EMBEDDED SYSTEMS

Classification of healthy and diseased leaves

2025/2026

Francesco Rivitti 10695516

January 28, 2026

Contents

1 Introduction

1.1 Problem Statement

One of the most pressing issues affecting both urban vegetation and agricultural fields is the spread of pests and infestations. These organisms can cause significant damage to plants, compromising not only their esthetic value but also their health and productivity. One of the most critical challenges facing farmers today is the delayed detection of crop stress, caused by diseases, pests, or nutrient deficiencies, which contributes to an estimated financial loss of up to **€250 billion** annually worldwide. This highlights how insect infestations are not only a local environmental issue but also a major economic problem.

Within this broader context, I decided to focus on the Japanese beetle (**Popillia japonica**), since in Milan, its presence has become almost ubiquitous. This invasive species is particularly problematic because it damages the leaves of several tree species by feeding on their foliage, weakening the plants and reducing their vitality. P. japonica was first detected in Lombardy in 2014, and its range has since expanded by about **10 kilometers per year**. Today, the only province in Lombardy still unaffected is Mantua. In response to the growing number of outbreaks, the Lombardy Region Phytosanitary Service has implemented various measures to limit and control the spread of the beetle, including annual visual inspections.

1.2 Proposed Solution

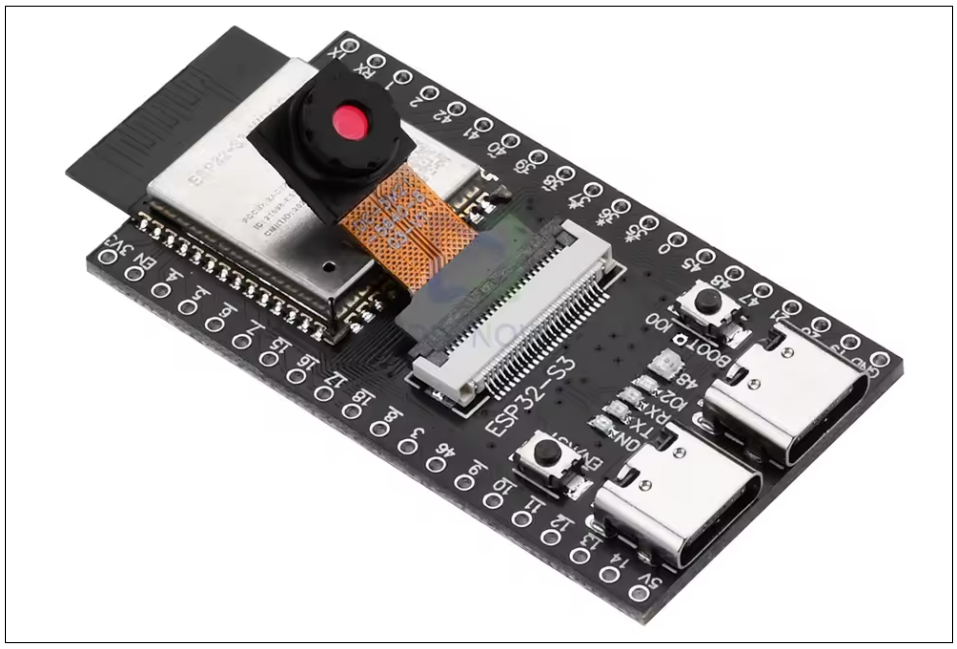
Ground monitoring: an ESP32-based system capable of classifying healthy vs. diseased leaves in real time. The device aims to provide continuous monitoring of trees in urban parks. Instead of just doing annual checks.

1.3 Hardware Specifications

1.3.1 Microcontroller

The chosen microcontroller is the **ESP32-S3 WROOM-1**, with the following relevant specifications:

- 16 MB Flash
- 512 kB RAM
- 8 MB PSRAM
- Dual-core Xtensa LX7 @ 240 MHz
- Integrated support for camera interfaces



ESP32 S3 with camera module

1.3.2 Camera Module

For image acquisition, I employed the **OV2640** camera module, which is fully compatible with the ESP32-S3 and widely used in embedded vision applications due to its compact size and low power consumption. The main specifications are:

- **Resolution:** up to 2 Mp (1600×1200 px)
- **Output formats:** RGB565, YUV422, JPEG, and grayscale
- **Maximum frame rate:** 15 fps at UXGA (1600×1200 px), up to 30 fps at SVGA (800×600 px)

- **Special features:**
  - Image scaling and cropping
  - Auto white balance (AWB), auto exposure control (AEC), auto gain control (AGC)
  - Support for JPEG compression on-chip

The OV2640 represents a good trade-off between image quality and hardware requirements, making it Ill suited for resource-constrained AI inference on microcontrollers such as the ESP32-S3.

1.4 Project Goal

The objective of my work is to design an embedded system device capable of classifying a linden leaf image as either healthy or diseased. To be effective in real-world applications, the model must satisfy several requirements:

- Classification **accuracy** of at least **90%**.
- End-to-end inference time below **1 second**.
- **Flash memory** usage below **1 MB**.
- **RAM** usage within the **512 kB** limit, with optional support from **PSRAM**.

2 Data Acquisition

Images of linden tree leaves were collected using the **OV2640** camera module. The dataset consisted of 97 images, including:

- **50** diseased leaves
- **47** healthy leaves

Images were resized to 240×240 pixels.

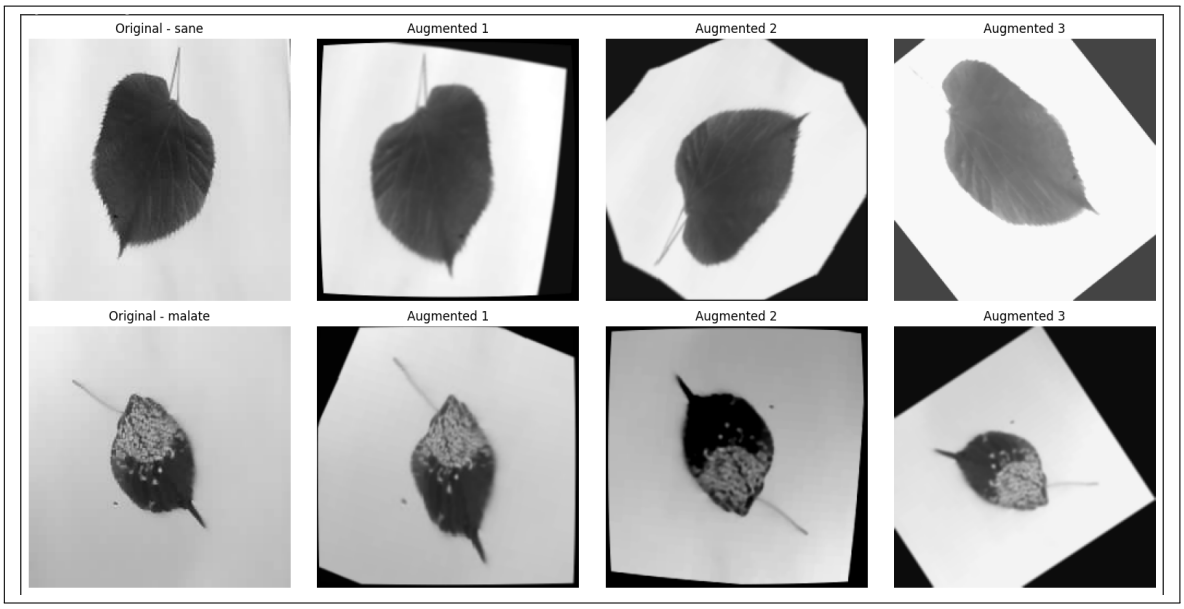
Healthy leaf

Diseased leaf

2.1 Dataset Augmentation

Since the initial dataset consisted of only **97 images** (50 diseased and 47 healthy leaves), I applied **data augmentation** to increase its size and improve model generalization. Data augmentation is a technique that generates new training samples from existing ones by applying random transformations such as rotations, flips, scaling, brightness adjustments, or noise injection. These transformations preserve the semantic meaning of the images

(i.e., healthy or diseased leaves) while introducing variability that makes the model more robust to real-world conditions.



Augmentation Examples

Using a custom script on Google Colab, I augmented the dataset to a total of:

- **200** diseased leaf images.
- **188** healthy leaf images.

This balanced and extended dataset allowed the model to better learn discriminative features and reduced the risk of overfitting due to the limited number of original samples.

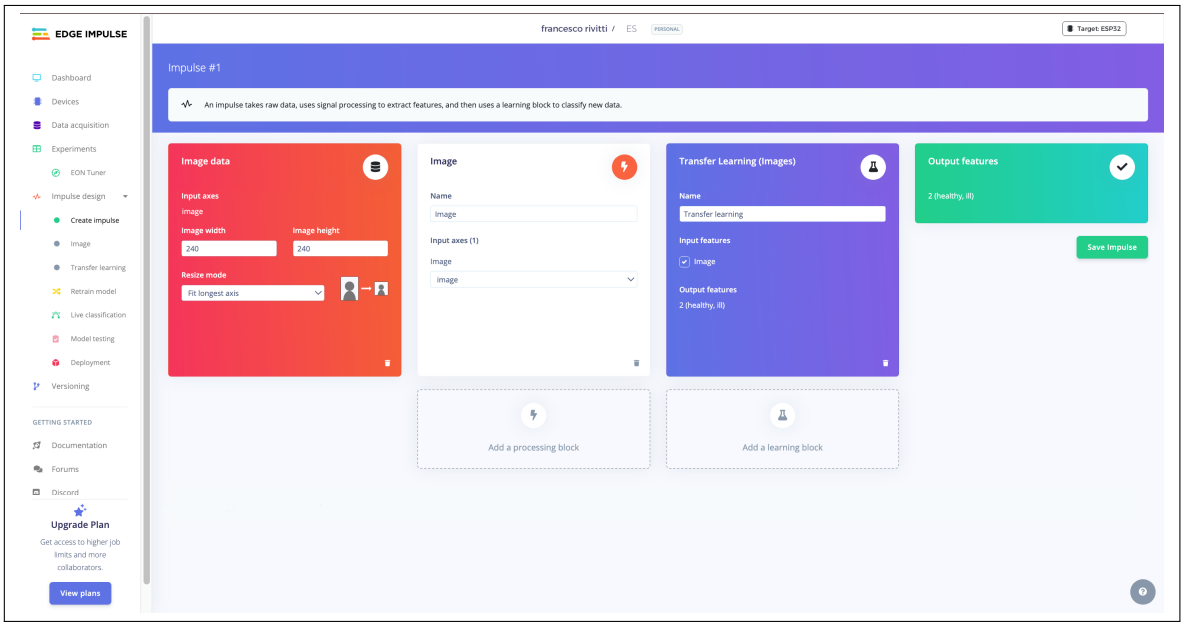
3 Model Training on Edge Impulse

3.1 Impulse Creation

The dataset was uploaded to Edge Impulse, automatically split into:

- 80% training set.
- 20% test set.

Different input resolutions (96×96, 160×160, 240×240) and preprocessing pipelines were tested. The classifier was trained to output two classes: ***healthy*** and ***ill***.



Edge Impulse Create impulse tab

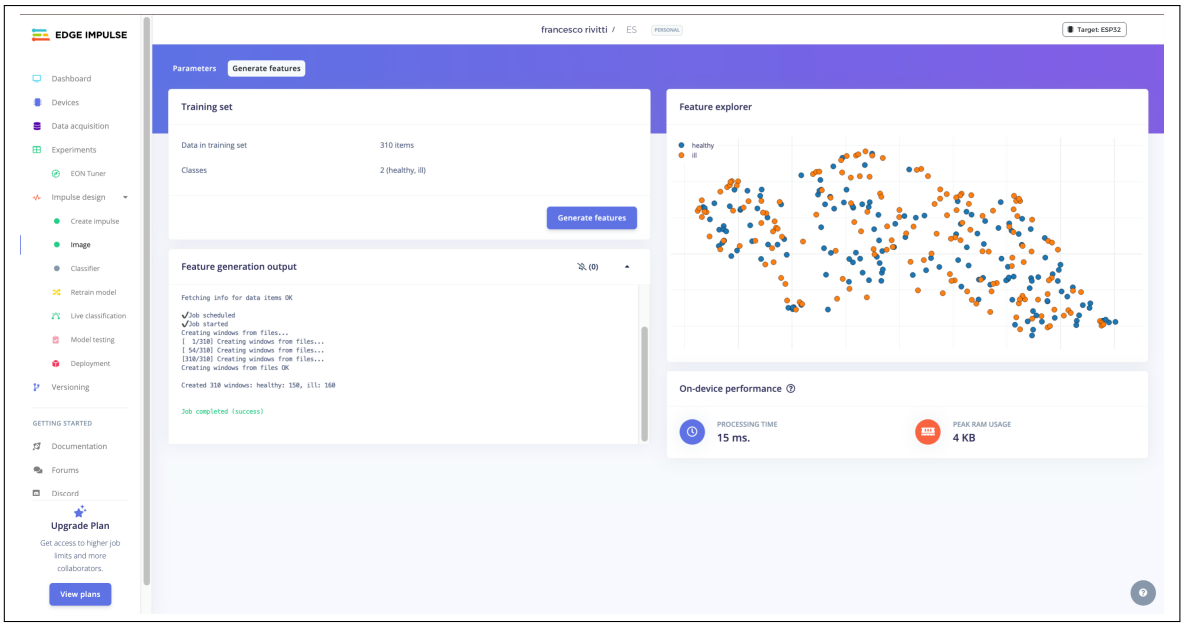
3.2 Feature Extraction in the Image Tab

In the **Image** tab of Edge Impulse, I configured the preprocessing pipeline to transform the raw input images into feature vectors suitable for model training. This step is essential because it reduces image complexity while preserving the relevant information for classification.

Two different input configurations were tested:

- **Grayscale images:** reduce the dimensionality of the input, requiring less memory and computational power. This representation is particularly useful for small models (e.g., MobileNet 96x96) that cannot handle the complexity of RGB input.
- **RGB images:** preserve the full color information of the leaves, allowing the model to exploit additional features such as color variations due to disease. However, this requires higher memory usage and longer inference times.

Feature extraction generated a set of numerical descriptors from the processed images. These descriptors served as the actual input to the classifier, significantly impacting both accuracy and resource consumption. The experiments highlighted a trade-off between **accuracy** (generally higher with RGB inputs) and **efficiency** (faster inference and lower memory footprint with grayscale inputs).

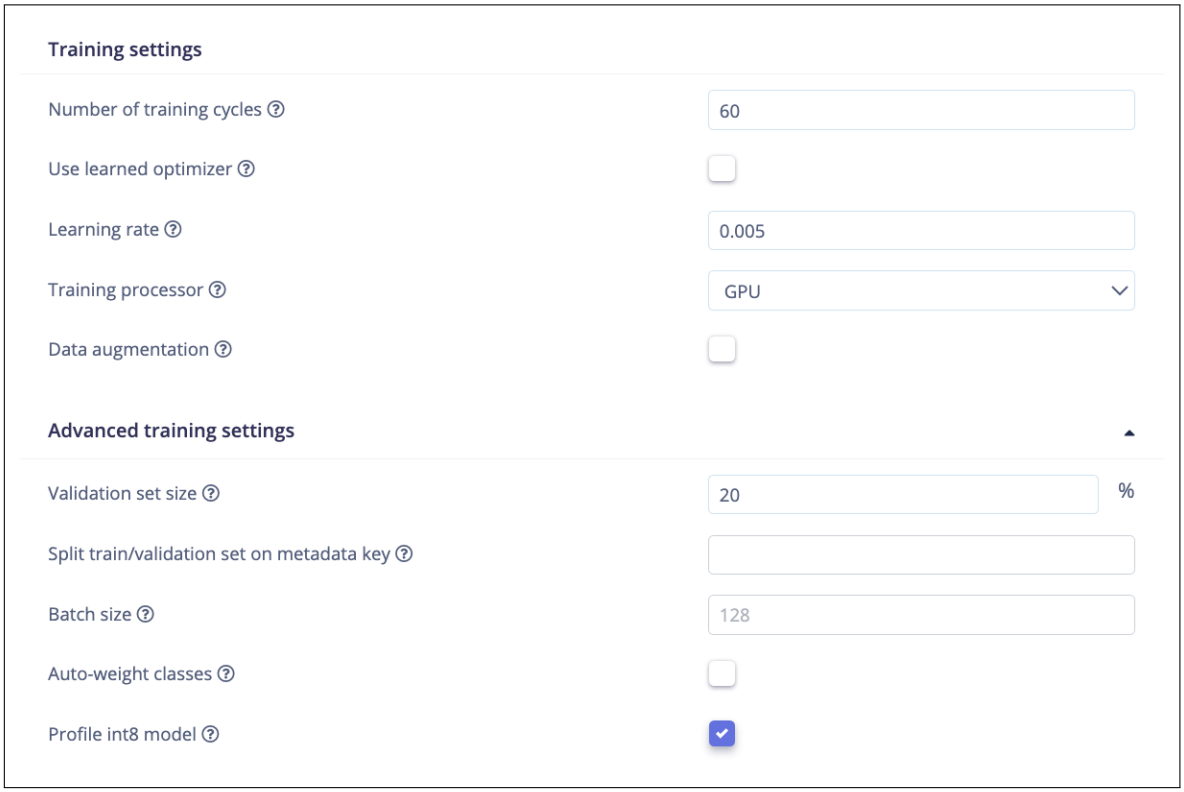


Edge Impulse Image tab

3.3 Hyperparameter Tuning

Several adjustments were made:

- Epochs increased to 60 to avoid underfitting.
- Learning rate increased from 0.0005 to 0.005.
- 20% validation set used for unbiased model evaluation.



Training settings

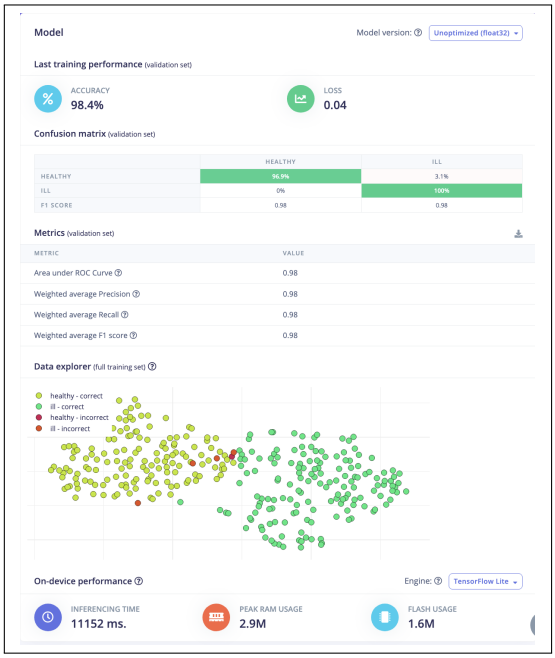
3.4 Best Performing Model

The best model was based on **MobileNetV2 (160x160,** $\alpha = 0.35$**)**, with:

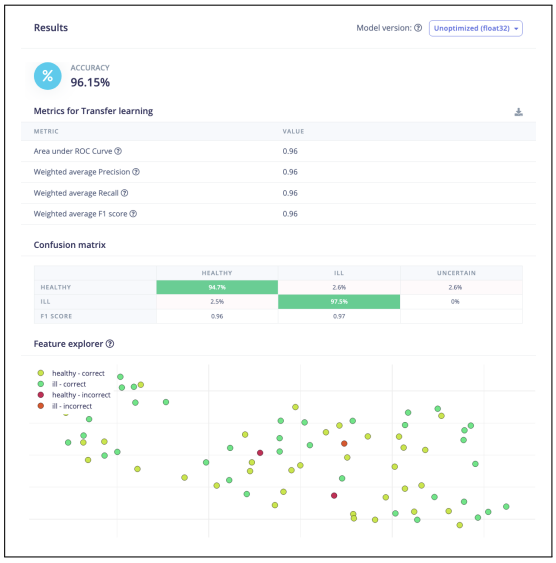
- Intermediate dense layer with 8 neurons before the output layer.
- Dropout layer (0.5) to reduce overfitting.

The model achieved:

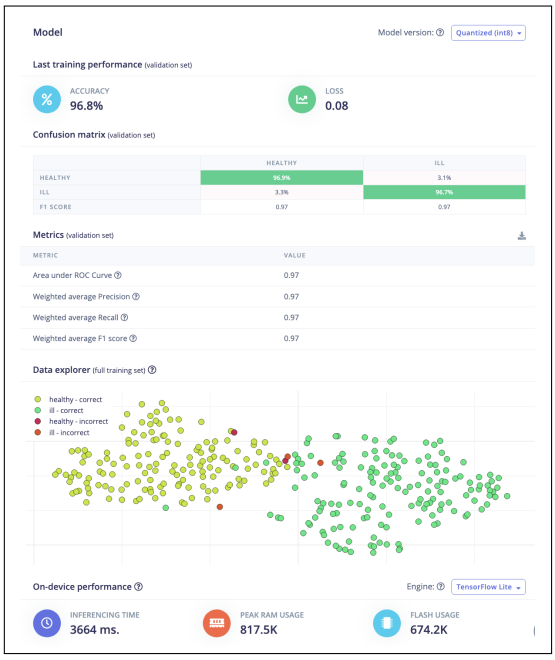
- Accuracy $> 97\%$.
- Inference time ~ 500 ms (measured on Arduino IDE).
- Flash usage ~ 674 kB, RAM ~ 817 kB (too high); the systems will also use PSRAM.



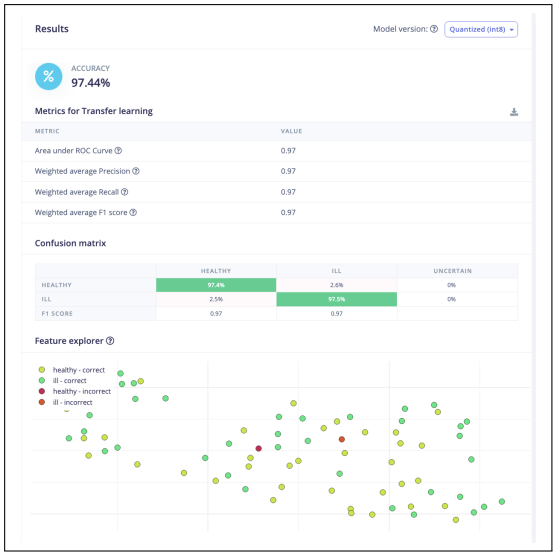
Best performing model float32



Testing best performing model float32



Best performing model int8



Testing best performing model int8

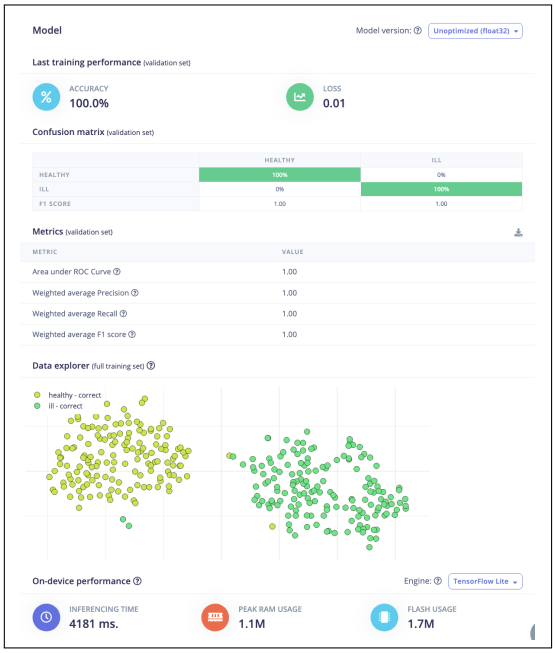
3.5 Lighter Model

The second proposed model was based on **MobileNetV2 (96x96, $\alpha = 0.35$)**, with:

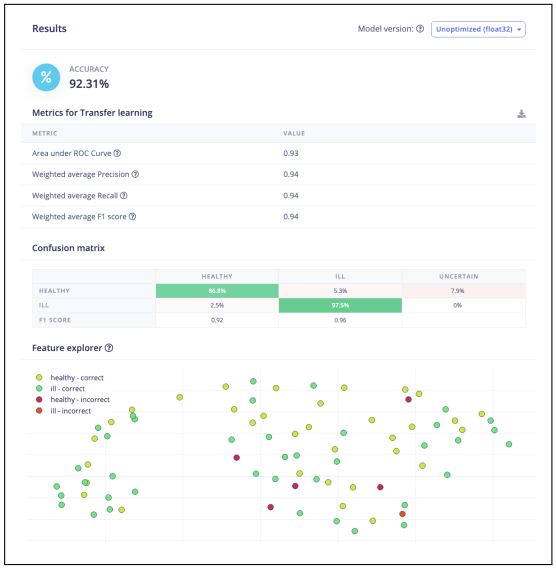
- Intermediate dense layer with 16 neurons before the output layer.
- Dropout layer (0.1).

The model achieved:

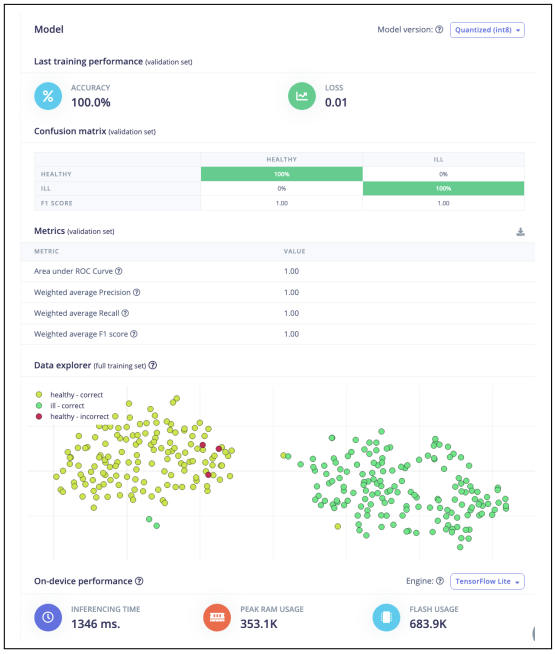
- Accuracy $\sim$ 91%.
- Inference time $\sim$ 200 ms (measured on Arduino IDE).
- Approximately the same flash usage.
- RAM usage has been reduced to 353 kB.



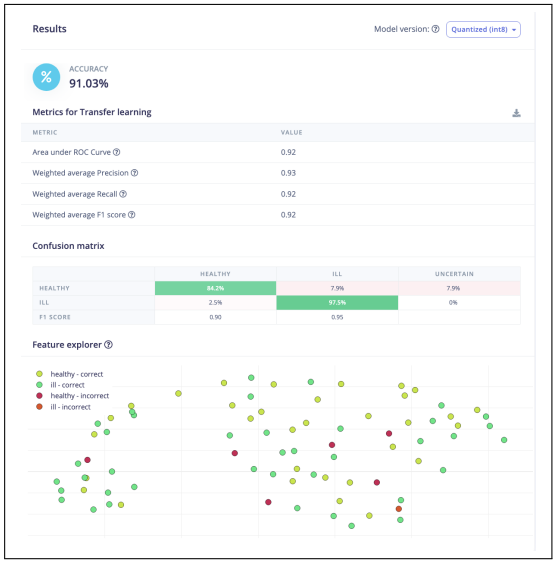
Transfer learning model float32



Model testing model float32



Transfer learning model int8



Model testing model int8

4 Deployment on ESP32

4.1 Export and Integration

The trained model was exported from Edge Impulse as an Arduino library in int8 quantized format. I used the `esp32_camera` example from the edge impulse library and modified the pin mappings to support our specific camera model.

- **Best Performing Model**

```
Predictions (DSP: 12 ms., Classification: 532 ms., Anomaly: 0 ms.):
Predictions:
  healthy: 0.39453
  ill: 0.60547
```

532 ms inference time for **MobileNetV2 (160x160,** $\alpha = 0.35$**)**

- **Lighter Model**

```
Predictions (DSP: 3 ms., Classification: 207 ms., Anomaly: 0 ms.):
Predictions:
  healthy: 0.08594
  ill: 0.91406
```

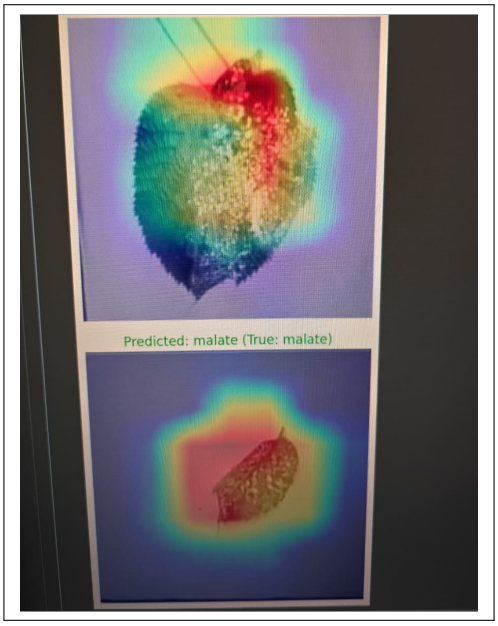
207 ms inference time for **MobileNetV2 (96x96,** $\alpha = 0.35$**)**

You can follow this link for a GIF demonstration of how it works.

5 Explainability: Class Activation Maps (CAM)

To better understand the model's decision-making process, I used Class Activation Maps (CAM). This technique, commonly applied in computer vision models such as CNNs, highlights the regions of an input image that are most relevant for the model's prediction. In practice, it generates a heatmap where brighter (or hotter) areas show where the network "looked" to make its decision.

I applied CAMs to verify whether the model was identifying diseased plants for the right reasons, for instance, by focusing on the characteristic holes caused by Popillia japonica, or if it was relying on unrelated features. The results revealed that the classifier indeed focused on the damaged regions of the leaves when predicting the "ill" class, confirming the extraction of meaningful features. This step is crucial: if the model were assigning correct labels for the wrong reasons, it could lead to unreliable behavior during deployment.



CAM Heatmaps

6 Custom Model Development SqueezeNet

6.1 Motivation

To further investigate the effectiveness of **transfer learning**, I designed and trained a custom deep learning model based on the SqueezeNet architecture, without relying exclusively on the Edge Impulse platform.

6.2 Model Architecture

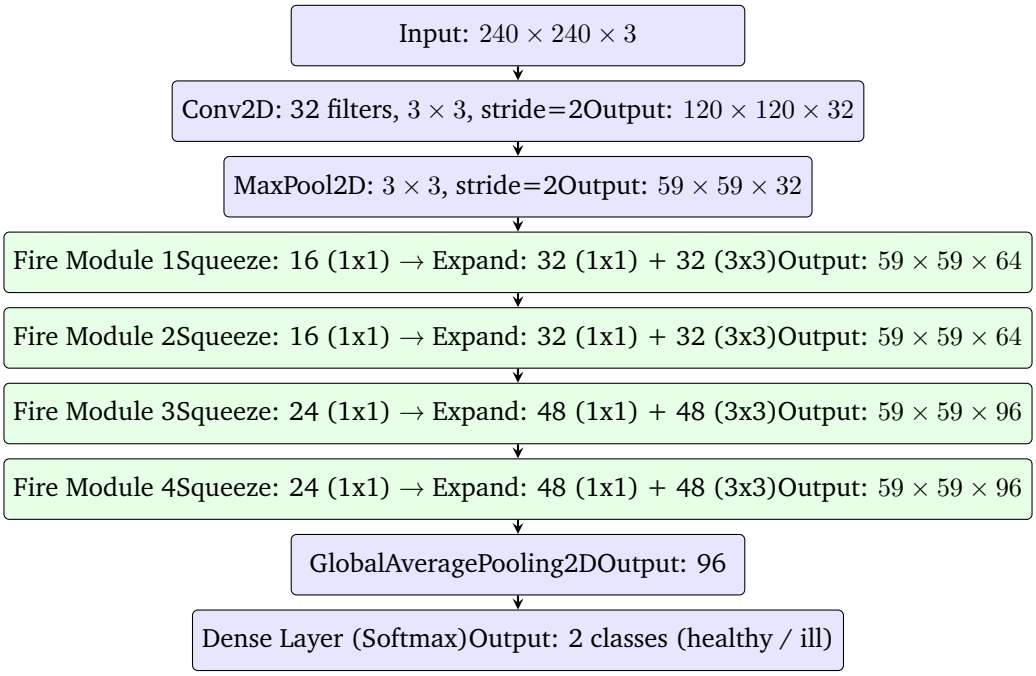


Figure 6.1: Custom SqueezeNet architecture with four Fire Modules.

6.3 Transfer Learning Approach

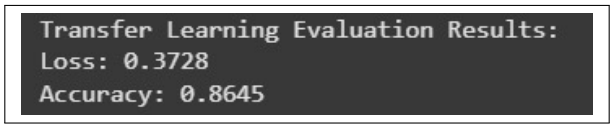
To overcome the limitation of a limited dataset, I applied a transfer learning strategy:

1. **Pretraining on Imagenette:** The same SqueezeNet architecture was pretrained on the *Imagenette* dataset, a subset of ImageNet containing 10 classes and about 14,000 images. I only modified the final dense layer to output 10 classes instead of 2. After training, the model achieved about **80% accuracy** on the Imagenette test set, which was sufficient since the goal was to initialize convolutional filters with useful weights.
2. **Freezing convolutional filters:** Once pretrained, all convolutional layers were frozen, meaning their weights remained fixed during subsequent training.
3. **Replacing the classification head:** The final dense layer with 10 neurons was replaced by a new dense layer with 2 neurons, corresponding to the *healthy* and *ill* classes of our dataset.
4. **Fine-tuning on leaf dataset:** Only the final dense layer was trained on our leaf dataset, while the pretrained convolutional filters remained unchanged.

6.4 Results

This transfer learning strategy significantly improved performance:

- Accuracy improved from **57% (scratch)** to **86% (transfer learning)**.



Transfer learning results

- The non-quantized model size was only $150\,\mathrm{kB}$.
- After quantization to int8, accuracy dropped slightly to around **80%**.

Quantized SqueezeNet size

Accuracy on the test set

The quantized model was successfully uploaded to Edge Impulse, exported as an Arduino library, and deployed on the ESP32. The deployment worked correctly, although the inference time was relatively high ($\sim$2 seconds).

This improvement confirms that pretraining on a large dataset enabled the convolutional filters to learn rich and meaningful features, which were then effectively transferred to the leaf classification task. The pretrained filters act as strong feature extractors, far superior to those learned from scratch on a small dataset like mine.

7 Conclusions

In this project, I investigated the deployment of lightweight deep learning models on resource-constrained embedded systems for the detection of diseased leaves, with a specific focus on execution on low-power microcontroller-based platforms such as the ESP32. Through experiments conducted using Edge Impulse, I evaluated different preprocessing strategies, model architectures, and hyperparameter configurations, explicitly considering memory usage, inference latency, and computational constraints typical of embedded environments.

Through experiments with Edge Impulse, I evaluated different preprocessing strategies, model architectures, and hyperparameter settings. The best-performing model, based on MobileNetV2 (160×160, $\alpha = 0.35$), achieved an accuracy above 97% with an inference time of around 500 ms.

To better understand the model's decision-making process, I applied Class Activation Maps (CAM), which confirmed that the classifier correctly focused on the damaged regions of the leaves caused by Popillia japonica. This result is particularly important, as it ensures that the model is learning the right features for disease detection rather than relying on spurious correlations.

Finally, I experimented with a custom lightweight architecture based on SqueezeNet, which represents a promising direction for balancing performance with memory efficiency. Despite everything, the inference time was too high (2 seconds), which does not meet the initial time constraint.

Overall, this work demonstrates the feasibility of integrating AI-based perception capabilities into embedded systems operating under strict resource constraints. Future improvements may include further model optimization and energy-aware deployment on mobile embedded platforms such as drones, enabling scalable and autonomous monitoring of large areas without relying on extensive manual inspections or complex distributed sensor infrastructures.