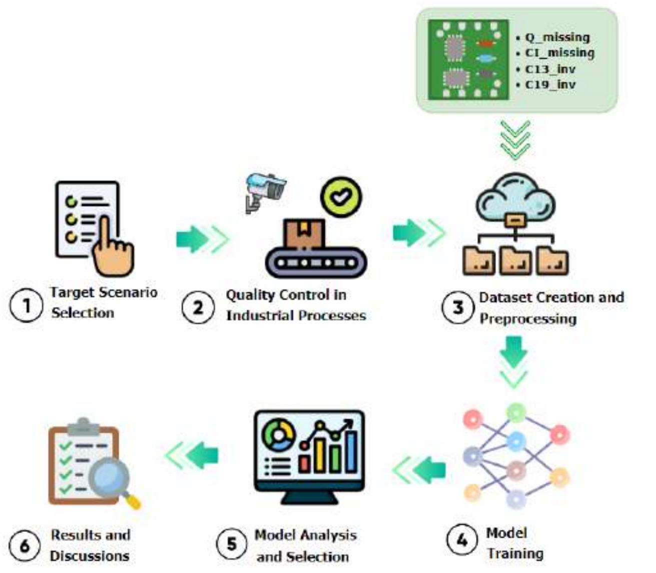
Nguyen and Bui's work [8] presents a study on PCB defect inspection using deep learning techniques. Aiming to propose an automated real-time supervision algorithm, the authors use an enhanced ResNet-50 Convolutional Neural Network (CNN) model that achieves an average accuracy rate of up to 96.29% under good lighting and brightness conditions. The authors do not consider the use of low-cost, low-power devices, and the hardware used included a 5-megapixel webcam for image capture, an NVIDIA Jetson Nano B01 development kit for processing the proposed model, an Arduino Mega board, motor driver modules, DC motors, and optical sensors for conveyor belt control.

III. Methodology

Figure 1 illustrates the general methodology used in this work. First, the target scenario for the ML model application was determined. For this study, the industrial environment was chosen. Second, the application was selected: quality control of PCBs. Third, the dataset was determined. Finally, different network architectures were investigated, and the results of each were analyzed for the proposed problem.

A. Training dataset

Figure 2 illustrates the process of constructing the custom dataset developed for this work. Initially, it was necessary



• Q_missing
• CI_missing
• C13_inv
• C19_inv
1 Target Scenario Selection
2 Quality Control in Industrial Processes
3 Dataset Creation and Preprocessing
4 Model Training
5 Model Analysis and Selection
6 Results and Discussions

Fig. 1: General methodology used in this work. A step-by-step approach is detailed up to the final result.

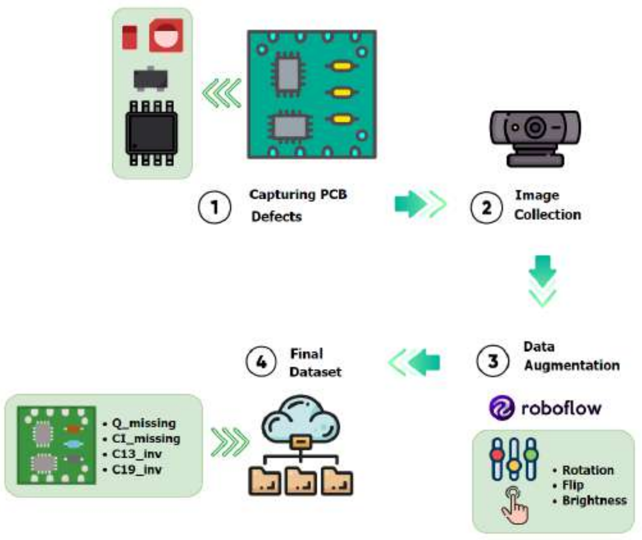


Fig. 2: Dataset construction process divided into four steps.

to produce defective PCBs. A 3-megapixel resolution webcam was used. The mapped defects included unmounted and incorrectly mounted electronic components. Four classes were defined: Q missing (unmounted transistor), CI missing (unmounted ICs), C13 inv (incorrectly mounted C13 capacitor), and C19 inv (incorrectly mounted C19 capacitor). A total of 240 images were collected. Some data augmentation techniques were employed to increase sample diversity [9], including image rotation, flipping, and brightness adjustments. After these procedures, the dataset was expanded to 576 images, distributed as 504 for the training set, 48 for the validation set, and 24 for the test set. In the end, each class contained 240 examples. This exclusive dataset, created specifically for the detection of assembly faults in electronic components, is publicly available at [10].

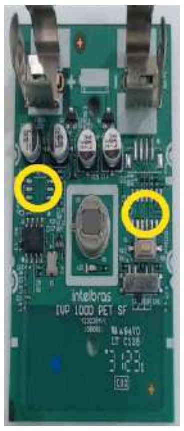


Fig. 3: Label of the CI-Missing class, highlighted by the yellow circle. This class indicates a missing IC assembly.

Figure 3 illustrates a label of the CI missing class, marked by the two yellow circles.

B. Employed Deep Learning Architectures

Different deep convolutional neural network architectures were evaluated. The architectures are detailed below.

The first was YOLOv8 (You Only Look Once) [7]. YOLO is a widely used model for object detection and image segmentation due to its high speed and accuracy. In the proposed work, training was conducted over 200 epochs.

Next, the FOMO (Faster Objects, More Objects) model [6] was tested. Developed by the engineers at Edge Impulse [11], the FOMO algorithm is used for real-time multiple object detection. The model uses MobileNetV2 as the base for its structure and, by default, performs a spatial reduction of 1/8 from input to output, cutting the MobileNet model at the intermediate layer 6 ('block-6-expand-relu'). This architecture was trained on the Edge Impulse platform with configurations including 200 training epochs, a learning rate of 0.001, and a batch size of 32.

Finally, the MobileNetV2 network [5] was evaluated. The MobileNetV2 convolutional neural network architecture was designed to be efficient regarding speed and computational resource usage. For training this model, 30,000 epochs and a batch size of 32 samples were used.

For all the mentioned architectures, the concept of transfer learning was applied—a machine learning technique in which a model developed for a specific task is reused as a starting point for a model on a new task [12].

C. Proposed Application Architecture

The architecture of the proposed application can be seen in Figure 4. It consists of a Logitech 1080p camera, which was used to capture the images for the dataset and for deploying the model, a Raspberry Pi model 4B, Cortex-A72, with 4 GB of RAM and 32 GB of Flash memory. Notably, the dashboard illustrated in the figure represents an enhancement planned for future work and is not part of the current implementation.

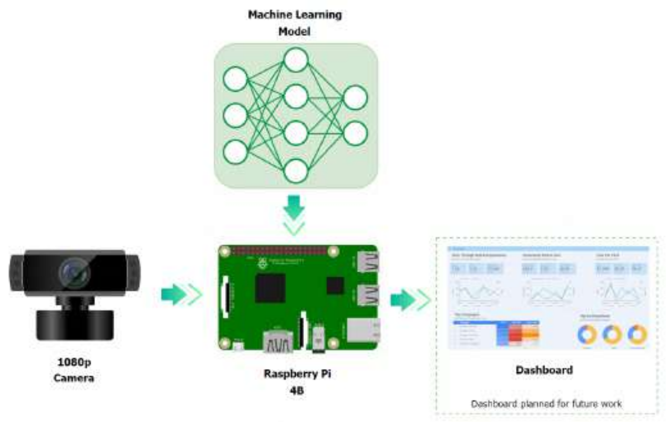


Fig. 4: Proposed application architecture

IV. Metrics

Different metrics were applied to evaluate the performance of the neural networks.

Precision quantifies the proportion of correct predictions in relation to the total cases classified as positive. The formula that defines it is as follows

$$\text{Precision} = \frac{TP}{TP + FP}, \tag{1}$$

where TP stands for true positive and FP for false positive.

The recall metric measures the proportion of positive examples that the model correctly identified in relation to all examples that are actually positive and can be calculated using the following formula

$$\text{Recall} = \frac{TP}{TP + FN}, \tag{2}$$

where FN stands for false negative.

The F1-Score metric is the harmonic mean between precision and recall, determined by the following formula

$$\text{F1-Score} = 2 * \frac{(\text{Precision} * \text{Recall})}{(\text{Precision} + \text{Recall})}. \tag{3}$$

The Mean Average Precision (mAP) parameter is obtained by calculating each object class's average of the AP (Average Precision) metrics. The AP of a specific class represents the average of correct detections the model returns up to a certain point in the detection list [13]. The formula for mAP is as follows:

$$\text{mAP} = \frac{1}{N}\sum_{i=1}^{N} \text{AP}_i, \tag{4}$$

where N is the total number of classes, and AP_i is the Average Precision of the i-th class.

V. Results

In this section, the models' results are presented in terms of performance metrics, such as precision, recall, F1-Score, and mAP, and computational performance, including total model size, CPU usage, RAM usage, and inference time. A Raspberry Pi 4B, equipped with a Cortex-A72 processor, 4 GB of RAM, and 32 GB of Flash memory, was used to assess the computational characteristics.

TABLE I: YoloV8 results

	Size	CPU	RAM	FPS	Prec.	Recall	F1	mAP
Float 32	22 MB	30%	670 MB	0,2	96,8%	92,7%	94,6%	96,1%
Int 8	11 MB	20%	550 MB	0,3	94,3%	94,8%	94,5%	95,6%

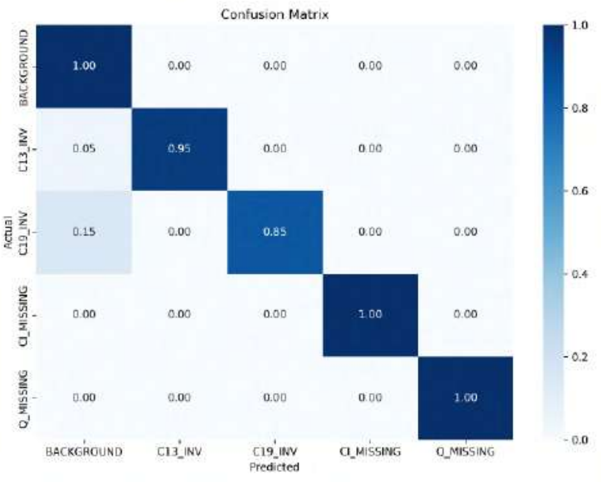


Fig. 5: Confusion Matrix - YoloV8

As shown in Table I, the YOLOv8n float32 model performed well after training, achieving a precision of 96.8%, recall of 92.7%, F1-Score of 94.6%, and mAP of 96.1%. Despite the high metrics, the computational performance of the model was not favorable. The model had a total size of 22 MB, required 670 MB of RAM, utilized 30% of the CPU, and achieved an inference rate of 0.2 frames per second.

Figure 5 presents the model's confusion matrix, describing its classification performance on the test dataset. Notably, there was higher accuracy in the CI-Missing and Q-Missing classes, followed by C13-INV and C19-INV.

The weight quantization technique from float32 to int8 (8 bits) was used to reduce computational demands. As a result, the model size was reduced to 11 MB, RAM requirement decreased to 550 MB, and CPU usage dropped to 20%. Processing time improved to 0.3 frames per second. The performance metrics for the quantized model were: precision of 94.3%, recall of 94.8%, F1-Score of 94.5%, and mAP of 95.6%.

The FOMO float32 model achieved the following perfor-

TABLE II: FOMO results

	Size	CPU	RAM	FPS	Prec.	Recall	F1	mAP
Float 32	152 kB	25%	250 MB	10	97%	99%	98%	99%
Int 8	91 kb	20%	230 MB	20	97%	99%	98%	99%

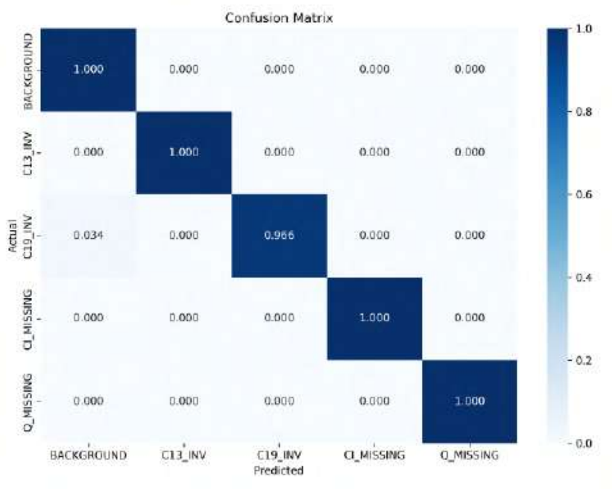


Fig. 6: Confusion Matrix - FOMO

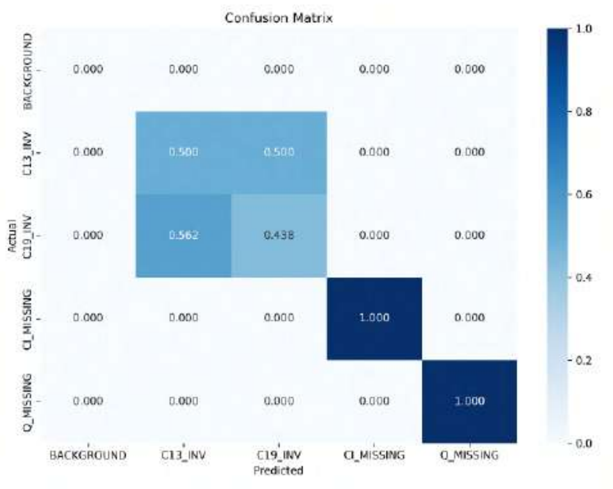


Fig. 7: Confusion Matrix - MobileNetV2

TABLE III: MobileNetV2 results

	Size	CPU	RAM	FPS	Prec.	Recall	F1	mAP
Float 32	11 MB	30%	350 MB	3	61%	87,5%	71,8%	100%
Int 8	3 MB	25%	300 MB	6	55%	79%	65%	65%

mance metrics: 97% precision, 99% recall, 98% F1-Score, and 99% mAP, as shown in Table II. Additionally, the model has a size of 152 kB, requires 250 MB of RAM, uses 25% of the CPU, and has an inference time of 95 milliseconds per image, equivalent to approximately 10 frames per second. The confusion matrix for this model is shown in Figure 6, where it can be observed that the highest accuracies were for the classes C13-INV and CI-Missing, followed by C19-INV and Q-Missing.

With the quantized int8 model, the performance metrics remained unchanged. In contrast, the model size decreased to 91 kB, RAM requirement dropped to 230 MB, CPU usage reduced to 20%, and inference time lowered to 50 milliseconds, resulting in 20 frames per second.

According to the data in Table III, the performance evaluation metrics for the MobileNetV2 float32 model included a precision of 61%, recall of 87.5%, F1-Score of 71.8%, and an mAP of 100%. The model has a size of 11 MB, requires 350 MB of RAM, uses 30% of the CPU, and achieves an inference capacity of 3 frames per second. The model's confusion matrix is presented in Figure 7.

The quantized int8 model's total size was reduced to 3 MB. RAM requirement and CPU usage decreased to 300 MB and 25%, respectively. Inference capacity increased to 6 frames per second.

The precision, recall, F1-Score, and mAP metrics were 55%, 79%, 65%, and 65%, respectively.

Comparing these results with those presented in reference [4], it is evident that the performances differ. The main reason for this difference lies in the variation of datasets used. While Githu's work focuses on detecting specific defects in PCBs, our dataset was specifically designed to identify assembly faults in electronic components, resulting in different detection challenges. Additionally, the image capture conditions and the complexity of the defects may have influenced the differences in results.

VI. Discussions

As detailed in the previous section, the FOMO and YOLOV8 models demonstrated the best performances. However, FOMO stood out, with precision, recall, F1-Score, and mAP metrics exceeding 97%. In contrast, the metrics of the MobileNet model were notably lower.

The FOMO model also excelled due to its significantly lower computational requirements. Its compact size and low RAM and CPU demands make it ideal for low-capacity devices. The model's inference rate was also the fastest among the three options tested, with the quantized model reaching 20 frames per second.

Structured with a convolutional architecture, the FOMO model efficiently applies convolution operations across the entire input data. In this context, convolution transforms the input pixel matrix into a new matrix, enhancing essential patterns and features.

The model's architecture is based on MobileNetV2, with an input layer containing 1,080,000 features. The best performance was achieved when using the 8 layers of the base model, i.e., cutting the base model at intermediate layer 8.

After this cutoff, a new 2D convolutional layer was introduced. This layer consists of 32 filters, where each filter is a small matrix used to process specific regions of the input. The term 'kernel' refers to the size of this filter matrix; in this case, it is size 1, indicating a 1x1 convolution window. The ReLU activation function is applied after the convolution operation, meaning negative values are transformed into zero while positive values remain unchanged, helping introduce non-linearities into the neural network. The final step includes an additional 2D convolutional layer composed of 4 filters and a kernel of size 1. The purpose of this layer is to provide the final class to be detected.

VII. Conclusion

The proposed work investigated three machine-learning architecture options for quality control in assembling electronic boards in an industrial environment. The model that achieved the best performance was FOMO. However, further testing with different models is still necessary to achieve even better results. Notably, the proposed methodology is not dependent on a specific model, meaning that any architecture can be used for object detection.

The pursuit of reduced computational resource consumption expands the possibilities for application on low-capacity devices, making the model more accessible. In this way, finding sustainable solutions enabling process automation in small businesses with limited resources contributes to operational efficiency and promotes more economical and sustainable practices.

Expanding the available dataset is essential to improving the model's effectiveness and robustness. Furthermore, conducting tests in real industrial environments is crucial to validate the model's performance consistently. Due to the current scarcity of available images, expanding the training set should be a priority for future work. Additionally, developing a dashboard capable of integrating with the Raspberry Pi processor is necessary, allowing for the visualization of all classes detected by the model in a connected and interactive manner.

References

[1] J. Shen, N. Liu, and H. Sun, "Defect detection of printed circuit board based on lightweight deep convolution network," *IET Image Processing*, vol. 14, no. 15, pp. 3932–3940, 2020.

[2] T. Khare, V. Bahel, and A. C. Phadke, "Pcb-fire: Automated classification and fault detection in pcb," in *2020 Third International Conference on Multimedia Processing, Communication & Information Technology (MPCIT)*. IEEE, 2020, pp. 123–128.

[3] L. S. Dalenogare, G. B. Benitez, N. F. Ayala, and A. G. Frank, "The expected contribution of industry 4.0 technologies for industrial performance," *International Journal of production economics*, vol. 204, pp. 383–394, 2018.

[4] S. Githu. Pcb defect detection with computer vision - raspberry pi. [Online]. Available: https://docs.edgeimpulse.com/experts/image-projects/pcb-defect-detection-with-computer-vision-raspberry-pi

[5] 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.

[6] E. Impulse. (2023, May) FOMO: Object detection for constrained devices. [Online]. Available: https://docs.edgeimpulse.com/docs/edge-impulse-studio/learning-blocks/object-detection/fomo-object-detection-for-constrained-devices

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[8] V.-T. Nguyen and H.-A. Bui, "A real-time defect detection in printed circuit boards applying deep learning," *EUREKA: Physics and Engineering,(2)*, pp. 143–153, 2022.

[9] S. Alexandrova, Z. Tatlock, and M. Cakmak, "Roboflow: A flow-based visual programming language for mobile manipulation tasks," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 5537–5544.

[10] RoboFlow, "Componentes com defeitos dataset," 2024, acessado em: 01 set. 2024. [Online]. Available: https://universe.roboflow.com/masters-5bgc5/componentes-com-defeitos

[11] S. Hymel, C. Banbury, D. Situnayake, A. Elium, C. Ward, M. Kelcey, M. Baaijens, M. Majchrzycki, J. Plunkett, D. Tischler, A. Grande, L. Moreau, D. Maslov, A. Beavis, J. Jongboom, and V. J. Reddi, "Edge impulse: An mlops platform for tiny machine learning," 2023.

[12] N. Agarwal, A. Sondhi, K. Chopra, and G. Singh, "Transfer learning: Survey and classification," in *Smart Innovations in Communication and Computational Sciences*, S. Tiwari, M. C. Trivedi, K. K. Mishra, A. Misra, K. K. Kumar, and E. Suryani, Eds. Singapore: Springer Singapore, 2021, pp. 145–155.

[13] S. Robertson, "A new interpretation of average precision," in *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 689–690. [Online]. Available: https://doi.org/10.1145/1390334.1390453