FEDERAL UNIVERSITY OF ITAJUBÁ – UNIFEI

Institute of Systems Engineering and Information Technology – IESTI
Computer Engineering Program

TRAFFIC SIGN RECOGNITION

Rodrigo Zaparoli Silveira – 2023003016

Itajubá – MG
2025

TABLE OF CONTENTS

1. INTRODUCTION

This project proposes the development of a low-cost embedded system for detecting the presence of traffic signs in images, using embedded machine learning techniques (**TinyML**). The goal is to indicate whether or not a traffic sign is present in the captured image.

For this, an **Arduino Nano 33 BLE Sense** microcontroller is used, which supports machine learning models, along with a **CMOS OV7675** camera. The goal is to perform inference in real-time, directly on the microcontroller.

2. OBJECTIVES

Develop an embedded system based on TinyML, using the Arduino Nano 33 BLE Sense, capable of detecting whether a traffic sign is present in a real-time image captured by the integrated camera.

For this Project the framework **Edge Impulse** was used throughout all development.

3. PROJECT DESCRIPTION

The system works by capturing images through the onboard camera and, after a pre-processing step, applying a pre-trained machine learning model to classify the image as “sign” or “background.”

To simplify the architecture and enable execution on the microcontroller, object localization techniques like bounding boxes were not used. While common in object detection tasks, such techniques require more processing power and RAM—resources not available on the Arduino Nano 33 BLE Sense.

Thus, the system performs binary classification of the entire image, significantly reducing model complexity, speeding up inference, and improving overall performance.

The dataset used was obtained from public repositories on Kaggle and adapted to include both images with and without traffic signs. Some images were adjusted to balance class proportions and avoid training bias.

4. METHODOLOGY

4.1 Hardware Used

- Arduino Nano 33 BLE Sense
- CMOS OV7675 Camera

4.2 Data Collection

A public dataset available on Kaggle was used, complemented with images that do not contain traffic signs to balance the “with sign” and “without sign” classes. Problematic images were manually removed or adjusted to ensure better quality and representativeness.

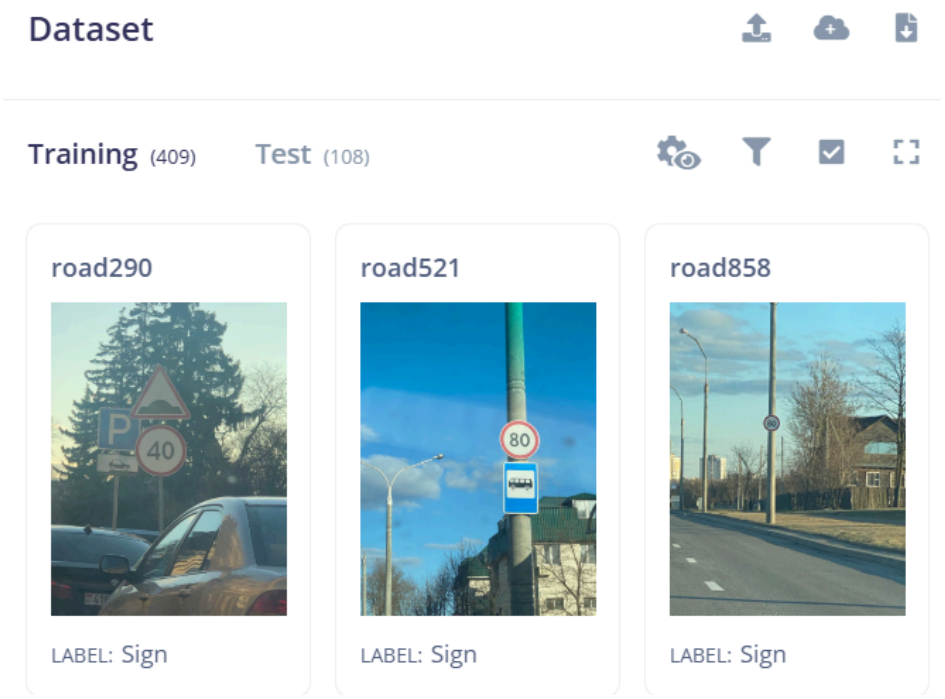

Image 1 – Dataset examples.

4.3 Pre-processing

Images went through the following pipeline:

- Resizing to 96x96 pixels
- Conversion to grayscale
- Pixel normalization to the range [0,1]

This process aims to standardize the data and reduce input complexity for the model.

Image 2 – Result of image pre-processing

4.4 Model Architecture

- **Architecture**: Convolutional Neural Network (CNN)
- **Frameworks**: TensorFlow, Edge Impulse
- **Technique**: Transfer Learning

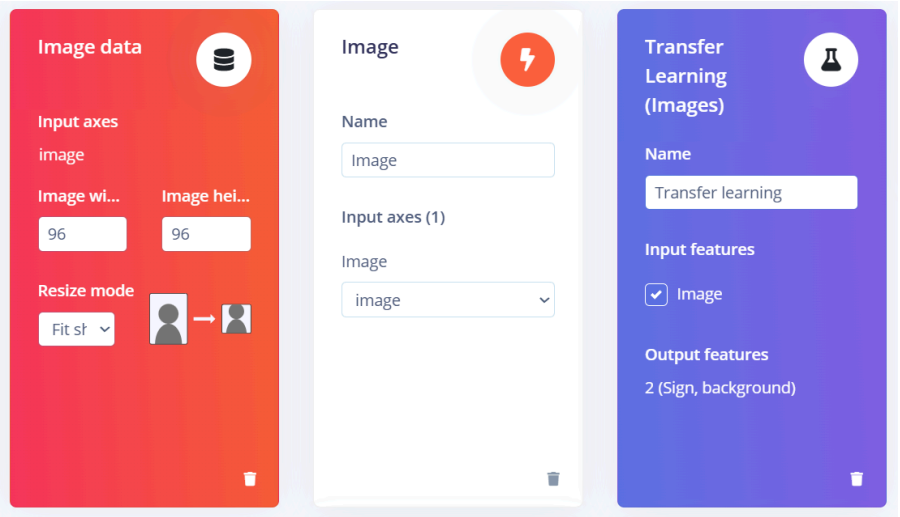

Image 3 – Structure of the final model used.

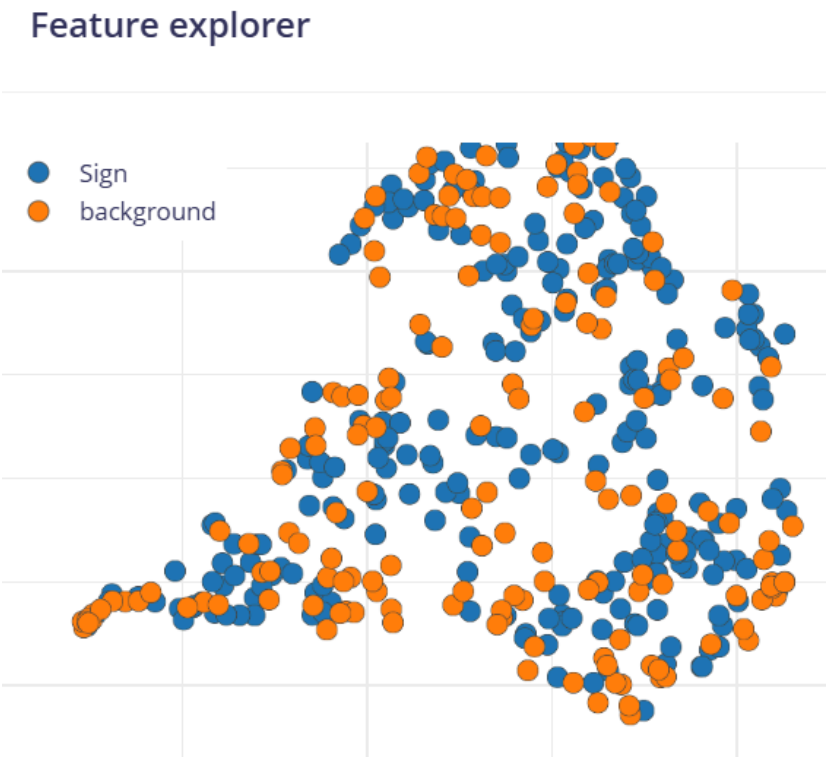

Image 4 – Visualization of extracted features.

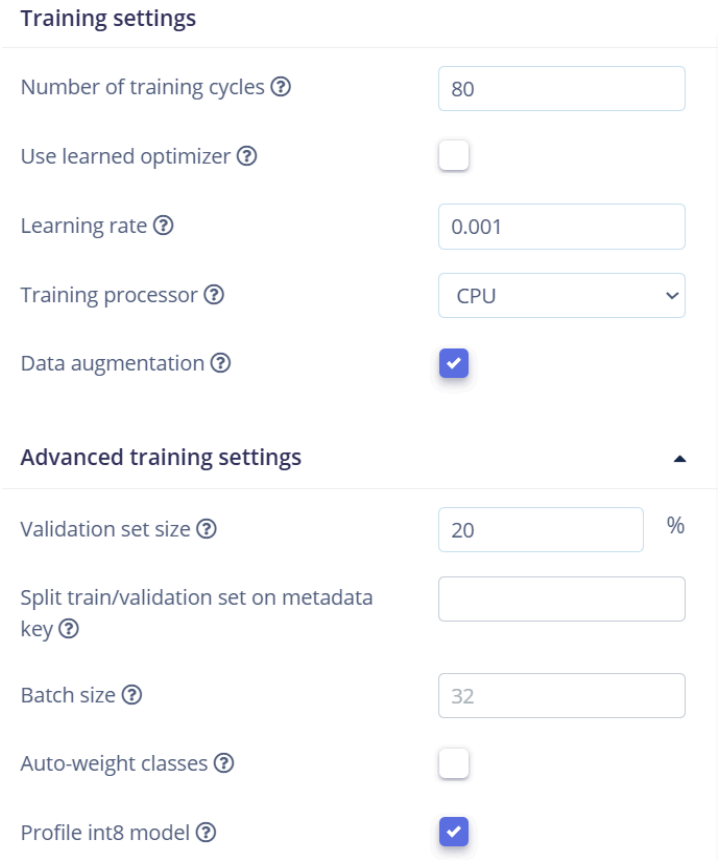

Image 5 – Training configuration.

4.5 Model Results and Accuracy

During pre-processing, resizing is a fundamental step. However, it presents an important trade-off between performance and accuracy. In simple scenarios, where there is only one well-centered sign with good lighting, the model performs well and reliably detects the presence of a sign.

In more complex situations, downscaling can significantly compromise the quality of visual information. This includes:

- Presence of multiple signs in a single image, which are merged or lost in the low-resolution representation;
- Variations in capture angle, distorting the sign's shape and hindering recognition by simple CNNs;
- Adverse lighting conditions, like shadows or glare, which lose contrast and details after grayscale conversion and resolution reduction.

Even with these limitations, the model proved functional for the general purpose of the project, fulfilling its goal of detecting (presence or absence of a sign) with acceptable performance in most tests. For more demanding applications, higher resolutions and more robust architectures would be needed, requiring more powerful hardware.

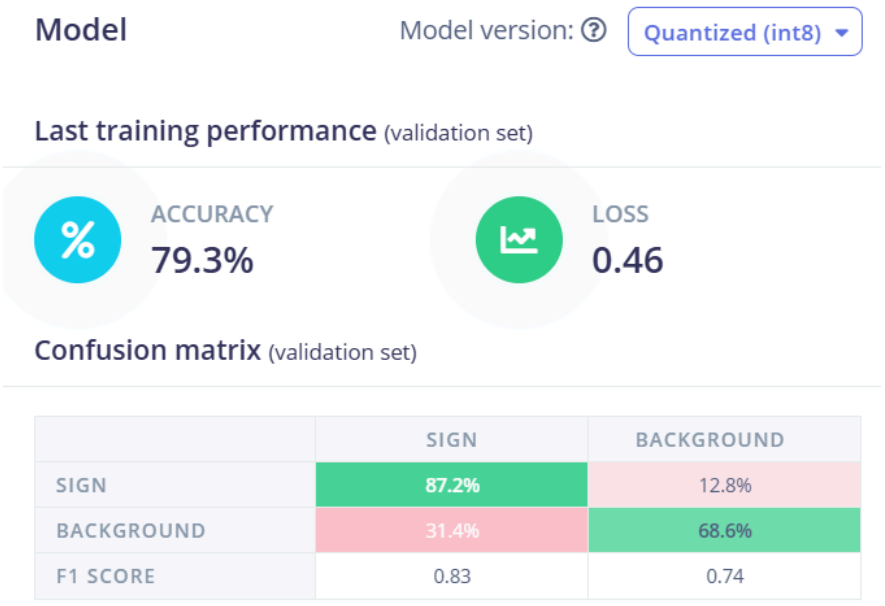

Image 6 – Model accuracy.

5. INFERENCE

The inference phase was performed through simulation, using the board's camera to capture images from a computer screen displaying different traffic signs. This approach creates poor conditions for the model, and framing proved to be an important factor during inference. Even so, the model was able to correctly identify the presence of signs in most displayed images.

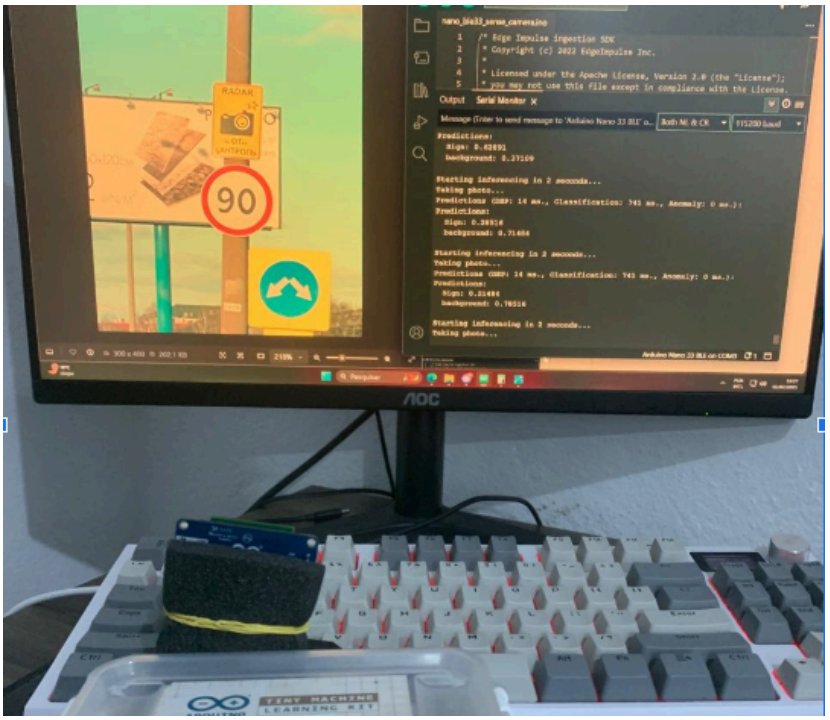

Image 7 – Inference method.

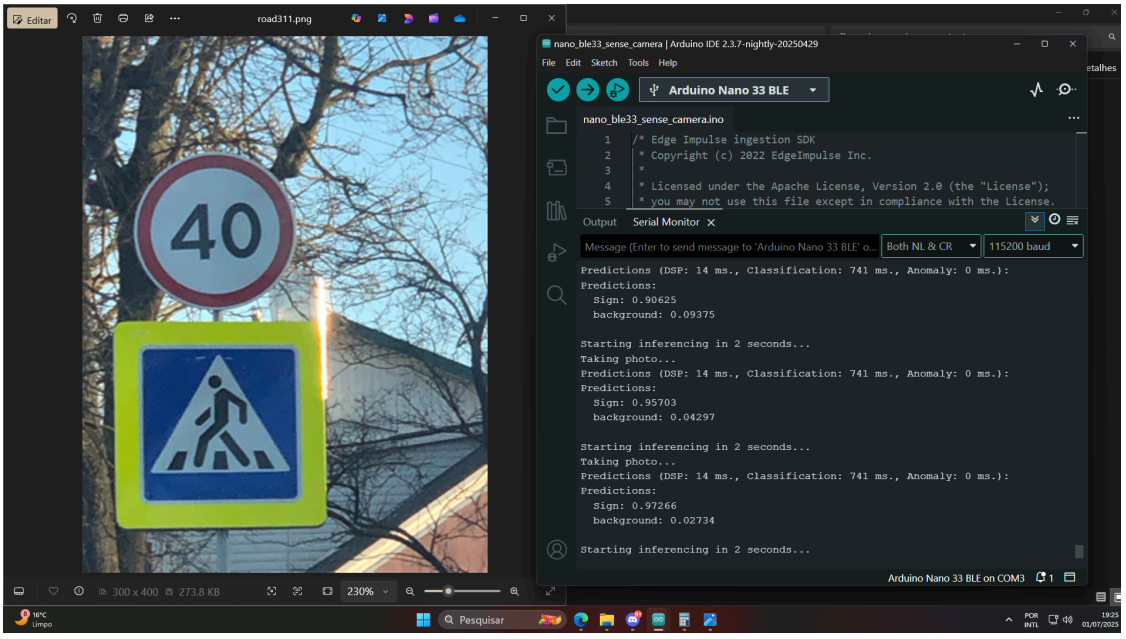

Image 8 – Inference in Arduino IDE.

6. REFERENCES

KAGGLE. Traffic Signs Image Classification 97% [CNN]. Available at: https://www.kaggle.com/. Accessed: June 30, 2025.

GITHUB. aarcosg/traffic-sign-detection. Available at: https://github.com/aarcosg/traffic-sign-detection. Accessed: June 30, 2025.

GITHUB. jean2612/Dataset-Placas-Transito. Available at: https://github.com/jean2612/Dataset-Placas-Transito. Accessed: June 30, 2025

EDGE IMPULSE. *Building a Speed Sign Detector with Edge Impulse* [video]. YouTube, 6 months ago. Available at: https://www.youtube.com/watch?v=F0xlOGt6RJY&t=1242s. Accessed: July 1, 2025.

KAGGLE. *Road Sign Detection* [digital resource]. Kaggle, 2020. Available at: https://www.kaggle.com/datasets/andrewmvd/road-sign-detection. Accessed: July 1, 2025.