Ollscoil na hÉireann, Corcaigh

National University of Ireland, Cork

Lightweight Edge AI Vision Models for IoT-Based Insect Monitoring

Thesis presented by

Amin Kargar Barzi

ORCID 0000-0001-9402-000X

for the degree of

Doctor of Philosophy

University College Cork

Tyndall National Institute

Head of School/Department: Dr Fatima Gunning

Supervisor(s): Dr. Brendan O'Flynn, Dr. Salvatore Tedesco, Dr. Michael Gaffney, Dr. Dimitrios Zorbas

2025

TABLE OF CONTENTS

List of figures

List of tables

Declaration

This is to certify that the work I am submitting is my own and has not been submitted for another degree, either at University College Cork or elsewhere. All external references and sources are clearly acknowledged and identified within the contents. I have read and understood the regulations of University College Cork concerning plagiarism.

Amin Kargar Barzi

Acknowledgement

This work would not have been possible without the financial support provided by a number of funded programmes, for which I am sincerely grateful. This work was carried out as part of the Haly.ID project, 2020EN508 funded by Ireland's Department of Agriculture, Food and the Marine under Grant: 2020 Trans National ERA-NET. This project is co-funded by the European Regional Development Fund (ERDF) under Ireland's European Structural and Investment Funds Programmes 2014–2020. I am supported by a Walsh Scholarship funded by Teagasc, The Irish Food and Agriculture Authority. Aspects of this work have been supported by Research Ireland under Grant 12/RC/2289-P2-INSIGHT2, 13/RC/2077-CONNECT and 21/RC/10303-P2-VISTA MILK2 which are co-funded by the European Regional Development Fund (ERDF) under Ireland's European Structural and Investment Funds Programme 2014-2020.

I would like to take this opportunity to express my deepest gratitude to my supervisors, Dr. Brendan O'Flynn, Dr. Salvatore Tedesco, Dr. Michael Gaffney, and Dr. Dimitrios Zorbas, for their continuous guidance, encouragement, understanding, and support throughout these years, in every situation, work or otherwise. Their mentorship has shaped every part of this journey, and none of this would have been possible without their support.

I would also like to express my sincere thanks to the Tyndall National Institute for providing an incredible research environment, resources, and support throughout my PhD. The facilities, technical expertise, and everyday guidance from the wider Tyndall community played a major role in shaping this work. I am also grateful to Teagasc for their collaboration and financial support, which greatly contributed to the progress of this research. Special thanks are extended to all the HALY.ID project partners for their valuable collaboration, discussions, and fieldwork support. I would in particular like to thank Dr. Francesco Betti Sorbelli for his contribution to the deployment of the IoT device used for data collection, and Dr. Mariusz Wilk for his contribution to the design and development of the first version of the IoT device.

Special thanks to my colleagues and friends at Tyndall National Institute, specifically the WSN group, who helped, encouraged, and motivated me along the way.

My heartfelt thanks go to my parents, my sisters, and my in-laws, along with their families, for their endless love, support, and belief in me, even from afar.

Finally, and most importantly, to my love, thank you for being my constant source of strength, patience, and happiness throughout this journey. Thank you for standing beside me through every up and down. I will always be grateful to you.

Abbreviations and Notations

AI: Artificial Intelligence
AIA: Average Incremental Accuracy
AP: Average Precision
BCE: Binary Cross Entropy
BCE-Dice: Binary Cross Entropy and Dice Loss
BLE: Bluetooth Low Energy
BMSB: Brown Marmorated Stink Bug
BWT: Backward Transfer
CCL: Connected-Component Labelling
CI: Confidence Interval
CLAHE: Contrast Limited Adaptive Histogram Equalization
CL: Continual Learning
CI: Confidence Interval
DML: Distance Metric Learning
DE-Net: Dilated Encoder Network
DL: Deep Learning
DSC: Dice Similarity Coefficient
DwSConv: Depthwise Separable Convolution
FSL: Few-shot Learning
FC: Fully Connected
FAO: Food and Agriculture Organization of the United Nations
FE: Feature Extractor
FWT: Forward Transfer
GAN: Generative Adversarial Network
GAP: Global Average Pooling
HH: Halyomorpha halys
ICT: Information and Communications Technology
IL: Incremental Learning
IoT: Internet of Things
IoU: Intersection over Union
IPM: Integrated Pest Management
LLMs: Large Language Model
MAC: Multiply-Accumulate
MAE: Mean Absolute Error
MCU: Microcontroller Unit
MSE: Mean Squared Error
NAS: Neural Architecture Search
P: Precision
PA: Precision Agriculture
PAM: Presence-Absence Model
PCA: Principal Component Analysis
R: Recall
ReLU: Rectified Linear Unit
RoI: Regions of Interest
RPi: Raspberry Pi
SConv: Standard Convolution
SSL: Semi-supervised Learning
tflite: TensorFlow Lite
YOLO: You Only Look Once

Abstract

Smart automated insect monitoring is essential for early detection of insect pest infestations in orchards. It assists farmers in controlling insect pest populations in their fields and preventing crop losses and improving crop quality. Traditional approaches relying on manual inspections are labor-intensive, error-prone, and highly challenging for large-scale or remote orchards. With recent advances in Artificial Intelligence (AI) and Information and Communications Technology (ICT), there is an opportunity to automate the insect monitoring process effectively and directly in the field. However, key challenges persist, such as limited energy availability, unreliable connectivity in remote areas, data scarcity, and the dynamic nature of orchards. This thesis addresses these challenges through an integrated framework for AI-powered IoT-based resource-constrained edge devices, targeting Brown Marmorated Stink Bug.

The work was carried out in three main phases. First, a low-power edge-based detection system was developed and deployed using a Microcontroller Unit (MCU)-based OpenMV board equipped with a camera to explore the feasibility of capturing, processing, and transmitting insect trap images entirely on the edge. This involved optimizing the entire pipeline, including image capture, inference, data transfer, and energy management, while ensuring the system could operate in harsh and remote orchards conditions with limited power and connectivity. The main aim was to validate the concept of performing all image processing directly on low-power consumption, resource constrained hardware and to evaluate its practicality in deployment scenarios. The device operated fully on-device without cloud connectivity and processed two images in ~30 seconds while consuming less than 300 mA of current in active mode. Additionally, this phase enabled the collection of a real-world image dataset under uncontrolled outdoor conditions.

In the second phase, the thesis shifts toward efficient Deep Learning (DL) model design, building on the real-world data obtained in the first stage. A lightweight DL model, named SemiY-Net, was proposed for simultaneous object segmentation and counting. The model was designed to meet the strict memory and computational constraints of MCUs while providing meaningful outputs. Several strategies were employed to achieve this balance, including architectural pruning, layer-level memory profiling, and skip connection management. Unlike typical segmentation models

designed for high-resource platforms, SemiY-Net introduces a design that deliberately reduces spatial resolution specifically at the output, trading off accuracy in favor of deployment feasibility. The model was trained and validated on a dataset of annotated insect trap images, supporting both segmentation masks and insect count labels. In addition to its accuracy, it runs entirely on-board MCUs without reliance on a server or cloud support. Quantitatively, SemiY-Net reduced parameters by >75% compared to state-of-the-art models, required <1 MB of storage and RAM, and achieved ~5 J energy consumption per inference on an OpenMV. On the field dataset collected in Phase 1, it achieved Dice Similarity ~85% and counting Mean Squared Error ~1.32.

The final stage of the thesis addresses a fundamental challenge in deploying DL models in real-world scenarios, particularly the difficulty of collecting and labeling proper datasets in remote and dynamic environments. To overcome this, a self-improving and fully edge-based framework was developed where the MCU-based edge nodes are supported by an edge server (e.g., Raspberry Pi) that handles DL model retraining and model adaptation. The MCUs perform real-time inference using the proposed SemiY-Net and transmit only compressed Regions of Interest (RoIs) to the edge server, reducing communication overhead by ~84%. The edge server then reconstructs the raw captured images from the received RoIs, generates the corresponding pseudo- labels and masks, and fine-tunes SemiY-Net to adapt to new field conditions and data distributions. In addition to the reconstructions, artificial images were also synthesized from the RoIs to increase the training sample diversity. These reconstructed and augmented samples were then used in periodic continual learning cycles, where the server fine-tuned SemiY-Net with the new data and redistributed updated model weights back to the edge nodes. This system enables continuous adaptation without human intervention. Quantitatively, models fine-tuned with pseudo-labels retained accuracy comparable to those trained with expert annotations, while inference energy consumption on MCUs remained ~5 J and communication energy consumption dropped >80% compared to raw image transfer.

Together, these three phases form a practical edge AI-based IoT system for insect monitoring. The system covers the entire pipeline, from hardware to model development and deployment. By eliminating reliance on cloud and human annotation, it delivers intelligent automation directly to the orchard, enabling reliable and long-term insect monitoring even in resource-constrained environments.

1 CHAPTER I. INTRODUCTION

1.1 Research context and framework of the HALY.ID project

This PhD thesis (funded as a Teagasc Walsh scholar) was part of a European-funded research project (under the Era-Net programme[1]) named HALY.ID[2] aimed to develop innovative technologies to monitor and sustainable management of the Brown Marmorated Stink Bug (BMSB), also known as *Halyomorpha Halys* (*HH*). This was a collaborative research project between the Tyndall National Institute, University College Cork (Ireland), Teagasc - Agriculture and Food Development Authority (Ireland), Università degli Studi di Perugia (Italy), Università degli Studi di Modena e Reggio Emilia (Italy), Technische Universität Braunschweig (Germany), University Polytecnica of Bucharest (Romania), and One Planet Research Center, Imec (Netherlands). The system proposed in HALY.ID incorporates IoT technologies, image processing, drone-based imaging, hyperspectral imaging, and environmental monitoring for the purposes of early detection and monitoring of invasive insect species and supporting sustainable pest management, as well as developing a postharvest sorting system to identify BMSB-induced damage on fruits, as shown in Figure 1.1. As part of this project, this thesis focused on smart edge-based system for insect monitoring at orchards.

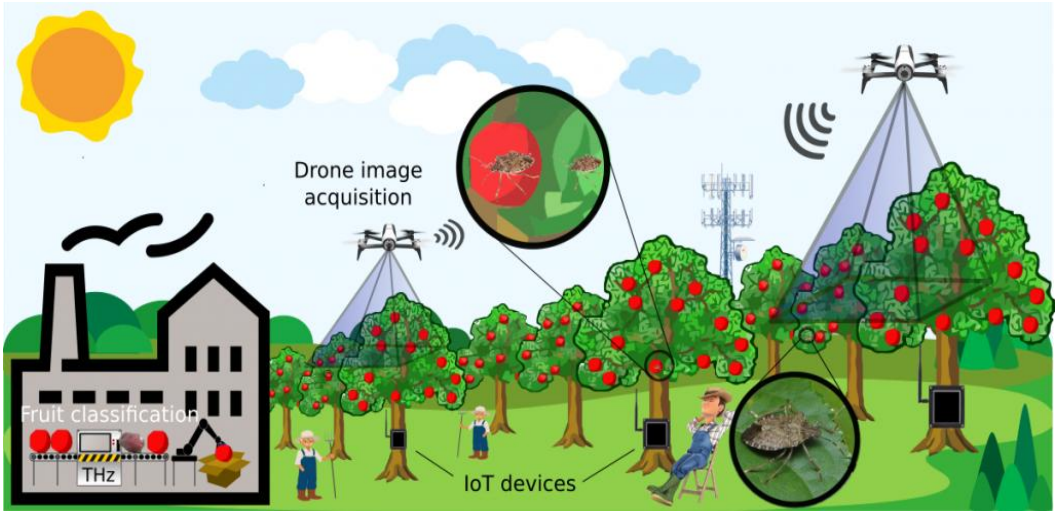


Figure 1.1. Overview of the HALY.ID project.

[1] https://ictagrifood.eu/
[2] https://www.haly-id.eu/

1.2 Research motivation

Insects, despite their small size, have a significant impact, both positive and negative on ecosystems, agriculture, and food systems. Pest insects, in particular, threaten food security by reducing crop yield and quality. As the global population grows, ensuring food availability becomes even more important [1]. According to the Food and Agriculture Organization of the United Nations (FAO), up to 40% of global crop production is lost each year to plant pests and diseases, costing over $220 billion annually, with invasive insect species causing at least $70 billion [2] of damage. Factors, such as climate change and global trade, contribute to the spreading of these insects into new areas where they did not exist before [3] as environmental conditions change. They can also crawl inside vehicles, cargo, or shipping containers, and this "insect tourism" or "hitchhiking" behaviour plays a major role in their rapid spread across regions [4].

Managing and controlling invasive insects in a sustainable way is a significant challenge. While pesticides are commonly used, they can harm the environment and food quality if overused or misused [5][6]. Consequently, effective pest monitoring is important to control insect populations, as it could help farmers know when and where action is needed and which control method is best, while simultaneously reducing unnecessary pesticide use. Currently, most insect monitoring relies on sticky traps that farmers must check manually. They attract insects using colours or pheromones, but farmers must physically inspect each trap, identify the insects by eye, and count them. This is not efficient, especially in large or remote orchards. Moreover, it is time-consuming, often inaccurate, and might require a high level of domain specific knowledge and field experience for accurate identification [7][8].

With recent advances in Artificial Intelligence (AI) and Information and Communications Technology (ICT), there is an opportunity to automate the insect monitoring process effectively. This could reduce the need for onsite visits, reduce fuel costs and time spent manually monitoring traps, and support larger goals such as the reduction of pesticide and improving food quality and quantity [9][10]. However, bringing AI into the field in such technologies comes with multiple challenges. Often orchards and farms are far from urban centres and these remote areas often lack reliable internet connectivity or power, and the installation of expensive or power-

hungry systems is not practical, since such costly solutions are not cost-effective or sustainable for these remote environments [11][12][13]. Equally important, there is rarely enough annotated data to train a reliable model for insects, and settings change in orchards which causes trained models to degrade over time [14][15]. Thus, the system needs to be not only efficient and low-power, but also able to keep learning and adapting without human help. This thesis focuses on addressing these challenges, developing edge AI technologies for insect monitoring as part of a real-world application so as to design, build, and test an AI-based solution that can practically work in remote orchards environments. This provides farmers with the necessary information to establish an early warning system for insect population growth in their orchards and farms, so they can take effective and sustainable actions to mitigate the damage these insects can cause.

Halyomorpha Halys (see Figure 1.2) is a destructive example of an invasive insect originally from East Asia which traditionally was found in China, Japan, Korea, and Taiwan but which now infests orchards in North America and across Europe. This invasive insect has a life cycle that includes overwintering adults, egg masses, five nymphal instars, and new adults. Egg masses are laid on the underside of leaves, and nymphs hatch and progress through five instars, increasing in size from about 2.4 mm to around 12 mm. Adults measure 12-17 mm, have a mottled brown shield-shaped body with characteristic white bands on the antennae and legs [16].

Seasonal activity begins when adults emerge from overwintering sites between April and early June, depending on environmental factors such as temperature. Activity increases during summer, and adults start overwintering from late September

Figure 1.2. The Brown Marmorated Stink Bug or Halyomorpha halys on a pear fruit (Source: HALY.ID)

to mid-October. The species can produce one or two generations per year depending on climate [16]. This is a polyphagous insect that can cause considerable damage to a wide range of fruit trees, vegetables, and ornamental plants, such as apples, pears, peaches, hazelnuts, grapes [17]. Italy is one of the geographical areas that is highly infested by this insect; for instance, in 2019 this insect caused huge damage to fruit production in northern Italy, costing €588 million [18]. The control of BMSB is challenging due to multiple contributing factors, such as their high reproduction rate, mobility, tendency to feed on multiple types of plants, and the inefficacy of general insecticides against them [19]. Due to its proliferation in orchards across Europe and the high production loss that it causes, this insect was selected as a research focus within HALY.ID project.

1.3 Objectives

This PhD research focuses on the development of a practical, robust, and energy-efficient edge AI-based framework that can operate reliably under the real-world conditions of remote orchards for insect detection and identification. Therefore, the main objectives of this PhD research thesis are as follows:

- To develop a low-power edge-based prototype device for insect detection and identification that can capture and analyze insect trap images and transmit required data using a resource-constrained microcontroller unit (MCU), enabling reliable operation in remote agricultural environments with limited power and connectivity.
- To develop and validate a novel TinyDL model designed for deployment on a resource-constrained MCU for automated insect monitoring, capable of performing insect segmentation and counting under strict memory and computational constraints.
- To design and validate a practical and comprehensive edge-based insect monitoring framework that operates reliably under real-world agricultural constraints to address fundamental deep-learning (DL) challenges, including the lack of annotated datasets and post-deployment data drift.

1.4 Scientific contributions

This thesis presents a practical edge AI-based system for automated insect monitoring in agricultural environments. The main scientific contributions of this work are summarized as follows:

- The design and deployment of a low-power edge-based insect monitoring device capable of performing image capture and inference directly on resource-constrained microcontroller units (MCUs) under real-world agricultural conditions.
- The development of a lightweight deep learning model (SemiY-Net) for insect segmentation and counting that operates within the strict memory and computational constraints of MCU-based platforms.
- The proposal of a self-improving edge-based monitoring framework that addresses the challenges of limited annotated datasets and post-deployment data drift through pseudo-labeling and incremental learning.
- The validation of the proposed edge-AI insect monitoring system using real-world datasets collected under uncontrolled orchard deployment conditions.

The structure of the thesis and the detailed contributions of each chapter are described in the following section.

1.5 Thesis layout and novelties

The remainder of this thesis is organized as follows:

Chapter 2: This chapter provides background definitions and a comprehensive review of the state of the art in image-based systems and associated AI algorithms used for insect detection and population estimation. This review identified clear gaps in the current state of the art in this research area which are addressed in the following technical chapters of the thesis.

Chapter 3: This chapter addresses the first objective of this PhD thesis by developing and validating a low-power edge-based prototype device for insect detection and identification using resource-constrained MCUs.

In this chapter, a low-power edge-based detection system was developed and deployed in an orchard using an MCU-based board equipped with a camera to investigate the feasibility of capturing, processing, and transmitting insect trap images on the edge. This involved optimizing the entire pipeline, including image capture, inference, data transfer, and energy management, while ensuring the system could operate in remote conditions with limited power and internet access. The main aim was to validate the concept of performing processing directly on ultra-low-power hardware and to evaluate its practicality in deployment scenarios. Additionally, this phase enabled the collection of a real-world image dataset under uncontrolled outdoor conditions to be used in the following chapter.

The main contributions of this chapter are summarized as follows:

- A low-power consumption and low-cost vision-based smart insect monitoring system based on MCUs to detect and count insects in orchards, specifically the Brown Marmorated Stink Bug, is proposed.
- A custom lightweight and low-power DL model for image classification to be implemented on resource-constrained MCU with a memory capacity in the range of kilobytes is proposed, evaluated, and compared to other solutions in the literature.
- A power characterization of the proposed system is performed to confirm its low-power operation and the power efficiency of the proposed ML approach.

The work in this chapter is adapted from my first-author peer-reviewed articles (1, 2, 4) as detailed below. From the collaborative project-wide paper (3), only Section V, corresponding to my contribution, is included in this chapter of the thesis.

1. **Kargar, A.**, Zorbas, D., Tedesco, S., Gaffney, M., & O'Flynn, B. (2024). "Detecting Halyomorpha halys using a low-power edge-based monitoring system," *Computers and Electronics in Agriculture*, 221, 108935, DOI: 10.1016/j.compag.2024.108935.
2. **Kargar, A.**, Wilk, M. P., Zorbas, D., Gaffney, M. T., & O'Flynn, B. (2022), "A novel resource-constrained insect monitoring system based on machine vision with edge AI," In *2022 IEEE 5th International Conference on Image*

Processing Applications and Systems (IPAS) (pp. 1-6). IEEE, DOI: 10.1109/IPAS55744.2022.10052895.

3. Almstedt, L., Betti Sorbelli, F., Boom, B., Calvini, R., Costi, E., Dinca, A., Ferrari, V., Giannetti, D., Ichim, L., **Kargar, A**., et al., (2024), "Beyond the naked eye: Computer vision for detecting brown marmorated stink bug and its punctures," *IEEE Transactions on AgriFood Electronics*, vol. 3, no. 1, pp. 6-17, DOI: 10.1109/TAFE.2024.3429537.

Dataset:

4. **Kargar, A.**, Zorbas, D., Gaffney, M., O'Flynn, B., & Tedesco, S. (2024). Image Dataset of Brown Marmorated Stink Bug (BMSB) or Halyomorpha Halys (HH)-V1 [Data set], *Zenodo*, DOI: 10.5281/zenodo.7887046.

Chapter 4: This chapter meets the second objective of this PhD thesis by developing and validating a novel TinyDL model for deployment on resource-constrained MCUs for insect segmentation and counting.

In this chapter, the thesis shifts toward optimized DL model design. The model was built on the real-world data obtained using the device introduced in the previous chapter. A lightweight DL model, named SemiY-Net, was proposed for simultaneous object segmentation and counting. The model was designed to meet the strict memory and computational constraints of MCUs while providing accurate outputs. Several strategies were employed to achieve this balance, including architectural pruning, layer-level memory profiling, and skip connection management. Unlike typical segmentation models designed for high-resource platforms, SemiY-Net introduces a design that deliberately reduces spatial resolution at intermediate layers and outputs. This results in a trade-off of accuracy in favor of deployment feasibility. The model was trained and validated on a dataset of annotated insect trap images for both segmentation and insect counting. In addition to its accuracy, it runs entirely on an MCU-based board without reliance on a server or cloud.

The main contributions of this chapter are as follows:

- SemiY-Net, a super lightweight dual-output DL model suitable for resource-constrained platforms such as MCUs, is proposed for image

segmentation and object (e.g., the target insect) counting by optimizing the RAM usage and the model size.

- A dataset for the HH insect was created using trap images captured in an orchard infested by the insect. The images were annotated in two different ways, which makes the dataset suitable for both detection (bounding box) and segmentation tasks (mask).
- The model is evaluated from several aspects, including accuracy metrics to evaluate its segmentation and counting performance, and implementation metrics to investigate its implementation possibility on resource-constrained platforms such as MCUs. Moreover, the proposed SemiY-Net model was implemented and validated on an MCU-based board to evaluate its power consumption and inference time.

The work in this chapter is adapted from the first author peer-reviewed articles as follows:

1. **Kargar, A.**, Zorbas, D., Gaffney, M., O'Flynn, B., & Tedesco, S. (2025), "Tiny deep learning model for insect segmentation and counting on resource-constrained devices," *Computers and Electronics in Agriculture*, 236, 110378, DOI: 10.1016/j.compag.2025.110378.
2. **Kargar, A.**, Zorbas, D., Gaffney, M., O'Flynn, B., & Tedesco, S. (2024), "Y-Net: Insect Counting and Segmentation using Deep Learning on Embedded Devices," In *2024 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)* (pp. 1-6). IEEE, DOI: 10.1109/I2MTC60896.2024.10561007.

Dataset:

3. **Kargar, A.**, Zorbas, D., Gaffney, M., O'Flynn, B., & Tedesco, S. (2024). Image Dataset of Brown Marmorated Stink Bug (BMSB) or Halyomorpha Halys (HH)-V2 [Data set], *Zenodo*, DOI: 10.5281/zenodo.14051319.

Chapter 5: This chapter addresses the third objective of this PhD thesis by developing and validating a practical and comprehensive edge-based insect monitoring framework to address data scarcity and post-deployment data drift.

In this chapter, a self-improving and fully edge-based framework was developed where MCU-based edge nodes are supported by an edge server (e.g., Raspberry Pi) that handles DL model retraining and model adaptation. The system incorporates a pseudo-labeling pipeline that benefits from contrastive feature extraction and clustering to generate segmentation masks for unlabelled data collected during deployment. These pseudo-labelled samples are then used to periodically retrain the model using an incremental learning strategy which allows it to adapt to changing environmental conditions or data distributions without human involvement. Moreover, communication between nodes and the server is minimized by transmitting only important data over low-bandwidth links such as LoRa. This setup eliminates the need for cloud access or power-hungry platforms and makes it suitable for long-term deployment.

The key contributions of this chapter are as follows:

- An efficient self-supervised incremental learning framework is proposed for a resource-constrained IoT insect counting and segmentation system, allowing automatic operation without human involvement, by distributing inference to edge nodes and retraining tasks to a nearby edge server.
- An efficient pseudo-labeling approach is proposed which combines a DL model, trained using a triplet loss function, and K-Means clustering to adapt the model to potential changes in input data.
- An efficient strategy to transfer only important regions of image, to reduce the communication load between edge devices, and enable automatic image reconstruction and mask generation on the edge server.

The work in this chapter is adapted from the first author peer-reviewed publications as follows:

1. **Kargar, A.**, Zorbas, D., Gaffney, M., O'Flynn, B., & Tedesco, S. (2025), "Self-Supervised Incremental Learning for Insect Segmentation and Counting on Resource-Constrained Devices," *IEEE Internet of Things Journal,* DOI: 10.1109/JIOT.2025.3630062.
2. **Kargar, A.**, Zorbas, D., Gaffney, M., O'Flynn, B., & Tedesco, S., "Concept Drift Mitigation on Resource-Constrained IoT Devices via Self-Learning,"

in *IEEE Sensors Applications Symposium (SAS)*, Newcastle, UK, Jul. 2025, DOI: 10.1109/SAS65169.2025.11105109.

Dataset:

3. **Kargar, A.**, Zorbas, D., Gaffney, M., O'Flynn, B., & Tedesco, S. (2025). Image Dataset of Brown Marmorated Stink Bug (BMSB) or Halyomorpha Halys (HH)-V3 [Data set], *Zenodo*, DOI: 10.5281/zenodo.16088064.

Chapter 6: Finally, this thesis is concluded by presenting the main conclusions, impact and findings of this research. In addition to summarizing the findings, the chapter acknowledges the limitations, and finally, suggests future research work in this research space.

1.6 Publication list

Publications that form part of this thesis are as follows:

Journal papers (First-Author Publications):

1. **Kargar, A.**, Zorbas, D., Gaffney, M., O'Flynn, B., & Tedesco, S. (2025), "Self-Supervised Incremental Learning for Insect Segmentation and Counting on Resource-Constrained Devices," *IEEE Internet of Things Journal,* DOI: 10.1109/JIOT.2025.3630062.
2. **Kargar, A.**, Zorbas, D., Gaffney, M., O'Flynn, B., & Tedesco, S. (2025), "Tiny deep learning model for insect segmentation and counting on resource-constrained devices," *Computers and Electronics in Agriculture*, 236, 110378, DOI: 10.1016/j.compag.2025.110378.
3. **Kargar, A.**, Zorbas, D., Tedesco, S., Gaffney, M., & O'Flynn, B. (2024). "Detecting Halyomorpha halys using a low-power edge-based monitoring system," *Computers and Electronics in Agriculture*, 221, 108935, DOI: 10.1016/j.compag.2024.108935.

Conference papers (First-Author Publications):

4. **Kargar, A.**, Zorbas, D., Gaffney, M., O'Flynn, B., & Tedesco, S. (2025) "Concept Drift Mitigation on Resource-Constrained IoT Devices via Self-

Learning," In *2025 IEEE Sensors Applications Symposium (SAS)*, Newcastle, UK, Jul. 2025, DOI: 10.1109/SAS65169.2025.11105109.

5. **Kargar, A.**, Zorbas, D., Gaffney, M., O'Flynn, B., & Tedesco, S. (2024), "Y-Net: Insect Counting and Segmentation using Deep Learning on Embedded Devices," In *2024 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)* (pp. 1-6). IEEE, DOI: 10.1109/I2MTC60896.2024.10561007.
6. **Kargar, A**., Wilk, M. P., Zorbas, D., Gaffney, M. T., & O'Flynn, B. (2022), "A novel resource-constrained insect monitoring system based on machine vision with edge AI," In *2022 IEEE 5th International Conference on Image Processing Applications and Systems (IPAS)* (pp. 1-6). IEEE, DOI: 10.1109/IPAS55744.2022.10052895.

Dataset

7. **Kargar, A.**, Zorbas, D., Gaffney, M., O'Flynn, B., & Tedesco, S. (2024). Image Dataset of Brown Marmorated Stink Bug (BMSB) or Halyomorpha Halys (HH) [Data set], *Zenodo*.

    - Version 1: https://zenodo.org/records/7887046
    - Version 2: https://zenodo.org/records/14051319
    - Version 3: https://zenodo.org/records/16088064

In addition to these publications, several other publications have been produced as a result of this research project, as follows:

Journal Paper (Co-Authored Publications):

8. Almstedt, L., Betti Sorbelli, F., Boom, B., Calvini, R., Costi, E., Dinca, A., Ferrari, V., Giannetti, D., Ichim, L., **Kargar, A.,** Lazar, C., Maistrello, L., Navarra, A., Niederprüm, D., Offermans, P., O'Flynn, B., Palazzetti, L., Patelli, N., Pinotti, C.M., Popescu, D., Rangarajan, A.K., Serghei, L., Ulrici, A., Wolf, L., Zorbas, D., Zurek, L., (2024), "Beyond the naked eye: Computer vision for detecting brown marmorated stink bug and its punctures," *IEEE Transactions on AgriFood Electronics*, vol. 3, no. 1, pp. 6-17, DOI: 10.1109/TAFE.2024.3429537.

9. Almstedt, L., Betti Sorbelli, F., Boom, B., Calvini, R., Costi, E., Dinca, A., Ferrari, V., Giannetti, D., Ichim, L., **Kargar, A.,** Lazar, C., Maistrello, L., Navarra, A., Niederprüm, D., Offermans, P., O'Flynn, B., Palazzetti, L., Patelli, N., Pinotti, C.M., Popescu, D., Rangarajan, A.K., Serghei, L., Ulrici, A., Wolf, L., Zorbas, D., Zurek, L., (2024), "A Comprehensive Pest Monitoring System for Brown Marmorated Stink Bug," *IEEE Transactions on AgriFood Electronics*, vol. 3, no. 1, pp. 110-120, DOI: 10.1109/TAFE.2024.3469538.

Conference papers (Co-Authored Publications)

10. Khamitov, R., **Kargar, A.**, Azamat, M., Ristin, M., O'Flynn, B., & Zorbas, D. (2025), "Incremental Learning of Image Classification Models Over Low Data-Rate Networks," In *2025 21st International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)* (pp. 1-8). IEEE, DOI: 10.1109/DCOSS-IoT65416.2025.00030.

11. Zarymkanov, T., **Kargar, A.**, Pinotti, C. M., O'Flynn, B., & Zorbas, D. (2023), "Enhancing Machine Learning Training Performance in Smart Agriculture Datasets Using a Mobile App", In *2023 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor)* (pp. 455-460). IEEE, DOI: 10.1109/MetroAgriFor58484.2023.10424407.

12. Zurek, L. J., **Kargar, A.**, O'Flynn, B., Niederprüm, D., Wolf, L., & Zorbas, D. (2023), "Evaluation of wireless technologies for an embedded camera-based pest monitoring system," In *2023 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor)* (pp. 449-454). IEEE, DOI: 10.1109/MetroAgriFor58484.2023.10424396.

13. Almstedt, L., Baltieri, D., Betti Sorbelli, F., Cattozzi, D., Giannetti, D., **Kargar, A.,** Maistrello, L., Navarra, A., Niederprüm, D., O'Flynn, B., Palazzetti, L., Patelli, N., Piccinini, L., Pinotti, C.M., Wolf, L., Zorbas, D., (2023), "Technological innovations in agriculture for scouting halyomorpha halys in orchards" In *2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things*

(DCOSS-IoT) (pp. 702-709). IEEE, DOI: 10.1109/DCOSS-IoT58021.2023.00110.

1.7 Achievements and Recognition

The research was supported through a prestigious Teagasc Walsh Scholarship[3], awarded only to candidates conducting nationally significant agri-tech research, thereby recognizing its potential impact in smart agriculture and AI-driven pest monitoring.

Furthermore, one of the resulting papers was named Best Paper in the session Image Processing and Biological Applications at the 2022 IEEE 5th International Conference on Image Processing Applications and Systems (IPAS), Genova, Italy.

Moreover, the developed device and the project were also featured in the television program "10 Things to Know About[4]" on RTE and will be broadcast in the next episode of this series. Moreover, the work received national recognition through coverage on RTÉ News[5,6].

Finally, and most notably, the work was awarded the "Best Application of AI in a Student Project" at the 2025 AI Awards Ireland[7].

[3] https://teagasc.ie/about/research-innovation/the-walsh-scholars-programme/
[4] https://10thingstoknowabout.ie/
[5] https://www.tyndall.ie/news/rte-news-highlights-tyndalls-ai-research-to-protect-crops/
[6] https://www.rte.ie/news/ireland/2025/1117/1544305-irish-ai-climate/
[7] https://aiireland.ie/2025/11/18/ai-awards-celebrates-irish-innovation-in-2025/

1.8 References

[1] WHO; FAO; OIE; UNEP, "The future of food and agriculture - Trends and challenges," *United Nations Human Settlements Programme (UN-Habitat): Addis Ababa, Ethiopia. (2021)*, p. 52, 2017, Accessed: Nov. 13, 2025. [Online]. Available: https://openknowledge.fao.org/handle/20.500.14283/i6583e

[2] "About | Plant Production and Protection | Food and Agriculture Organization of the United Nations." Accessed: Oct. 15, 2025. [Online]. Available: https://www.fao.org/plant-production-protection/about/en

[3] D. R. Paini, A. W. Sheppard, D. C. Cook, P. J. De Barro, S. P. Worner, and M. B. Thomas, "Global threat to agriculture from invasive species," *Proc Natl Acad Sci U S A*, vol. 113, no. 27, pp. 7575–7579, Jul. 2016, doi: 10.1073/PNAS.1602205113/SUPPL_FILE/PNAS.1602205113.SAPP.PDF.

[4] J. M. Gippet, A. M. Liebhold, G. Fenn-Moltu, and C. Bertelsmeier, "Human-mediated dispersal in insects," *Curr Opin Insect Sci*, vol. 35, pp. 96–102, Oct. 2019, doi: 10.1016/J.COIS.2019.07.005.

[5] A. C. Teixeira, J. Ribeiro, R. Morais, J. J. Sousa, and A. Cunha, "A Systematic Review on Automatic Insect Detection Using Deep Learning," *Agriculture*, vol. 13, no. 3, p. 713, Mar. 2023, doi: 10.3390/AGRICULTURE13030713.

[6] J. She *et al.*, "A method for automatic real-time detection and counting of fruit fly pests in orchards by trap bottles via convolutional neural network with attention mechanism added," *Ecol Inform*, vol. 70, p. 101690, Sep. 2022, doi: 10.1016/J.ECOINF.2022.101690.

[7] A. Bereciartua-Pérez, L. Gómez, A. Picón, R. Navarra-Mestre, C. Klukas, and T. Eggers, "Insect counting through deep learning-based density maps estimation," *Comput Electron Agric*, vol. 197, p. 106933, Jun. 2022, doi: 10.1016/J.COMPAG.2022.106933.

[8] W. Albattah, M. Masood, A. Javed, M. Nawaz, and S. Albahli, "Custom CornerNet: a drone-based improved deep learning technique for large-scale multiclass pest localization and classification," *Complex and Intelligent Systems*, vol. 9, no. 2, pp. 1299–1316, Apr. 2023, doi: 10.1007/S40747-022-00847-X/TABLES/8.

[9] A. Passias *et al.*, "Insect Pest Trap Development and DL-Based Pest Detection: A Comprehensive Review," *IEEE Transactions on AgriFood Electronics*, vol. 2, no. 2, pp. 323–334, Aug. 2024, doi: 10.1109/TAFE.2024.3436470.

[10] J. Bremmer, A. Gonzalez-Martinez, R. Jongeneel, H. Huiting, R. Stokkers, and M. Ruijs, *Impact assessment of EC 2030 Green Deal Targets for sustainable crop production*, no. 2021–150. Wageningen Economic Research, 2021.

[11] T. Dlamini and S. Vilakati, "Remote and Rural Connectivity: Infrastructure and Resource Sharing Principles," *Wirel Commun Mob Comput*, vol. 2021, no. 1, p. 6065119, Jan. 2021, doi: 10.1155/2021/6065119.

[12] E. Yaacoub and M. S. Alouini, "A Key 6G Challenge and Opportunity-Connecting the Base of the Pyramid: A Survey on Rural Connectivity," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 533–582, Apr. 2020, doi: 10.1109/JPROC.2020.2976703.

[13] M. Vaezi *et al.*, "Cellular, Wide-Area, and Non-Terrestrial IoT: A Survey on 5G Advances and the Road Toward 6G," *IEEE Communications Surveys and Tutorials*, vol. 24, no. 2, pp. 1117–1174, 2022, doi: 10.1109/COMST.2022.3151028.

[14] J. Liu and X. Wang, “Plant diseases and pests detection based on deep learning: a review,” *Plant Methods 2021 17:1*, vol. 17, no. 1, pp. 1–18, Feb. 2021, doi: 10.1186/S13007-021-00722-9.

[15] F. Magistri *et al.*, “From one field to another-Unsupervised domain adaptation for semantic segmentation in agricultural robotics,” *Comput Electron Agric*, vol. 212, p. 108114, 2023, doi: 10.1016/j.compag.2023.108114.

[16] A. L. Acebes, “Host plant effects on the biology, behavior and ecology of brown marmorated stink bug, Halyomorpha halys (Stål)(Hemiptera: Pentatomidae),” *Virginia Tech*, 2016.

[17] I. Secretariat *et al.*, *Scientific review of the impact of climate change on plant pests*. FAO on behalf of the IPPC Secretariat, 2021.

[18] D. Giannetti, N. Patelli, L. Palazzetti, F. Betti Sorbelli, C. M. Pinotti, and L. Maistrello, “First use of unmanned aerial vehicles to monitor Halyomorpha halys and recognize it using artificial intelligence,” *Pest Manag Sci*, vol. 80, no. 8, pp. 4074–4084, Aug. 2024, doi: 10.1002/PS.8115.

[19] V. Ferrari *et al.*, “Evaluation of the potential of near infrared hyperspectral imaging for monitoring the invasive brown marmorated stink bug,” *Chemometrics and Intelligent Laboratory Systems*, vol. 234, p. 104751, Mar. 2023, doi: 10.1016/J.CHEMOLAB.2023.104751.

2 CHAPTER II. BACKGROUND

2.1 Introduction

This chapter presents the background and motivation of this PhD thesis and the real-world challenges it aims to address. The current state of the art in Edge AI, resource constrained image analysis systems are reviewed and research gaps identified which are addressed in the subsequent chapters. The focus is on developing an AI-powered system that can run directly on the edge in remote environments where power, connectivity, and human resources are limited.

The chapter first discusses the shift from cloud-based AI to Edge AI and explains why conventional solutions are not suitable for remote deployment. It then looks at image-based models and narrows the focus to those that can run on resource-constrained edge devices. Afterwards, the focus shifts to the specific application of insect monitoring in orchards, which provides a practical and well-motivated test case to validate edge AI systems under the real-world conditions and challenges of remote areas. Finally, recent relevant studies are reviewed and the gaps and challenges are identified for this application, which provides the motivation and is the source of hypotheses of this PhD thesis.

2.2 Moving from cloud to edge AI

The rapid development of Artificial Intelligence (AI) in recent decades has revolutionized different fields and domains ranging from computer science and engineering to digital marketing, social science and agriculture [1]. These AI-based systems typically rely on cloud-based infrastructure established in highly resourced data centers where the raw data is transferred from the end user to the cloud for analysis and then the corresponding results are provided back to the end user. Cloud computing effectively provides unlimited storage and computational power [2], which are essential for data analysis and AI computations, particularly when advanced AI models such as large language models (LLMs), transformer architectures, or deep learning (DL) models are used. Despite these advantages, cloud-based systems face several issues that limit their effectiveness in modern applications. These include

privacy and security concerns, latency, network bandwidth and reliability [3][4][5]. For instance, autonomous driving applications rely heavily on low-latency responses, while surveillance systems often require hundreds of gigabits per second of bandwidth to transmit high-quality video data [3][6]. Moreover, such systems are not reliable in areas with poor communications network coverage since limited or unstable connectivity impacts the ability to transmit data, causing delays in uploads, potential data loss during transmission, and reduced real-time availability, which is critical for time-sensitive applications.

To address these challenges, fog computing, where computation and related services are moved closer to the end user or the source of data by using the existing communication network and infrastructure, can provide some solutions. Fog computing approaches enable local processing to be carried out near the data source, which improves performance, particularly in terms of response time. However, its distributed nature and reliance on the network infrastructure are considered as key drawbacks [4].

These challenges have led to the emergence of edge computing research opportunities where computation is shifted closer to the data source where the data is captured by end devices. Unlike fog computing, which still depends on intermediate nodes in the network, edge computing performs all required computations locally on the device itself. This eliminates the need to transfer data elsewhere and removes the dependency on network infrastructure. This improves response time and privacy, conserving bandwidth, and reduces dependency on network and powerful communication infrastructure. As a result, this approach reduces power consumption and costs (e.g., CAPEX and OPEX) by lowering communication and cloud expenses and energy usage, which consequently makes it a good choice for sensor deployments in remote areas with poor network coverage. Edge-based systems have been used in many applications in recent years across many research and commercial sectors. One of the key areas where they have been widely studied and applied is healthcare where they are used for real-time monitoring, early detection of abnormalities, and enhanced data privacy [7][8][9]. Smart cities and smart homes are other examples where edge devices are used for monitoring and control purposes, such as energy management, environmental sensing, and security [10][11][12]. Manufacturing and industrial settings also benefit from edge computing to improve operational efficiency [13][14].

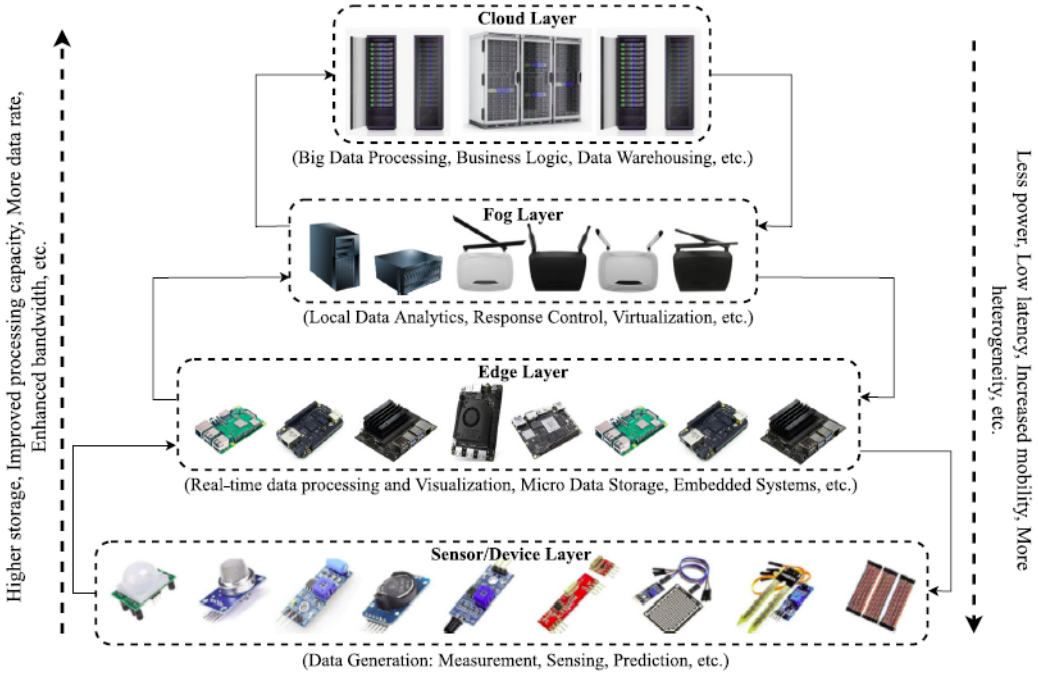


Figure 2.1. Taxonomy of cloud, fog and edge frameworks. (source:[4])

These examples demonstrate the feasibility and potential of edge-based systems in real-world scenarios.

The general taxonomy of the mentioned cloud, fog and edge frameworks is shown in Figure 2.1 [4]. In the figure, when moving from cloud to edge computing approaches, available resources (processing, bandwidth, and power) decrease, but this shift results in improvements in response time, power consumption, and bandwidth requirements which are critical factors for many real-world applications. Edge-based systems, despite all their advantages, can be highly impacted by the constrained resources of the edge devices, specifically in terms of computational power and memory storage [6]. Therefore, it is necessary to develop optimized AI models that can operate within the limited computational and memory resources available on edge devices.

2.3 Optimized AI models for edge environments

Edge devices are generally equipped with limited computational power and storage, which makes running AI models on them challenging. Therefore, AI models need to be lightweight and optimized to run effectively according to the resource constraints of edge devices on which they are running. In this thesis, the specific focus is on developing deep learning (DL)-based models, an important and powerful subset of AI, due to their suitability and effectiveness for vision-based applications.

Therefore, here is presented a brief review of DL models used in this area. AlexNet [15] was the first well-known DL model that started the modern DL era in 2012. It achieved around 57% Top-1 accuracy on the ImageNet [16] dataset. After that, several models were proposed including VGGNet [17], ResNet [18], Inception [19], DenseNet [20], and NASNet [21] where their main focus was on improving accuracy. For instance, NASNet reached up to 82.7% accuracy on ImageNet.

As these models became deeper (more complex) and more accurate, they also became larger and more complex, thus requiring more computational and memory resources. This makes them unsuitable (and even impossible) for deployment on hardware with limited resources. Therefore, many works focus not only on accuracy but also on efficiency by reducing the size and complexity of models while maintain high levels of accuracy. One of the first models that focused on these aspects was SqueezeNet [22]. This model introduced the Fire module developed to squeeze and expand feature maps efficiently, and it achieved an accuracy level comparable to AlexNet but with approximately 50× reduction in the number of parameters and model size. The MobileNet family [23][24][25] are well-known models that balance the model complexity (reducing parameters and computation) and accuracy by replacing standard convolutions with depthwise separable convolutions, which makes them suitable and possible to run on mobile devices. Similarly, EfficientNet [26][27] proposed several models that balance accuracy, model size, and input resolution using compound scaling, each of which could be used according to the limitations of the device resources.

While models like MobileNet and EfficientNet are sufficiently efficient so as to run on modern smartphones or embedded platforms such as Raspberry Pi, they are still too complex for resource-constrained microcontroller units (MCUs), which are typically a low-cost and low-power hardware, making them an excellent fit for Internet of Things (IoT)-based systems, as will be elaborated later in Section 2.6.3. These devices typically have only a few hundred KB of RAM, and a flash memory in the range of just a few MB, while in comparison, modern smartphones have RAM and storage in the range of GB [28]. This clearly highlights that it would be highly challenging (or even impractical) to run conventional DL models on such resource-constrained devices [29]. For instance, as reported in [30], although MobileNetV2

reduces model size by 4.6x compared to ResNet-18, it requires 1.8 times more RAM to perform inference, which makes it more challenging to run on MCUs.

Although, there are several DL model compression techniques, such as knowledge distillation [31], pruning [32], and quantization [33], whose main objective is to reduce the model parameters, size or FLOPs, not all methods, including knowledge distillation and pruning, directly decrease the memory usage of a DL model; these methods lead to a smaller model size, but there is no guarantee that the model would use lower memory, thus the model might still have the same memory usage at inference time [34]. However, quantization, which is widely used for model size reduction, can be an effective method that also reduces memory usage by representing weights and activations in lower precision (e.g., int8 instead of float32) but it is still insufficient to meet the strict resource constraints of MCUs on its own.

TinyML [35], which is in fact the intersection of AI and resource-constrained embedded systems, has emerged to focus specifically on the development of efficient AI models on resource-constrained devices such as MCUs, as shown in Figure 2.2 [35].

MCUNet [30] and MCUNetV2 [34] are examples of tiny models which were specifically designed for MCUs to meet their strict resource limitations. For example, MCUNetV2 achieved 71% Top-1 accuracy on ImageNet dataset running on an MCU with less than 1 MB of flash.

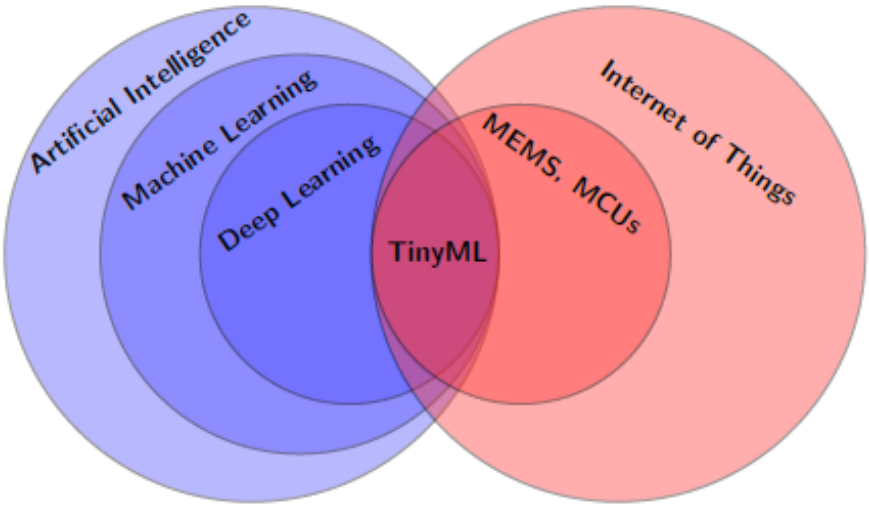


Figure 2.2. TinyML is the intersection of AI and resource-constrained embedded systems. (Source: [35])

Besides such general tiny models, there are also custom models designed and proposed for specific applications. These are often developed based on the requirements, constraints, and deployment considerations, such as input resolution, acceptable delay, available memory, and task (e.g., classification, object detection, segmentation). Designing such models by considering these factors can lead to a better performance and efficiency than using off-the-shelf models, with examples such as [36][37][38].

However, deploying systems that benefit from these optimized DL models in remote areas with strict limitations, such as a lack of power lines, unreliable communication infrastructure, and limited or costly human access, remains a significant challenge.

2.4 Challenges and limitations

Deploying AI-based systems in remote areas, where resources and facilities are limited, adds more complexity to system design, specifically to the AI component. Some of the most critical limitations of remote areas include power constraints, unreliable connectivity, and limited or costly human access [39][40][41], all of which directly affect system design.

The primary and often one of the most critical challenges associated with deploying sensor systems in remote areas is energy availability. These areas typically lack power grids or other sources of electricity, therefore any deployed device must rely only on batteries as their power supply. A long battery life is essential, since battery replacements require significant maintenance, cost and human intervention. This can be particularly challenging in remote areas, often with limited access to devices [41]. As a result, the system and its design must be optimized for low power consumption to ensure long-term operation and reduce the need for human intervention to replace batteries. In addition, the system can be equipped with energy harvesting solutions, such as solar panels, to recharge the batteries and extend its lifetime [41] and reduce maintenance requirements. Therefore, both the hardware and the software (including the AI) should be designed for low-power operation. From a hardware perspective, MCUs are good options for the main processor due to their low-power processing characteristics. They are also low-cost, which is another desirable factor, especially for IoT-based systems where many nodes may need to be deployed

in the target environment. Despite these key advantages, as mentioned earlier, MCUs suffer from strict limitations in memory and computational capacity which makes it challenging to run complex AI algorithm on them.

Another significant limitation associated with remote sensor deployments is connectivity. IoT-based systems are typically required to transfer data to the cloud or a server, generally using wireless networks [41]. Communication technologies can be divided into two main categories based on coverage range: short-range and long-range. Short-range technologies, such as Bluetooth, Bluetooth Low Energy (BLE) or even Wi-Fi, are mainly used to connect devices over short distances, typically in the range of meters, for example, in smart home and wearable devices [42][43]. In contrast, long-range technologies such as 4G, 5G, and LoRa are suitable for communication over longer distances, generally in the range of kilometres. Apart from coverage range, other important factors are also considered during system design (i.e., depending on the application and its requirements) such as power consumption, data rate, and cost. For remote areas, long-range technologies such as 5G may be suitable, however, the required infrastructure to support such networks is often unavailable in remote areas which makes deployment in these regions more challenging [40].

The complexity of these deployments requires practical and well-motivated test cases to validate edge AI systems under real-world conditions. In this context, insect monitoring is a challenging and relevant use case since it combines many of the technical and operational challenges discussed so far.

2.5 Insect monitoring as a realistic and demanding application

Pest insects have a major impact on global food production, causing yield losses and reducing the quality of fruits and vegetables. Ensuring food security will become a significant challenge as the world's population continues to grow; the human population is estimated to increase to nearly nine billion people by 2025 [44].

Moreover, Food and Agriculture Organization of the United Nations (FAO) reported that up to 40 percent of global crop production is lost annually, which costs about $220 billion from plant diseases and at least $70 billion from invasive insects [45]. Therefore, insect pests are one of the main reasons for crop production reduction

that threaten human food security, and these need to be carefully monitored and controlled. Moreover, the situation is expected to worsen as invasive species spread into countries where certain pests were absent previously. Events such as climate change, globalization and global trade will contribute to this spread as environmental conditions change around the world [46].

Using pesticides is the first strategy commonly used by agronomists to control insect populations in their fields, but this approach can have harmful side effects on human and animal health, biodiversity and the environment [47][48]. It is thus important to use pesticides appropriately to effectively control insect pest populations, while also minimizing the pesticides environmental impacts. The implementation of an Integrated Pest Management (IPM) strategy aims to address the impact of invasive insect pests on crops in a human- and eco-friendly and sustainable way. This strategy suggests using multiple techniques to control pests including cultural, mechanical/physical, biological, and chemical control [49]. Monitoring is an important part of IPM strategies that can use the monitoring information to specify the timing of insecticide applications, identification of pest species, and information on the size of the insect population. The monitoring data helps growers to take an informed decision such as the correct insect control strategy to use and choose the best possible controlling method accordingly [50]. Therefore, early insect detection is critical since this could improve the potential for control of these pests and reduces the overall level of crop damage while also reducing the impact of pesticide usage on food quality and the environment [51].

Traditionally, growers use commercial sticky traps in their crops to attract target insect pest species using pheromone lure or the trap colour as shown in Figure 2.3. Traditionally, farmers have had to physically monitor their orchards and manually inspect the traps and count the insect population and species on a regular basis (e.g., daily or weekly) with an associated increase in fuel consumption and emissions. This can be problematic for large-scale orchards due to the amount of time, effort, and manpower cost it requires [52]. In addition, sometimes visually identifying insect species with the human eye can be difficult because of their high visual similarity [53].

Figure 2.3. Farmers inspecting for insects on sticky traps in orchards, illustrating the labor-intensive nature of traditional insect monitoring[8].

These factors can increase the error in the establishment of the estimated insect population and lead to incorrect use of pesticides. Therefore, devoting significant time and resources to this manual monitoring procedure may not always result in accurate results. Approaches such as the presence-absence model (PAM) [54] have been suggested to count a subset of insects to assess their populations, but these non-automated, in-person inspections are still inefficient, especially for large-scale remote fields typical on this industry [55].

With the recent progress of information and communication technology (ICT) and Artificial Intelligence (AI), Precision Agriculture (PA) has emerged as a promising approach to improve crop production and agricultural sustainability [47][56]. An automated insect monitoring system is a critical aspect of PA that aims to optimize the insect population estimation task. This reduces the complexity of the monitoring task, which is currently performed manually by growers [57]. Moreover, the use of such systems reduces the need for in-person visits, resulting in a reduction in fuel consumption, emissions, costs, and time [58]. Automated insect monitoring also supports the implementation of the EU's ambition to reach a 50% reduction in pesticide usage by 2030 [59].

The automation of insect monitoring has widely been investigated in recent years, using image processing and machine vision [60]. Such systems employ camera sensors in orchards capable of taking images of crops or plants. The generated images are then analyzed for insect detection and classification. However, there are several

[8] https://ncipmhort.cfans.umn.edu/integrated-pest-management-ipm
https://onapples.com/blog/the-fight-against-apple-pests.php

challenges in the development of automation processes that should be considered during the design of such a system, as summarised below:

- **limited or costly human access:** Orchards are generally located in remote areas, and visiting them frequently for maintenance tasks would be inefficient in terms of cost, labour, time, fuel consumption, and environmental impact due to emissions. Therefore, minimizing the need for on-site human intervention is critical for sustainable system design.
- **Power consumption:** Orchards generally have no power lines, hence the monitoring system should be low-power and last for extended periods without the need for battery maintenance.
- **Communication of the results:** There may be a lack of reliable internet connectivity and communication infrastructure since orchards are outside of cities in remote areas.
- **Accuracy:** The system should be accurate enough so that farmers can trust its information and make appropriate decisions accordingly.
- **Dynamic environment:** The environmental conditions in orchards are highly variable, and the characteristics of each orchard can differ significantly from others. For example, plant and insect species diversity, weather and lighting conditions, and background complexity introduce challenges for the performance of vision-based systems.
- **Cost of purchase and maintenance:** The cost of such a system should be as low as possible. If such a system has a high cost, farmers are not likely to use it since many of them should most likely be required to be deployed in orchards. Moreover, the maintenance cost should also be as low as possible by using commercially available consumables (e.g., traps).
- **Ease of deployment and maintenance:** The system must not require any prior knowledge of electronics or information technology processes for the farmers.

All these considerations and challenges make insect monitoring a realistic and demanding test case for developing and validating AI-based systems in remote areas. This application requires accurate, efficient, and adaptive solutions capable of operating in low-power, low-connectivity conditions while also handling the variability and unpredictability of real-world environments.

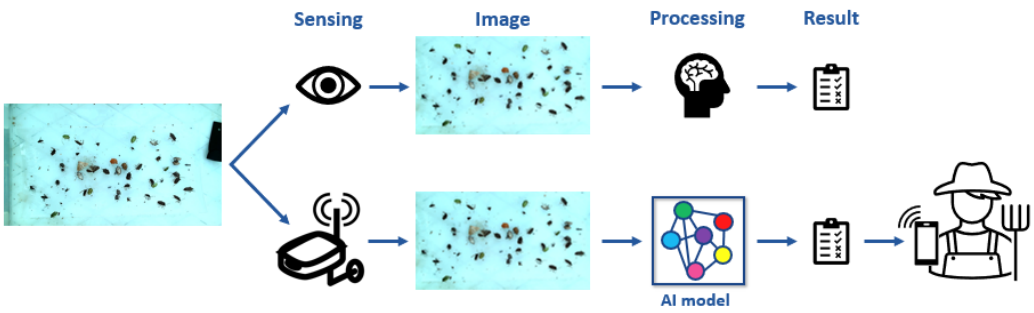


Figure 2.4. Human and AI-based system for insect detection, comparing manual inspection with automated detection using computer vision and AI methods. (Source: Adapted from [61] with modifications).

2.6 Related work for insect monitoring using AI

Smart vision-based insect monitoring systems aim to mimic the natural human process of insect identification, as illustrated in Figure 2.4 [61]. In a typical scenario, a person observes the insect trap using their eyes, then the brain processes the visual input to identify the species and count the number of insects. This approach requires humans to be trained to recognize the target insect. Similarly, in smart vision-based systems, a camera captures images of the trap surface. Then the AI component, which aims to mimic the functions of the human brain, processes the image to detect, classify and count the insects. Finally, the output is transferred to the user.

A smart vision-based system consists of two main parts: hardware and software. The hardware part includes three key modules: a camera unit to capture images, a processing unit to analyses the images, and a transmission unit to transfer the data. On the software side, the core function is the image analysis algorithm which should automatically detect, classify and count the insects [60]. Therefore, the AI component is the critical part of a smart insect monitoring system that enables automated estimation of insect population. In the following, the AI-based systems and devices that have been studied for this application and highlight their main strengths and limitations are reviewed.

2.6.1 Literature review methodology

This literature review followed a narrative and exploratory approach to identify relevant work on automated vision-based AI-powered insect monitoring and hardware supported systems used in real-world conditions. All publications were sourced through Google Scholar which provides broad coverage of peer-reviewed literature.

The search process involved keyword combinations where the brackets group similar keywords such as {“insect”, “pest”}, {“monitoring”, “detection”, “identification”, “segmentation”, “counting”}, {“smart”, “automated”}, {“system”, “device”}, {“artificial intelligence”, “machine learning”, “deep learning”, “AI”, “ML”, “DL”, “TinyML”}, {“edge AI”, “embedded AI”, “on-device AI”}, and {“cloud”, “server”, “edge”, “in-field”}. Because the target insect in this study is the Brown Marmorated Stink Bug (*Halyomorpha halys*), additional searches included keywords such as “*Halyomorpha halys*”, “Brown Marmorated Stink Bug”, and “BMSB” to ensure that species-specific studies were captured.

Relevant publications were selected based on title and abstract screening, followed by full-text assessment of studies matching the scope of the review. Moreover, references of key publications were also examined to identify additional related work. Particular emphasis was placed on DL-based algorithm and studies describing devices (e.g., hardware implementation), including both field-deployed imaging devices that transmit data to cloud or server platforms, and systems performing on-device processing at the edge on embedded or low-power hardware.

Studies were included if they:

- Presented DL-based methods for insect detection, identification, and counting.
- Described a hardware component such as IoT imaging units, embedded devices, or low-power processors.
- Focused on the target insect of this study.

Studies were excluded if they focused solely on biological entomology, lacked any hardware or deployment aspect, or relied exclusively on cloud processing without describing the deployed device.

For clarity, the reviewed literature is organised into two main categories:

- ***Cloud- or server-based systems:*** Studies where image capture is performed by a device deployed in the field, while the main processing is carried out on a remote server or cloud platform.

- ***Edge-based systems:*** Studies where AI models are executed locally on embedded or resource-constrained hardware within the field environment.

2.6.2 Cloud- or server-based systems

In recent years, several studies have employed a system running on the cloud or a powerful server for image analysis. In these systems, raw images are captured using a device deployed in the field and then transmitted to the cloud or a remote server for image analysis. For example, Guo et al. [62] proposed an automatic system equipped with a solar panel to monitor flying vegetable insects with an RGB camera and a You Only Look Once (YOLO)-based detection algorithm [63] named YOLO-SIP (YOLO for Small Insect Pests). The system periodically captures images from yellow sticky traps and sends them to a cloud server over a wireless network for insect identification and classification. The proposed YOLO-SIP achieved a mean average precision (mAP) of 84.22%, an average mean absolute error (MAE) of 0.422, and an average mean squared error (MSE) of 1.126 on five target insects. Likewise, other studies [64][65] used a similar image acquisition and collection strategy and did the image processing offline on a remote server.

Tang et al. [66] focused on butterfly segmentation in ecological images and introduced a dilated encoder network (DE-Net) capturing more high-level features to generate high-resolution binary output that segmented the butterfly from the input images. The authors trained and tested the proposed DE-Net on a GPU-based system which achieved improved performance in terms of segmentation metrics, model size, and segmentation time compared to some widely used segmentation models including U-net, SegNet, FCN, and DeepLabv3+. In another study, Teixeira et al. [67] suggested segmentation by the U-Net model as a pre-processing step and then using a YOLO [68] model for grape moth detection on pheromone trap images. The authors evaluated the proposed method using U-Net with two different backbones in the pre-processing phase and YOLOv5s and YOLOv8s in the detection phase. The proposed pre-processing with U-Net and YOLOv8s achieved 95% of average precision (AP), while YOLOv5s obtained 94%. Junior et al. [69] proposed an insect detection system to detect and count aphids and parasitoids from trap images in the lab. The authors utilized a Convolutional Neural Network (CNN)-based model named Mask R-CNN

with ResNet50 backbone for insect detection on a server and achieved R^2 score of 0.81 and 0.78 for aphids and parasitoids counting compared to manual counting, respectively. The authors in [70] proposed a remote insect trap monitoring approach using four-layer IoT (comprising perception layer, transport layer, processing layer, and application layer) to construct the remote trap insect monitoring system. In this system, captured images were transferred to a remote server for insect detection and classification using Faster R-CNN with ResNet50 backbone for feature extraction.

In addition, there are a number of studies such as [71][72][73][74][75][76] which their main focus was on accuracy of the detection and recognition algorithms and as such are not concerned about the hardware, cost and other factors that are important in the considered application space, i.e. agricultural settings. In fact, factors such as power consumption, connectivity limitations, and cost are equally important, as the system must operate autonomously in remote outdoor environments for extended periods. As a result, studies that focus only on the accuracy aspect do not address the practical constraints of real-world agricultural monitoring.

The insect of interest in this study (HH) has been the subject of some studies recently. Sorbelli et al. [77] created a dataset of HH using images captured by drones and devices such as smartphones. The image processing was performed offline on a remote system. In order to improve the quality of the dataset samples, a preliminary screening process was conducted by removing images for blurriness and brightness issues. Subsequently, different versions of YOLO-v5 were trained on the dataset and analyzed using several metrics for HH detection. Palazzetti et al. [78] introduced a dataset for the target insect using autonomously captured aerial telephotos. The authors trained and evaluated several state-of-the-art DL models including YOLOv5, YOLOv8, RetinaNet, and Faster-RCNN on the dataset to detect HH on aerial images. They also designed a custom Android application to handle drone inspections of the orchard and detect HH on captured images (upon user request) by transferring the image to a server equipped with trained models. These two studies build upon the authors' earlier works. In this regard, the use of drones, the creation of an insect dataset, and the preliminary results were previously reported in [79][80][81][82][83][84][85][86]. It should be noted that these works were carried out within the HALY.ID project by collaborative partners.

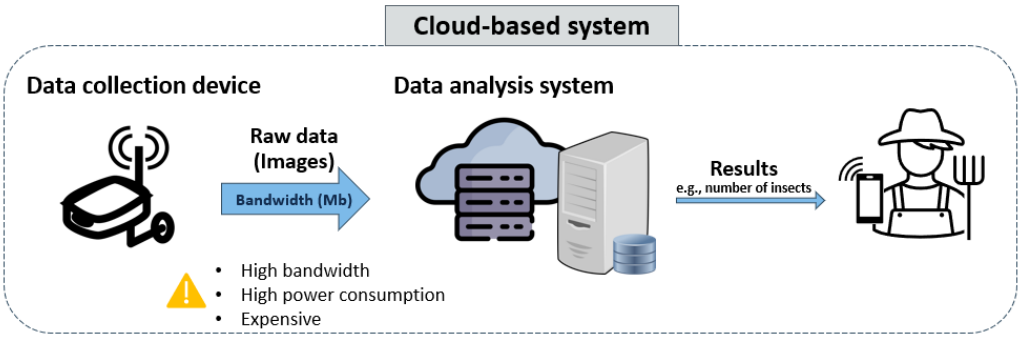


Figure 2.5. A typical cloud-based systems pipeline where captured images are transmitted to a remote cloud server for processing and analysis.

These studies are based on images captured on-site from infested orchards by HH. Since HHs has a brownish colour, it can blend easily with trees, branches, and trunks in orchards. Consequently, to detect insects in such images, a more complex and bigger DL model is needed that requires powerful hardware.

Within this line of research, investigators proposed cloud or server-based systems that rely on powerful and power-hungry servers or computers, with the primary focus being on achieving the highest model accuracy without concern for computational power, memory usage, or energy consumption. Figure 2.5 depicts a typical cloud or server-based system, where the data acquisition device, deployed in orchards, first captures data. These raw data are then transferred to a powerful cloud system or server to run the AI component and analyse the data. Finally, the results are returned to the users. Such systems depend on a reliable internet connection or a high-bandwidth communication setup to transfer raw captured images to the cloud or server. Therefore, this is not a practical approach and can cause issues in remote deployments, where such infrastructure is often unavailable. Moreover, this approach increases system cost and energy consumption, which is not in line with the requirements of this application, as discussed in the previous subsection.

2.6.3 Edge-based systems

More recently, edge-based systems have been considered for the insect monitoring task to tackle the issues of cloud-based systems as described above. Figure 2.6 illustrates an edge-based system in which the cloud/server is removed, and all the computation and analysis are carried out on the data acquisition device itself deployed in the field.

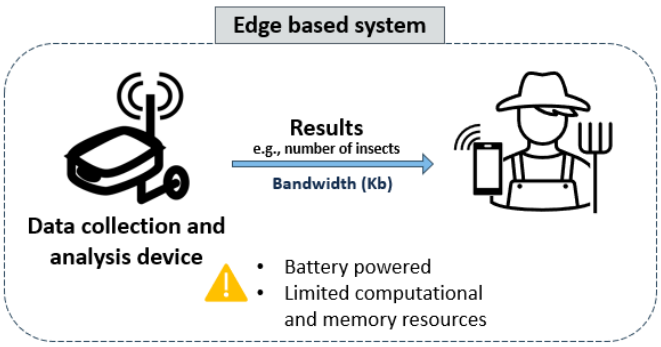


Figure 2.6. A typical edge-based systems pipeline where image processing and inference are performed locally on the edge device.

Running the image processing on the edge device allows for identifying insects from captured images at the source in real-time, and more importantly, it eliminates the need for a reliable internet connection or communication infrastructure to transfer images to a third-party platform (e.g., cloud or servers) for off-line image processing. This is a critical factor since most orchards are in remote areas that suffer from a lack of reliable connectivity as well as power lines.

Consequently, edge-based systems have attracted much attention because of their advantages over cloud- or server-based systems, including lower latency, lower data transfer volumes and network traffic, and the ability to deploy in remote locations with limited connectivity [87]. In such systems, the objective is to perform the image analysis task on the monitoring device itself in the field. However, edge devices typically have a considerably limited resource budget compared to cloud or server-based systems.

Brunelli et al. [88][89] developed an automated pest detection edge device to detect Codling Moths in a pheromone-based trap. Their system consisted of a Raspberry Pi (RPi) board for image capturing and processing. The power supply consisted of a battery and a solar panel to recharge the battery. The system detection pipeline included regions of interest (RoI) extraction and RoIs classification. For the classification part, three different Deep Neural Network (DNN) models including LeNet, VGG16, and MobileNetV2 were trained, and gained an accuracy of over 95%. In terms of power consumption, several configurations, including RPi3, RPi4 and Intel Movidius Neural Compute Stick, were analysed, and a system based on RPi3 and LeNet consumed the lowest amount of energy at 123.2J.

Bjerge et al. [90] evaluated YOLOv5 and YOLOv3 on their collected dataset and achieved 92% of AP, 94% recall, 93% F1-score, and 94.4% mAP. The authors used a RPi 3B based system to collect the dataset and applied the detection model after collecting the data but stated that the model could be run on a RPi or other similar boards for on-site prediction.

Similarly, Zhong et al. [91] proposed a YOLO-based system for counting and recognizing six different flying insect species including bees, flies, mosquitoes, chafers, moths, and fruit flies. The proposed method consisted of a YOLO model for the detection and approximate counting of insects and a Support Vector Machines (SVM) model for more accurate counting and classification. A RPi 2B was used to implement the proposed model and achieved 92.5% accuracy in counting and 90% accuracy in classification. The system required 5 minutes for the image analysis task.

Rahman et al. [92] proposed an IoT-based system for harmful insect detection, which uses a RPi to analyze captured images. They investigated several DL models, including VGG19, VGG16, ResNet50, InceptionV3, and Xception, for detecting paddy pests, with results showing that VGG19 achieved the highest accuracy at 98.91%. The system can transfer data to the cloud when an internet connection is available, and the authors also proposed a blockchain-based architecture to ensure secure data transmission.

Pham et al. [93] proposed AlerTrap which is an edge-computing insect monitoring device for remote areas. The authors used a RPi 3B board for running the insect detection and counting algorithms, and also utilized an Arduino microcontroller board to control the power supply of the system. The authors investigated the performance of three different models including Single Shot Detector (SSD)-MobilenetV1, SSD-MobilenetV2, and YOLOv4-tiny. Finally, SSD-MobilenetV2 was selected for real-time implementation based on processing time and the detection accuracy, which was over 95%. Rustia et al. [94] developed an edge-based insect pest monitoring system for mango orchards. The proposed algorithm consists of a tinyYOLOv3 and tree-based CNN classifiers. The proposed model achieved an F1-score of 96% and an average inference time of about 11 seconds on a RPi Zero W which occupied approximately 151 and 57 MB of GPU memory and storage, respectively.

Saradopoulos et al. [95] proposed an edge- and camera-based system for insect counting in the context of smart cities. The performance of three different devices including a RPi, an ESP32, and a Google Coral were compared. ESP32 was suggested as the best module for their application, with an inference time of 51 seconds and power consumption of 6mA in deep sleep mode. Moreover, the quantization technique was used to reduce the model size to 0.5MB and fit it into the device memory. This study used regression to count the insects and the authors calculated the accuracy of their system using a custom metric defined as $1 - |MC - AC|/MC$, where MC is the true number and AC is the automatically counted insects. Based on this metric, they achieved an average accuracy of 95%, thus species classification was ignored.

Li et al. [96] developed a few-shot recognition method to identify cotton pests with an accuracy of over 95% accuracy. They benefited from a lightweight CNN model trained with triplet loss and implemented on a Field Programmable Gate Array (FPGA)-based board.

The proposed edge-based systems mostly rely on RPi boards or similar relatively powerful embedded devices, such as Jetson [97][98][99] or smartphones [100][101][102]. Such hardware offers higher computational capabilities but is still expensive and consumes more energy compared to microcontroller units (MCUs). MCUs are widely known for being low-power and low-cost processors, making them more suitable for large-scale IoT deployments where hundreds or thousands of nodes may be required. In remote agricultural applications such as insect monitoring, energy efficiency and cost-effectiveness are critical, as devices often need to operate for long periods on battery and must be affordable for widespread deployment. Therefore, although RPis can handle more complex AI models and higher-resolution image processing, their higher energy demands and costs make MCUs an attractive alternative when the application constraints are considered.

In this regard, Sütő [103] proposed an embedded insect monitoring system using an MCU-based board called OpenMV Cam H7 and the target insect was the Codling Moth. The algorithm used a selective search method for object proposals, and a pre-trained MobileNetV2 was used as a base model for the classification part. The author also proposed a system based on an RPi Zero W in [104]. In this study, the author used the same algorithm for insect identification and counting as with the

previous work and the system required approximately 400 seconds for one cycle operation with an average power consumption of over 2.1W and 22mW in the active and inactive mode, respectively.

In another study by Brunelli et al. [105], an ultra-low power smart pest detection system was proposed based on GAP8 SOC [106] to detect Codling moth. Their device worked only with gray-scale images with a resolution of 244x324 px and extracted some potential areas of the captured image (40×40 px) and utilized a quantized CNN-based model for classification. This system needed 3.5mJ of energy to perform the image processing and the classification and achieved an accuracy of 93%.

Bompani et al. [107] developed ultra-low power insect pest monitoring systems based on heterogeneous multicore microcontrollers. They trained MobileNetV3-SSDLite for their detection task using two different datasets and compared its performance to a lightweight Viola–Jones detector algorithm (i.e., implemented in their recent work [108]) on GAP9 and GAP8 processing units. The DL model achieved a detection accuracy between 83% and 72% with a model size of 3.44 MB and over 1.3 MB of external RAM usage. In this study, the authors used quantization and greyscale imaging with the size of 320×240 px to compress the model.

Crupi et al. [109] employed ultra-low power pocket-sized drones for pest insect monitoring using DL. The authors fine-tuned and then quantized (i.e., int8) FOMO-MobileNetV2 and SSDLite-MobileNetV3, and implemented them on the Arduino Portenta and GAP9 platforms, respectively. The models achieved a mean average precision (mAP) of 0.66 and 0.79, respectively. To decrease the memory footprint of the FOMO to lower than 1 MB, they reduced the input image size to 160×160 px and 96×96 px. For SSDLite, the memory footprint was around 1.8 MB with an input size of 320×320 px. The authors also investigated the greyscale (instead of RGB) inputs to further reduction of the memory footprint, which led to a reduction in the detection accuracy. This optimized SSDLite-MobileNetV3 model on the GAP9 SoC was later used by the authors in [110] and deployed on a nano-UAV to detect insects as a part of a ground-aerial system for pest control in orchards.

These studies are summarized in Table 2.1.

Table 2.1. Summary of reviewed related works on insect monitoring.

Paper	Method	Model	Hardware	Performance	Description
[62]	Cloud detection & counting	YOLO-SIP	camera device and remote server	mAP = 84.22%, MAE = 0.422, MSE = 1.126	Sticky traps images transmitted wirelessly to cloud for processing; requires connectivity; similar image acquisition and offline processing in [64][65].
[66]	Segmentation	DE-Net	Server	~95.5% Jaccard index; 83 MB model; ~0.07 s inference	Captures high-level features; generates high-resolution masks; no power, RAM, or deployment metrics reported.
[67]	Segmentation + detection	U-Net + YOLOv5s / YOLOv8s	Server	AP = 95% (YOLOv8s), 94% (YOLOv5s)	Two stages: U-Net pre-processing and YOLO detection; pheromone trap images; no resource or deployment metrics reported.
[69]	Detection + counting	Mask R-CNN (ResNet50)	Server	R^2 = 0.81 (aphids), R^2 = 0.78 and F1 = 0.55 (parasitoids)	In the lab from trap images; accurate but computationally expensive backbone.
[70]	Cloud-based IoT detection	Faster R-CNN (ResNet50)	4-layer IoT and remote server	94% accuracy; 0.2 s inference	A ball camera captured images; images sent using a WiFi network to a remote server; no power data.
[71-76]	Detection / recognition	Computer vision and ML/DL models	Cloud/Server	Accuracy as the main metric	Focus on recognition performance; ignore other aspects.
[77]	Detection (HH dataset)	YOLOv5 variants	Remote system	>75% mAP	Drone + smartphone images; preliminary filtering (blur & brightness); offline processing.
[78]	Drone platform (HH dataset)	YOLOv5, YOLOv8, RetinaNet, Faster-R-CNN	Android app and server	<71% mAP	Aerial telephotos; server-side inference; requires image transfer.

[88] [89]	Edge detection & classification	LeNet, VGG16, MobileNetV2	RPi3, RPi4, Intel Movidius NCS	>95% accuracy; ~123.2 J energy (RPi3 + LeNet)	RoI extraction + classification pipeline; energy usage compared across hardware.
[90]	Detection	YOLOv5, YOLOv3	RPi 3B	AP=92%, Recall=94%, F1=93%, mAP=94.4%	Model evaluated offline but claimed feasible on RPi; accuracy was the main metric.
[91]	Counting + classification	YOLO + SVM	RPi 2B	92.5% counting, 90% classification; ~5 min/image	Approximate YOLO counting + SVM refinement; slow inference.
[92]	IoT-based detection	VGG19, VGG16, ResNet50, InceptionV3, Xception	RPi	98.91% (VGG19)	Several models evaluated; blockchain for secure transfer; large backbones and high compute.
[93]	Edge detection	SSD-MobilenetV1/V2, YOLOv4-tiny	RPi 3B + Arduino	>95% accuracy	Arduino controls system power; SSD-MobilenetV2 chosen for best trade-off between latency & accuracy.
[94]	Edge detection	TinyYOLOv3 + CNNs	RPi Zero W	F1 = 96%; ~11 sec inference; ~151 MB GPU RAM; ~57 MB storage	Two-stage approach: detection by a tiny YOLO followed by tree-based classification; slow inference.
[95]	Edge counting	CNN (regression)	ESP32/ RPi/ Google Coral	~95% custom accuracy; ~51 sec inference; 6 mA deep sleep; model ~0.5 MB (ESP32)	Quantized; species classification ignored; best deployment on ESP32.

[96]	Few-shot recognition	Lightweight CNN + triplet loss	FPGA board	>95% accuracy	Fow-shot learning mechanism for small datasets; no power/cost reported.
[103]	MCU monitoring	Selective search + MobileNetV2	OpenMV Cam H7 (MCU)	0.164 counting custom loss	A low-power lightweight MCU-based device; no power, RAM, or deployment metrics reported.
[104]	Edge monitoring	Same as [103]	RPi Zero W	~400 sec/cycle; 2.1 W active, 22 mW inactive	Slow due to selective search algorithm; longer cycle time increasing power.
[105]	Edge classification	CNN	GAP8 SoC	93% accuracy; ~3.5 mJ per inference	Grayscale 244×324 input images; 40×40 ROIs; quantized; highly optimized classification pipeline.
[107]	Edge detection	MobileNetV3-SSDLite & Viola-Jones	GAP8 / GAP9	83%-72% accuracy; model size 3.44 MB; ≥1.3 MB external RAM; <0.22 sec inference	Quantization was used to reduce the memory; grayscale 320×240 input images.
[109]	Edge detection	FOMO-MobileNetV2 & SSDLite-MobileNetV3	Nano-UAV + Arduino Portenta H7 / GAP9	mAP = 0.66 (FOMO), 0.79 (SSDLite);	Quantization was used; input reduced to 160×160 or 96×96; grayscale reduces accuracy; footprint: 0.4-3.6 MB, inference: <0.462 sec, power: <0.5 W depending on input/data type.
[110]	Edge detection	SSDLite-MobileNetV3	Nano-UAV + GAP9	mAP = 0.8; memory: 1.8 / 3.6 MB, inference: <0.462 sec; power: <0.04 W.	Part of ground-aerial system for orchard pest control; quantization was used; 320×240 RGB input images.

In all the mentioned edge-based systems, the hardware has a direct impact on the selection of the AI model, as the model must align with the computational and resource limitations of the device. To address this, researchers not only choose lightweight models but also attempt to compress or optimize them further using techniques such as quantization, working with greyscale images, reducing input resolution, and more. This highlights the importance of hardware-software co-design, where both the AI model and the hardware platform are considered together during system development to ensure efficient and reliable performance in edge environments. In the works mentioned, a variety of different types of hardware were used for insect monitoring applications. Table 2.2 compares the most common hardware used in such applications. Based on the table, it is clear that microcontroller units (MCUs) have a much lower cost and power consumption than other solutions, thus, they can be considered the best option for insect monitoring applications deployed in remote areas.

Therefore, MCUs are a good fit for battery-powered IoT-based edge devices due to their low-cost and low-power consumption characteristics, which are critical factors for such devices, especially in this application where long-lasting performance in remote locations is essential. However, typical MCUs only have up to 1MB of RAM and 2MB of storage, which makes it challenging to implement and run DL models on them. To address these limitations, lightweight DL models (e.g., MobileNet family) are generally selected as a starting point due to their reduced computational and memory requirements. However, to fully meet the strict constraints of MCU-based platforms, additional optimization techniques, such as model compression, quantization, and architectural modifications, are required to further reduce model size

Table 2.2 Examples of hardware used in smart insect monitoring systems in the literature.

Platform	Example	Memory Capacity (MB)	Storage Capacity (GB)	Max Power Consumption[a] (W)	Price[b] ($)
Cloud/Server	GPU NVIDIA V100	≥16384	≥1024	250	≥5K
FPGA	Artix-7	≥1024	≥8	≥1	≥500
Mobile phone	iPhone 11	≥4096	≥64	8	≥500
Raspberry Pi	RPi 4B	≥1024	≥8	5	≥60
Microcontroller (MCUs)	STM32H7	≤1	≤0.002	0.3	30

[a] Power consumption is approximate as it depends on several factors.
[b] Prices are approximate due to the global chip shortage issue at the time of this report.

and computational complexity. It is worth noting that not all optimization techniques are equally suitable for MCU-based deployment. For example, standard pruning methods may not always lead to practical benefits on MCUs when using existing deployment libraries, as they often do not reduce the actual memory usage or inference time. Moreover, relying on a single optimization technique is often insufficient to meet the strict constraints of MCU-based systems. Therefore, effective deployment requires a combination of optimization strategies, carefully selected based on the target hardware capabilities and resource limitations, while also considering their impact on model accuracy.

2.7 Data scarcity and environmental variability

Typically, DL models rely heavily on large volumes of labelled data to achieve exceptional accuracy [111][112]. Lack of proper real-world data is a fundamental issue of deployed edge AI systems, which can lead to inaccurate DL performance and, eventually, to system failures. DL models are generally trained on datasets gathered for specific tasks, meaning they may not perform accurately under different deployment conditions or for various insect species in this application. As a result, new datasets need to be created for each new setup or target insect.

Insects are the most diverse animal group on our planet, with a wide range of morphologies [61], which make dataset creation really challenging. Building a proper dataset is time-consuming, expensive and tedious, especially the labelling or annotation part [113]. This becomes even more difficult with insects, as they appear seasonally and are not available for image acquisition during the whole year. For example, if there is a problem with the acquisition system, data collection is delayed until the next season, which could be the following year. Equally important, manual labelling by humans is labour-intensive, costly and requires expert domain knowledge [114], especially for insects which are small and diverse.

In addition, an issue that often arises in real-world applications is related to the fact that the characteristics of the data collected by the system can change over time, a problem known as data drift. This can occur because of seasonality, environmental changes, or even IoT components and sensor ageing [113][116]. These changes negatively affect the DL models and decrease the performance of real-world IoT-based

systems, particularly insect monitoring systems, since orchards are dynamic environments.

There are several techniques to address these issues, some of the most relevant ones are described in the following paragraphs.

Data augmentation is one of the first and most widely used approaches to increase the number of samples in a dataset to reduce overfitting. Classical image augmentation applies predefined transformations to original images to create new samples and improve dataset diversity, which results in improved robustness and generalization of DL models. The common transformations include rotation, flipping, cropping, zooming, adding noise, colour shifting, and others [117]. In some cases, it can also simulate real-world conditions, for example, by introducing colour or lighting shifts or blurriness. Augmentation can help simulate changes in image capturing conditions or environmental variations. This is widely used in studies such as [92][70][76][77]. In addition to classical augmentation, there is DL-based image augmentation, in which synthetic images are generated and added to the dataset using a DL model. Generative Adversarial Networks (GANs) are one of such methods that generate new synthetic images based on available images in the dataset [114] and are recently used in studies such as [118][119][120].

Transfer learning is another widely used technique in the literature. In this method, a pretrained model, which was trained on a large dataset (e.g., ImageNet), is used as a base model and fine-tuned for a specific task with limited available labelled data. In this way, learned knowledge is transferred and shared with the new task, which reduces the amount of labelled data required as well as the training time and computational resources [112][121]. Many studies use this method, including [67][69][77][78].

These two methods are widely used in studies but are not sufficient on their own. They serve as foundational approaches, generally combined with other methods and techniques to improve performance.

Few-shot learning (FSL) attempts to learn from a few labelled samples to perform a given task. Therefore, it is a good option in scenarios affected by a small dataset. This idea is inspired by a human's rapid learning process, where prior knowledge (i.e., obtained from a large amount of data) is used to adapt to a new task

with limited available labelled data [122]. FSL can be implemented in many ways, such as metric-based methods where the model learns an embedding space where samples from the same class are close together and those from different classes are far apart [123]. This approach has been applied in several agricultural applications, such as [96][124][125].

Humans possess an innate ability to learn incrementally [126]. Inspired by this, Incremental Learning (IL), also known as continual learning (CL) or lifelong learning, aims to provide similar abilities by re-training a model on batches of newly captured data as they become available. IL allows the model to learn and improve its performance over time, thus it is a good option for applications where data is collected gradually over time. It also offers a practical solution for dealing with dynamic environmental issues, where the characteristics of the data may change over time. By continuously fine-tuning the model over time using new data, the model adapts to potential shifts in data distribution [127][128].

In most real-world scenarios, it is possible to encounter situations where, in addition to a small labelled dataset, a large amount of unlabelled data is available. In semi-supervised learning (SSL), the idea is to make use of this unlabelled data alongside the labelled data to enhance learning performance. Therefore, SSL lies somewhere between supervised learning, which relies entirely on labelled data, and unsupervised learning, which does not require any labelled data. SSL can be implemented in many ways, such as pseudo-labeling methods, where the unlabelled data are labelled using the DL model and then used as additional training data in the next steps [129][113]. Many studies use this method, including [130][131][132].

All these methods are widely used across different applications and scenarios, and in some cases, they are combined to achieve better performance. For example, data augmentation and transfer learning are often applied together with approaches such as model compression techniques, CL, or SSL to address issues such as resource-constrained hardware, limited labelled datasets, overfitting and domain variability. This combination not only improves model accuracy but also enables adaptation to new environments, making them more efficient for deployment on resource-constrained devices.

In insect monitoring, as well as many other applications, there is only a small amount of labelled data available at the beginning of the training process and model development. After deployment, large volumes of unlabelled data are generated over time. In addition, the system will experience data drift over time. Therefore, a proper combination of the above-mentioned methods, given all the limitations and considerations, would provide a practical way to tackle these limitations in this case study.

2.8 Research gaps, objectives and hypothesis

Current literature shows a limited number of studies focusing on practical edge-based insect monitoring systems, where computing is carried out near the data source (e.g., fruit orchards), while considering the severe constraints of agricultural environments. Among these, very few works perform the computing function at the edge using low-cost, low-power, and resource-constrained MCU-based architectures. The limited literature that addressed MCU-based solutions primarily aimed to achieve high accuracy on these devices but often overlooked critical real-world issues including the lack of proper annotated datasets for training DL models and data drift due to the highly dynamic nature of deployment environments in agricultural fields. These gaps highlight the need for research into a practical, robust, and energy-efficient edge AI-based framework that can operate reliably under the real-world conditions of remote orchards for insect detection and identification.

Based on the challenges and the gaps, the main objectives of this thesis are defined as follows:

- To develop a low-power edge-based prototype device for insect detection and identification that can capture and analyze insect trap images and transmit required data using a resource-constrained microcontroller unit (MCU), enabling reliable operation in remote agricultural environments with limited power and connectivity.
- To develop and validate a novel TinyDL model designed for deployment on a resource-constrained MCU for automated insect monitoring, capable of performing insect segmentation and counting under strict memory and computational constraints.

- To design and validate a practical and comprehensive edge-based insect monitoring framework that operates reliably under real-world agricultural constraints to address fundamental deep-learning (DL) challenges, including the lack of annotated datasets and post-deployment data drift.

Based on the identified research gaps and the objectives defined above, the following research questions are formulated:

RQ1: Is it feasible to develop a low-power edge-based prototype device for insect detection and identification capable of performing image acquisition and lightweight inference directly on resource-constrained microcontroller units (MCUs) while operating reliably in remote orchards without cloud or internet connectivity?

RQ2: Can a novel TinyDL model designed for deployment on a resource-constrained microcontroller unit (MCU) for automated insect monitoring achieve acceptable performance for insect segmentation and counting while satisfying the strict memory and computational limitations of MCU-based hardware?

RQ3: Can a practical and comprehensive edge-based insect monitoring framework that operates reliably under real-world agricultural constraints address the challenges of limited annotated datasets and post-deployment data drift through pseudo-labeling and incremental learning without requiring cloud-based retraining?

These research questions guide the development and evaluation of the proposed edge-based insect monitoring system presented in the following technical chapters of this thesis.

Therefore, based on these objectives and research questions, the hypotheses below are formulated to evaluate the expected outcomes:

- It is hypothesised that a functional low-power insect monitoring device based on MCU hardware can be developed that operates within strict energy and memory constraints, while reliably performing image acquisition and lightweight inference locally in remote orchards without cloud or internet connectivity.
- It is hypothesised that a TinyDL model specifically optimized for insect identification and counting can achieve efficient execution on resource-constrained MCUs (in terms of model size, memory usage, inference time, and

energy consumption) while maintaining acceptable performance in terms of accuracy and robustness for insect segmentation and counting tasks.

- It is hypothesised that a self-learning edge-based framework based on pseudo-labeling and incremental learning can effectively mitigate the effects of limited training data and post-deployment data drift, without requiring manual annotation or cloud-based retraining while maintaining stable performance in terms of accuracy and robustness.

The objectives of this thesis are addressed in three technical chapters, organized as follows. Chapter 3 investigates the feasibility of a low-power edge-based insect monitoring system through the development and deployment of an MCU-based prototype device (Objective 1). Chapter 4 focuses on the development of a lightweight deep learning model designed for insect segmentation and counting while operating under the memory and computational constraints of MCUs (Objective 2). Chapter 5 extends this work by proposing a self-improving edge-based monitoring framework to address data scarcity and post-deployment data drift using pseudo-labeling techniques and incremental learning (Objective 3).

2.9 Conclusion

This chapter highlighted the key motivations and technical aspects behind the research presented in this thesis. It also reviewed the limitations in current studies in this research area and explained why these solutions are not practical in remote agricultural environments. Although some progress has been made in using edge-based systems and TinyDL models, the literature still lacks a reliable approach that combines real-world deployment, efficient inference on MCUs and self-adaptive learning in the field. These gaps provide the context for the main contributions of this work, which are presented in the following chapters. The next chapter focuses on the design and evaluation of a low-power edge-based monitoring device as the first step to address these challenges.

2.10 References

[1] A. Konya and P. Nematzadeh, “Recent applications of AI to environmental disciplines: A review,” Science of The Total Environment, vol. 906, p. 167705, Jan. 2024, doi: 10.1016/J.SCITOTENV.2023.167705.

[2] R. Singh and S. S. Gill, “Edge AI: A survey,” Internet of Things and Cyber-Physical Systems, vol. 3, pp. 71–92, Jan. 2023, doi: 10.1016/J.IOTCPS.2023.02.004.

[3] J. Ren, D. Zhang, S. He, Y. Zhang, and T. Li, “A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet,” ACM Comput Surv, vol. 52, no. 6, p. 111, Nov. 2020, doi: 10.1145/3362031.

[4] R. Kumar and N. Agrawal, “Analysis of multi-dimensional Industrial IoT (IIoT) data in Edge–Fog–Cloud based architectural frameworks : A survey on current state and research challenges,” J Ind Inf Integr, vol. 35, p. 100504, Oct. 2023, doi: 10.1016/J.JII.2023.100504.

[5] P. Kumari and P. Kaur, “A survey of fault tolerance in cloud computing,” Journal of King Saud University - Computer and Information Sciences, vol. 33, no. 10, pp. 1159–1176, Dec. 2021, doi: 10.1016/J.JKSUCI.2018.09.021.

[6] T. Meuser et al., “Revisiting Edge AI: Opportunities and Challenges,” IEEE Internet Comput, vol. 28, no. 4, pp. 49–59, 2024, doi: 10.1109/MIC.2024.3383758.

[7] V. Hayyolalam, M. Aloqaily, O. Ozkasap, and M. Guizani, “Edge Intelligence for Empowering IoT-Based Healthcare Systems,” IEEE Wirel Commun, vol. 28, no. 3, pp. 6–14, Jun. 2021, doi: 10.1109/MWC.001.2000345.

[8] M. Hartmann, U. S. Hashmi, and A. Imran, “Edge computing in smart health care systems: Review, challenges, and research directions,” Transactions on Emerging Telecommunications Technologies, vol. 33, no. 3, Mar. 2022, doi: 10.1002/ETT.3710.

[9] V. Hayyolalam, M. Aloqaily, O. Ozkasap, and M. Guizani, “Edge-Assisted Solutions for IoT-Based Connected Healthcare Systems: A Literature Review,” IEEE Internet Things J, vol. 9, no. 12, pp. 9419–9443, Jun. 2022, doi: 10.1109/JIOT.2021.3135200.

[10] L. U. Khan, I. Yaqoob, N. H. Tran, S. M. A. Kazmi, T. N. Dang, and C. S. Hong, “Edge-Computing-Enabled Smart Cities: A Comprehensive Survey,” IEEE Internet Things J, vol. 7, no. 10, pp. 10200–10232, Oct. 2020, doi: 10.1109/JIOT.2020.2987070.

[11] C. Xia, W. Li, X. Chang, F. C. Delicato, T. Yang, and A. Zomaya, “Edge-based energy management for smart homes,” in 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing ... (DASC/ PiCom/ DataCom/ CyberSciTech), Institute of Electrical and Electronics Engineers Inc., Oct. 2018, pp. 842–848. doi: 10.1109/DASC/PICOM/DATACOM/CYBERSCITEC.2018.00-19.

[12] S. Zhang, W. Li, Y. Wu, P. Watson, and A. Y. Zomaya, “Enabling edge intelligence for activity recognition in smart homes,” Proceedings - 15th IEEE International Conference on Mobile Ad Hoc and Sensor Systems, MASS 2018, pp. 228–236, Dec. 2018, doi: 10.1109/MASS.2018.00044.

[13] M. Sharma, A. Tomar, and A. Hazra, “Edge Computing for Industry 5.0: Fundamental, Applications, and Research Challenges,” IEEE Internet Things J, vol. 11, no. 11, pp. 19070–19093, Jun. 2024, doi: 10.1109/JIOT.2024.3359297.

[14] F. Liang, W. Yu, X. Liu, D. Griffith, and N. Golmie, “Toward Edge-Based Deep Learning in Industrial Internet of Things,” IEEE Internet Things J, vol. 7, no. 5, pp. 4329–4341, May 2020, doi: 10.1109/JIOT.2019.2963635.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” Adv Neural Inf Process Syst, vol. 25, 2012.

[16] O. Russakovsky et al., “ImageNet Large Scale Visual Recognition Challenge,” Int J Comput Vis, vol. 115, no. 3, pp. 211–252, Dec. 2015, doi: 10.1007/S11263-015-0816-Y/FIGURES/16.

[17] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings, Sep. 2014, Accessed: Jul. 25, 2025. [Online]. Available: https://arxiv.org/pdf/1409.1556

[18] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2016-December, pp. 770–778, Dec. 2016, doi: 10.1109/CVPR.2016.90.

[19] C. Szegedy et al., “Going deeper with convolutions,” Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 07-12-June-2015, pp. 1–9, Oct. 2015, doi: 10.1109/CVPR.2015.7298594.

[20] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely Connected Convolutional Networks,” Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, vol. 2017-January, pp. 2261–2269, Aug. 2016, doi: 10.1109/CVPR.2017.243.

[21] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning Transferable Architectures for Scalable Image Recognition,” Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 8697–8710, Jul. 2017, doi: 10.1109/CVPR.2018.00907.

[22] 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 <0.5MB model size,” Feb. 2016, Accessed: Aug. 06, 2025. [Online]. Available: https://arxiv.org/pdf/1602.07360

[23] A. G. Howard et al., “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” Apr. 2017, Accessed: Jul. 25, 2025. [Online]. Available: https://arxiv.org/pdf/1704.04861

[24] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 4510–4520, Jan. 2018, doi: 10.1109/CVPR.2018.00474.

[25] A. Howard et al., “Searching for MobileNetV3,” Proceedings of the IEEE International Conference on Computer Vision, pp. 1314–1324, May 2019, doi: 10.1109/ICCV.2019.00140.

[26] M. Tan and Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” 36th International Conference on Machine Learning, ICML 2019, vol. 2019-June, pp. 10691–10700, May 2019, Accessed: Jul. 25, 2025. [Online]. Available: https://arxiv.org/pdf/1905.11946

[27] M. Tan and Q. V. Le, “EfficientNetV2: Smaller Models and Faster Training,” Proc Mach Learn Res, vol. 139, pp. 10096–10106, Apr. 2021, Accessed: Jul. 25, 2025. [Online]. Available: https://arxiv.org/pdf/2104.00298

[28] S. S. Saha, S. S. Sandha, and M. Srivastava, “Machine Learning for Microcontroller-Class Hardware: A Review,” IEEE Sens J, vol. 22, no. 22, pp. 21362–21390, Nov. 2022, doi: 10.1109/JSEN.2022.3210773.

[29] C. Banbury et al., “MicroNets: Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers,” Proceedings of Machine Learning and Systems, vol. 3, pp. 517–532, Mar. 2021, Accessed: Aug. 15, 2025. [Online]. Available: https://github.com/ARM-software/ML-zoo.

[30] J. Lin, W. M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, “MCUNet: Tiny Deep Learning on IoT Devices,” Adv Neural Inf Process Syst, vol. 2020-December, Jul. 2020, Accessed: Jul. 25, 2025. [Online]. Available: https://arxiv.org/pdf/2007.10319

[31] J. Gou, B. Yu, S. J. Maybank, and D. Tao, “Knowledge Distillation: A Survey,” Int J Comput Vis, vol. 129, no. 6, pp. 1789–1819, Jun. 2021, doi: 10.1007/S11263-021-01453-Z/TABLES/6.

[32] H. Cheng, M. Zhang, and J. Q. Shi, “A Survey on Deep Neural Network Pruning: Taxonomy, Comparison, Analysis, and Recommendations,” IEEE Trans Pattern Anal Mach Intell, vol. 46, no. 12, pp. 10558–10578, 2024, doi: 10.1109/TPAMI.2024.3447085.

[33] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, “A Survey of Quantization Methods for Efficient Neural Network Inference,” Low-Power Computer Vision, pp. 291–326, Jan. 2022.

[34] J. Lin, W. M. Chen, H. Cai, C. Gan, and S. Han, “MCUNetV2: Memory-Efficient Patch-based Inference for Tiny Deep Learning,” Adv Neural Inf Process Syst, vol. 4, pp. 2346–2358, Oct. 2021, Accessed: Jul. 25, 2025. [Online]. Available: https://arxiv.org/pdf/2110.15352

[35] M. T. Lê, P. Wolinski, and J. Arbel, “Efficient Neural Networks for Tiny Machine Learning: A Comprehensive Review,” Nov. 2023, Accessed: Jul. 25, 2025. [Online]. Available: https://arxiv.org/pdf/2311.11883

[36] P. Sharma et al., “A lightweight deep learning model for automatic segmentation and analysis of ophthalmic images,” Sci Rep, vol. 12, no. 1, pp. 1–18, Dec. 2022, doi: https://doi.org/10.1038/s41598-022-12486-w.

[37] P. S. Thakur, T. Sheorey, and A. Ojha, “VGG-ICNN: A Lightweight CNN model for crop disease identification,” Multimed Tools Appl, vol. 82, no. 1, pp. 497–520, Jan. 2023, doi: 10.1007/S11042-022-13144-Z/TABLES/10.

[38] M. Wieczorek, J. Silka, M. Wozniak, S. Garg, and M. M. Hassan, “Lightweight Convolutional Neural Network Model for Human Face Detection in Risk Situations,” IEEE Trans Industr Inform, vol. 18, no. 7, pp. 4820–4829, Jul. 2022, doi: 10.1109/TII.2021.3129629.

[39] T. Dlamini and S. Vilakati, “Remote and Rural Connectivity: Infrastructure and Resource Sharing Principles,” Wirel Commun Mob Comput, vol. 2021, no. 1, p. 6065119, Jan. 2021, doi: 10.1155/2021/6065119.

[40] E. Yaacoub and M. S. Alouini, “A Key 6G Challenge and Opportunity-Connecting the Base of the Pyramid: A Survey on Rural Connectivity,” Proceedings of the IEEE, vol. 108, no. 4, pp. 533–582, Apr. 2020, doi: 10.1109/JPROC.2020.2976703.

[41] M. Vaezi et al., “Cellular, Wide-Area, and Non-Terrestrial IoT: A Survey on 5G Advances and the Road Toward 6G,” IEEE Communications Surveys and Tutorials, vol. 24, no. 2, pp. 1117–1174, 2022, doi: 10.1109/COMST.2022.3151028.

[42] A. A. Zaidan et al., “A survey on communication components for IoT-based technologies in smart homes,” Telecommunication Systems 2018 69:1, vol. 69, no. 1, pp. 1–25, Mar. 2018, doi: 10.1007/S11235-018-0430-8.

[43] S. Seneviratne et al., “A Survey of Wearable Devices and Challenges,” IEEE Communications Surveys and Tutorials, vol. 19, no. 4, pp. 2573–2620, Oct. 2017, doi: 10.1109/COMST.2017.2731979.

[44] WHO; FAO; OIE; UNEP, “The future of food and agriculture - Trends and challenges,” United Nations Human Settlements Programme (UN-Habitat): Addis Ababa, Ethiopia. (2021), p. 52, 2017, Accessed: Nov. 13, 2025. [Online]. Available: https://openknowledge.fao.org/handle/20.500.14283/i6583e

[45] “About | Plant Production and Protection | Food and Agriculture Organization of the United Nations.” Accessed: Oct. 15, 2025. [Online]. Available: https://www.fao.org/plant-production-protection/about/en

[46] D. R. Paini, A. W. Sheppard, D. C. Cook, P. J. De Barro, S. P. Worner, and M. B. Thomas, “Global threat to agriculture from invasive species,” Proc Natl Acad Sci U S A, vol. 113, no. 27, pp. 7575–7579, Jul. 2016, doi: 10.1073/PNAS.1602205113/SUPPL_FILE/PNAS.1602205113.SAPP.PDF.

[47] A. C. Teixeira, J. Ribeiro, R. Morais, J. J. Sousa, and A. Cunha, “A Systematic Review on Automatic Insect Detection Using Deep Learning,” Agriculture, vol. 13, no. 3, p. 713, Mar. 2023, doi: 10.3390/AGRICULTURE13030713.

[48] J. She et al., “A method for automatic real-time detection and counting of fruit fly pests in orchards by trap bottles via convolutional neural network with attention mechanism added,” Ecol Inform, vol. 70, p. 101690, Sep. 2022, doi: 10.1016/J.ECOINF.2022.101690.

[49] S. Frank, L. Bradley, and K. Moore, “Integrated Pest Management, Chapter 8,” Feb. 2022. [Online]. Available: https://content.ces.ncsu.edu/extension-gardener-handbook/8-integrated-pest-management-ipm

[50] V. Ferrari et al., “Evaluation of the potential of near infrared hyperspectral imaging for monitoring the invasive brown marmorated stink bug,” Chemometrics and Intelligent Laboratory Systems, vol. 234, p. 104751, Mar. 2023, doi: 10.1016/J.CHEMOLAB.2023.104751.

[51] E. C. Tetila, B. B. MacHado, G. V. Menezes, N. A. De Souza Belete, G. Astolfi, and H. Pistori, “A Deep-Learning Approach for Automatic Counting of Soybean Insect Pests,” IEEE Geoscience and Remote Sensing Letters, vol. 17, no. 10, pp. 1837–1841, Oct. 2020, doi: 10.1109/LGRS.2019.2954735.

[52] A. Bereciartua-Pérez, L. Gómez, A. Picón, R. Navarra-Mestre, C. Klukas, and T. Eggers, “Insect counting through deep learning-based density maps estimation,” Comput Electron Agric, vol. 197, p. 106933, Jun. 2022, doi: 10.1016/J.COMPAG.2022.106933.

[53] W. Albattah, M. Masood, A. Javed, M. Nawaz, and S. Albahli, “Custom CornerNet: a drone-based improved deep learning technique for large-scale multiclass pest localization and classification,” Complex and Intelligent Systems, vol. 9, no. 2, pp. 1299–1316, Apr. 2023, doi: 10.1007/S40747-022-00847-X/TABLES/8.

[54] H. Lee, W. Choi, S. E. Eom, and J. J. Park, “Rapid estimation of the density of whiteflies (Hemiptera: Aleyrodidae) on sticky traps in paprika greenhouses using the presence–absence model,” J Asia Pac Biodivers, vol. 15, no. 2, pp. 225–230, Jun. 2022, doi: 10.1016/J.JAPB.2022.03.001.

[55] J. Chen, Y. Fan, T. Wang, C. Zhang, Z. Qiu, and Y. He, “Automatic Segmentation and Counting of Aphid Nymphs on Leaves Using Convolutional Neural Networks,” Agronomy, vol. 8, no. 8, p. 129, Jul. 2018, doi: 10.3390/AGRONOMY8080129.

[56] T. Saranya, C. Deisy, S. Sridevi, and K. S. M. Anbananthen, “A comparative study of deep learning and Internet of Things for precision agriculture,” Eng Appl Artif Intell, vol. 122, p. 106034, Jun. 2023, doi: 10.1016/J.ENGAPPAI.2023.106034.

[57] D. Popescu, A. Dinca, L. Ichim, and N. Angelescu, “New trends in detection of harmful insects and pests in modern agriculture using artificial neural networks. a review,” Front Plant Sci, vol. 14, p. 1268167, Nov. 2023, doi: 10.3389/FPLS.2023.1268167/XML.

[58] A. Passias et al., “Insect Pest Trap Development and DL-Based Pest Detection: A Comprehensive Review,” IEEE Transactions on AgriFood Electronics, vol. 2, no. 2, pp. 323–334, Aug. 2024, doi: 10.1109/TAFE.2024.3436470.

[59] J. Bremmer, A. Gonzalez-Martinez, R. Jongeneel, H. Huiting, R. Stokkers, and M. Ruijs, Impact assessment of EC 2030 Green Deal Targets for sustainable crop production, no. 2021–150. Wageningen Economic Research, 2021.

[60] M. Preti, F. Verheggen, and S. Angeli, “Insect pest monitoring with camera-equipped traps: strengths and limitations,” J Pest Sci (2004), vol. 94, no. 2, pp. 203–217, Mar. 2021, doi: 10.1007/S10340-020-01309-4/TABLES/2.

[61] W. Li, T. Zheng, Z. Yang, M. Li, C. Sun, and X. Yang, “Classification and detection of insects from field images using deep learning for smart pest management: A systematic review,” Ecol Inform, vol. 66, p. 101460, Dec. 2021, doi: 10.1016/J.ECOINF.2021.101460.

[62] Q. Guo, C. Wang, D. Xiao, and Q. Huang, “Automatic monitoring of flying vegetable insect pests using an RGB camera and YOLO-SIP detector,” Precis Agric, vol. 24, no. 2, pp. 436–457, Apr. 2023, doi: 10.1007/S11119-022-09952-W/FIGURES/8.

[63] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” Apr. 2020, Accessed: Jul. 28, 2025. [Online]. Available: https://arxiv.org/pdf/2004.10934

[64] W. Li, Z. Yang, J. Lv, T. Zheng, M. Li, and C. Sun, “Detection of Small-Sized Insects in Sticky Trapping Images Using Spectral Residual Model and Machine Learning,” Front Plant Sci, vol. 13, p. 915543, Jun. 2022, doi: 10.3389/FPLS.2022.915543/BIBTEX.

[65] W. Li, D. Wang, M. Li, Y. Gao, J. Wu, and X. Yang, “Field detection of tiny pests from sticky trap images using deep learning in agricultural greenhouse,” Comput Electron Agric, vol. 183, p. 106048, Apr. 2021, doi: 10.1016/J.COMPAG.2021.106048.

[66] H. Tang, B. Wang, and X. Chen, “Deep learning techniques for automatic butterfly segmentation in ecological images,” Comput Electron Agric, vol. 178, p. 105739, Nov. 2020, doi: 10.1016/J.COMPAG.2020.105739.

[67] A. C. Teixeira, G. A. Carneiro, R. Morais, J. J. Sousa, and A. Cunha, “Segmentation as a Pre-processing for Automatic Grape Moths Detection,” Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes

in Bioinformatics), vol. 14116 LNAI, pp. 388–398, 2023, doi: 10.1007/978-3-031-49011-8_31/FIGURES/5.

[68] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, "A Review of Yolo Algorithm Developments," Procedia Comput Sci, vol. 199, pp. 1066–1073, Jan. 2022, doi: 10.1016/J.PROCS.2022.01.135.

[69] T. De Cesaro Júnior, R. Rieder, J. R. Di Domênico, and D. Lau, "InsectCV: A system for insect detection in the lab from trap images," Ecol Inform, vol. 67, p. 101516, Mar. 2022, doi: 10.1016/J.ECOINF.2021.101516.

[70] B. Ramalingam et al., "Remote Insects Trap Monitoring System Using Deep Learning Framework and IoT," Sensors 2020, Vol. 20, Page 5280, vol. 20, no. 18, p. 5280, Sep. 2020, doi: 10.3390/S20185280.

[71] D. J. A. Rustia, J. J. Chao, J. Y. Chung, and T. Te Lin, "An Online Unsupervised Deep Learning Approach for an Automated Pest Insect Monitoring System," 2019 ASABE Annual International Meeting, pp. 1-, 2019, doi: 10.13031/AIM.201900477.

[72] N. Tuan Nam and P. Duy Hung, "Pest detection on Traps using Deep Convolutional Neural Networks," Proceedings of the 2018 International Conference on Control and Computer Vision - ICCCV '18, 2018, doi: 10.1145/3232651.

[73] D. J. A. Rustia and T. Te Lin, "An IoT-based Wireless Imaging and Sensor Node System for Remote Greenhouse Pest Monitoring," Chem Eng Trans, vol. 58, pp. 601–606, Jun. 2017, doi: 10.3303/CET1758101.

[74] L. Butera, A. Ferrante, M. Jermini, M. Prevostini, and C. Alippi, "Precise Agriculture: Effective Deep Learning Strategies to Detect Pest Insects," IEEE/CAA Journal of Automatica Sinica, vol. 9, no. 2, pp. 246–258, Feb. 2022, doi: 10.1109/JAS.2021.1004317.

[75] E. A. Al-Shahari, G. Aldehim, N. Sharaf Almalki, M. Assiri, A. Sayed, and M. M. Alnfiai, "Innovative Insect Detection and Classification for the Agricultural Sector Using Gannet Optimization Algorithm with Deep Learning," IEEE Access, vol. 12, pp. 108041–108051, 2024, doi: 10.1109/ACCESS.2024.3438308.

[76] D. Kapetas, P. Christakakis, V. Goglia, and E. M. Pechlivani, "AI-based robotic trap for real-time insect detection, monitoring and population prediction," 6th IEEE International Conference on Image Processing, Applications and Systems, IPAS 2025 - Proceedings, 2025, doi: 10.1109/IPAS63548.2025.10924486.

[77] F. Betti Sorbelli, L. Palazzetti, and C. M. Pinotti, "YOLO-based detection of Halyomorpha halys in orchards using RGB cameras and drones," Comput Electron Agric, vol. 213, p. 108228, Oct. 2023, doi: 10.1016/J.COMPAG.2023.108228.

[78] L. Palazzetti et al., "The hawk eye scan: Halyomorpha halys detection relying on aerial tele photos and neural networks," Comput Electron Agric, vol. 226, p. 109365, Nov. 2024, doi: 10.1016/J.COMPAG.2024.109365.

[79] D. Giannetti, N. Patelli, L. Palazzetti, F. Betti Sorbelli, C. M. Pinotti, and L. Maistrello, "First use of unmanned aerial vehicles to monitor Halyomorpha halys and recognize it using artificial intelligence," Pest Manag Sci, vol. 80, no. 8, pp. 4074–4084, Aug. 2024, doi: 10.1002/PS.8115.

[80] A. Sava, L. Ichim, and D. Popescu, "Detection of Halyomorpha Halys Using Neural Networks," 2022 8th International Conference on Control, Decision and Information Technologies, CoDIT 2022, pp. 437–442, 2022, doi: 10.1109/CODIT55151.2022.9803899.

[81] A. Dinca, D. Popescu, C. M. Pinotti, L. Ichim, L. Palazzetti, and N. Angelescu, “Halyomorpha Halys Detection in Orchard from UAV Images Using Convolutional Neural Networks,” Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 14135 LNCS, pp. 315–326, 2023, doi: 10.1007/978-3-031-43078-7_26/FIGURES/14.

[82] A. Dinca, N. Angelescu, L. Ichim, and D. Popescu, “Halyomorpha Halys Detection Using Efficient Neural Networks,” Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 13625 LNCS, pp. 459–470, 2023, doi: 10.1007/978-3-031-30111-7_39/TABLES/1.

[83] A. Dinca, N. Angelescu, L. Ichim, and D. Popescu, “Deep Neural Networks for Halyomorpha Halys Detection,” 2023 31st Mediterranean Conference on Control and Automation, MED 2023, pp. 317–322, 2023, doi: 10.1109/MED59994.2023.10185751.

[84] L. Ichim, R. Ciciu, and D. Popescu, “Using Drones and Deep Neural Networks to Detect Halyomorpha Halys in Ecological Orchards,” International Geoscience and Remote Sensing Symposium (IGARSS), vol. 2022-July, pp. 437–440, 2022, doi: 10.1109/IGARSS46834.2022.9883742.

[85] F. B. Sorbelli, L. Palazzetti, and C. M. Pinotti, “A Drone-based Automated Halyomorpha halys Scouting: A Case Study on Orchard Monitoring,” 2023 IEEE International Workshop on Metrology for Agriculture and Forestry, MetroAgriFor 2023 - Proceedings, pp. 380–385, 2023, doi: 10.1109/METROAGRIFOR58484.2023.10424287.

[86] F. B. Sorbelli, L. Palazzetti, and C. M. Pinotti, “Preliminary Results for Halyomorpha halys Monitoring Relying on a Custom Dataset,” 2023 IEEE International Workshop on Metrology for Agriculture and Forestry, MetroAgriFor 2023 - Proceedings, pp. 363–368, 2023, doi: 10.1109/METROAGRIFOR58484.2023.10424403.

[87] Y. Kalyani and R. Collier, “A Systematic Survey on the Role of Cloud, Fog, and Edge Computing Combination in Smart Agriculture,” Sensors 2021, Vol. 21, Page 5922, vol. 21, no. 17, p. 5922, Sep. 2021, doi: 10.3390/S21175922.

[88] D. Brunelli, A. Albanese, D. d’Acunto, and M. Nardello, “Energy Neutral Machine Learning Based IoT Device for Pest Detection in Precision Agriculture,” IEEE Internet of Things Magazine, vol. 2, no. 4, pp. 10–13, Feb. 2020, doi: 10.1109/IOTM.0001.1900037.

[89] A. Albanese, M. Nardello, and D. Brunelli, “Automated Pest Detection with DNN on the Edge for Precision Agriculture,” IEEE J Emerg Sel Top Circuits Syst, vol. 11, no. 3, pp. 458–467, Sep. 2021, doi: 10.1109/JETCAS.2021.3101740.

[90] K. Bjerge, J. Alison, M. Dyrmann, C. E. Frigaard, H. M. R. Mann, and T. T. Høye, “Accurate detection and identification of insects from camera trap images with deep learning,” PLOS Sustainability and Transformation, vol. 2, no. 3, p. e0000051, Mar. 2023, doi: 10.1371/JOURNAL.PSTR.0000051.

[91] Y. Zhong, J. Gao, Q. Lei, and Y. Zhou, “A Vision-Based Counting and Recognition System for Flying Insects in Intelligent Agriculture,” Sensors, vol. 18, no. 5, p. 1489, May 2018, doi: 10.3390/S18051489.

[92] W. Rahman et al., “Automated Detection of Harmful Insects in Agriculture: A Smart Framework Leveraging IoT, Machine Learning, and Blockchain,” IEEE Transactions on Artificial Intelligence, vol. 5, no. 9, pp. 4787–4798, 2024, doi: 10.1109/TAI.2024.3394799.

[93] D. A. Pham, A. D. Le, D. T. Pham, and H. B. Vo, “AlertTrap: On Designing An Edge-Computing Remote Insect Monitoring System,” Proceedings - 2021 8th NAFOSTED Conference on Information and Computer Science, NICS 2021, pp. 323–328, 2021, doi: 10.1109/NICS54270.2021.9701558.

[94] D. J. A. Rustia et al., “Edge-based wireless imaging system for continuous monitoring of insect pests in a remote outdoor mango orchard,” Comput Electron Agric, vol. 211, p. 108019, Aug. 2023, doi: 10.1016/J.COMPAG.2023.108019.

[95] I. Saradopoulos, I. Potamitis, S. Ntalampiras, A. I. Konstantaras, and E. N. Antonidakis, “Edge Computing for Vision-Based, Urban-Insects Traps in the Context of Smart Cities,” Sensors, vol. 22, no. 5, p. 2006, Mar. 2022, doi: 10.3390/S22052006.

[96] Y. Li and J. Yang, “Few-shot cotton pest recognition and terminal realization,” Comput Electron Agric, vol. 169, p. 105240, Feb. 2020, doi: 10.1016/J.COMPAG.2020.105240.

[97] Y. Zhang and C. Lv, “TinySegformer: A lightweight visual segmentation model for real-time agricultural pest detection,” Comput Electron Agric, vol. 218, p. 108740, Mar. 2024, doi: 10.1016/J.COMPAG.2024.108740.

[98] S. M. Fasih, A. Ali, T. Mabood, A. Ullah, M. Hanif, and W. Ahmad, “Fruit Fly Detection and Classification in IoT Setup,” Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 14108 LNCS, pp. 593–607, 2023, doi: 10.1007/978-3-031-37117-2_40/TABLES/6.

[99] K. Bjerge, H. M. R. Mann, and T. T. Høye, “Real-time insect tracking and monitoring with computer vision and deep learning,” Remote Sens Ecol Conserv, vol. 8, no. 3, pp. 315–327, Jun. 2022, doi: 10.1002/RSE2.245.

[100] D. Xu et al., “MAFIKD: A Real-Time Pest Detection Method Based on Knowledge Distillation,” IEEE Sens J, vol. 24, no. 20, pp. 33715–33735, 2024, doi: 10.1109/JSEN.2024.3449628.

[101] C. M. Badgujar, P. R. Armstrong, A. R. Gerken, L. O. Pordesimo, and J. F. Campbell, “Real-time stored product insect detection and identification using deep learning: System integration and extensibility to mobile platforms,” J Stored Prod Res, vol. 104, p. 102196, Dec. 2023, doi: 10.1016/J.JSPR.2023.102196.

[102] C. Zhou et al., “A smartphone application for site-specific pest management based on deep learning and spatial interpolation,” Comput Electron Agric, vol. 218, p. 108726, Mar. 2024, doi: 10.1016/J.COMPAG.2024.108726.

[103] J. Sütő, “Embedded System-Based Sticky Paper Trap with Deep Learning-Based Insect-Counting Algorithm,” Electronics 2021, Vol. 10, Page 1754, vol. 10, no. 15, p. 1754, Jul. 2021, doi: 10.3390/ELECTRONICS10151754.

[104] J. Suto, “A Novel Plug-in Board for Remote Insect Monitoring,” Agriculture 2022, Vol. 12, Page 1897, vol. 12, no. 11, p. 1897, Nov. 2022, doi: 10.3390/AGRICULTURE12111897.

[105] D. Brunelli, T. Polonelli, and L. Benini, “Ultra-low energy pest detection for smart agriculture,” Proceedings of IEEE Sensors, vol. 2020-October, Oct. 2020, doi: 10.1109/SENSORS47125.2020.9278587.

[106] E. Flamand et al., “GAP-8: A RISC-V SoC for AI at the Edge of the IoT,” Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors, vol. 2018-July, Aug. 2018, doi: 10.1109/ASAP.2018.8445101.

[107] L. Bompani et al., “Accelerating Image-based Pest Detection on a Heterogeneous Multicore Microcontroller,” IEEE Transactions on AgriFood Electronics, vol. 2, no. 2, pp. 170–180, Sep. 2024, doi: 10.1109/TAFE.2024.3451888.

[108] M. Rusci, L. Bompani, O. Baldoni, C. Montanari, D. Brunelli, and L. Benini, “Parallel Execution of the Viola-Jones Algorithm on MCUs for Low-Cost Automated Pest Detection,” 2023 IEEE Conference on AgriFood Electronics, CAFE 2023 - Proceedings, pp. 35–39, 2023, doi: 10.1109/CAFE58535.2023.10291634.

[109] L. Crupi, L. Butera, A. Ferrante, and D. Palossi, “A Deep Learning-based Pest Insect Monitoring System for Ultra-low Power Pocket-sized Drones,” Proceedings - 2024 20th International Conference on Distributed Computing in Smart Systems and the Internet of Things, DCOSS-IoT 2024, pp. 323–330, 2024, doi: 10.1109/DCOSS-IOT61029.2024.00056.

[110] L. Crupi, L. Butera, A. Ferrante, A. Giusti, and D. Palossi, “An Efficient Ground-aerial Transportation System for Pest Control Enabled by AI-based Autonomous Nano-UAVs,” ACM Journal on Autonomous Transportation Systems, vol. 2, no. 4, p. 16, Dec. 2025, doi: 10.1145/3719210/SUPPL_FILE/JATS-2024-0040-FILE003.MP4.

[111] J. V. de A. Porto, A. C. Dorsa, V. A. de M. Weber, K. R. de A. Porto, and H. Pistori, “Usage of few-shot learning and meta-learning in agriculture: A literature review,” Smart Agricultural Technology, vol. 5, p. 100307, Oct. 2023, doi: 10.1016/J.ATECH.2023.100307.

[112] L. Alzubaidi et al., “A survey on deep learning tools dealing with data scarcity: definitions, challenges, solutions, tips, and applications,” Journal of Big Data 2023 10:1, vol. 10, no. 1, pp. 1–82, Apr. 2023, doi: 10.1186/S40537-023-00727-2.

[113] Y. Ouali, C. Hudelot, and M. Tami, “An Overview of Deep Semi-Supervised Learning,” Jun. 2020, Accessed: Jul. 30, 2025. [Online]. Available: https://arxiv.org/pdf/2006.05278

[114] I. H. Rather, S. Kumar, and A. H. Gandomi, “Breaking the data barrier: a review of deep learning techniques for democratizing AI with small datasets,” Artificial Intelligence Review 2024 57:9, vol. 57, no. 9, pp. 1–61, Aug. 2024, doi: 10.1007/S10462-024-10859-3.

[115] Z. Yang, S. Al-Dahidi, P. Baraldi, E. Zio, and L. Montelatici, “A Novel Concept Drift Detection Method for Incremental Learning in Nonstationary Environments,” IEEE Trans Neural Netw Learn Syst, vol. 31, no. 1, pp. 309–320, Jan. 2020, doi: 10.1109/TNNLS.2019.2900956.

[116] F. Bayram, B. S. Ahmed, and A. Kassler, “From concept drift to model degradation: An overview on performance-aware drift detectors,” Knowl Based Syst, vol. 245, p. 108632, Jun. 2022, doi: 10.1016/J.KNOSYS.2022.108632.

[117] N. E. Khalifa, M. Loey, and S. Mirjalili, “A comprehensive survey of recent trends in deep learning for digital images augmentation,” Artif Intell Rev, vol. 55, no. 3, pp. 2351–2377, Mar. 2022, doi: 10.1007/S10462-021-10066-4/FIGURES/17.

[118] C. Karam, M. Awad, Y. Abou Jawdah, N. Ezzeddine, and A. Fardoun, “GAN-based semi-automated augmentation online tool for agricultural pest detection: A case study on whiteflies,” Front Plant Sci, vol. 13, p. 813050, Sep. 2022, doi: 10.3389/FPLS.2022.813050/BIBTEX.

[119] J. Cabrera and E. Villanueva, “Investigating Generative Neural-Network Models for Building Pest Insect Detectors in Sticky Trap Images for the Peruvian

Horticulture," Communications in Computer and Information Science, vol. 1577 CCIS, pp. 356–369, 2022, doi: 10.1007/978-3-031-04447-2_24/TABLES/3.

[120] C. Y. Lu, D. J. Arcega Rustia, and T. Te Lin, "Generative Adversarial Network Based Image Augmentation for Insect Pest Classification Enhancement," IFAC-PapersOnLine, vol. 52, no. 30, pp. 1–5, Jan. 2019, doi: 10.1016/J.IFACOL.2019.12.406.

[121] S. Niu, Y. Liu, J. Wang, and H. Song, "A Decade Survey of Transfer Learning (2010–2020)," IEEE Transactions on Artificial Intelligence, vol. 1, no. 2, pp. 151–166, Oct. 2020, doi: 10.1109/TAI.2021.3054609.

[122] S. Tian, L. Li, W. Li, H. Ran, X. Ning, and P. Tiwari, "A survey on few-shot class-incremental learning," Neural Networks, vol. 169, pp. 307–324, Jan. 2024, doi: 10.1016/J.NEUNET.2023.10.039.

[123] X. Li, X. Yang, Z. Ma, and J. H. Xue, "Deep metric learning for few-shot image classification: A Review of recent developments," Pattern Recognit, vol. 138, p. 109381, Jun. 2023, doi: 10.1016/J.PATCOG.2023.109381.

[124] S. V. Nuthalapati and A. Tunga, "Multi-Domain Few-Shot Learning and Dataset for Agricultural Applications," in Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops, 2021, pp. 1399–1408.

[125] M. Rezaei, D. Diepeveen, H. Laga, M. G. K. Jones, and F. Sohel, "Plant disease recognition in a low data scenario using few-shot learning," Comput Electron Agric, vol. 219, p. 108812, Apr. 2024, doi: 10.1016/J.COMPAG.2024.108812.

[126] H. Liu, Y. Zhou, B. Liu, J. Zhao, R. Yao, and Z. Shao, "Incremental learning with neural networks for computer vision: a survey," Artif Intell Rev, vol. 56, no. 5, pp. 4557–4589, May 2023, doi: 10.1007/S10462-022-10294-2/FIGURES/6.

[127] K. Shaheen, M. A. Hanif, O. Hasan, and M. Shafique, "Continual Learning for Real-World Autonomous Systems: Algorithms, Challenges and Frameworks," Journal of Intelligent and Robotic Systems: Theory and Applications, vol. 105, no. 1, pp. 1–32, May 2022, doi: 10.1007/S10846-022-01603-6/METRICS.

[128] L. Yang, Z. Luo, S. Zhang, F. Teng, and T. Li, "Continual Learning for Smart City: A Survey," IEEE Trans Knowl Data Eng, vol. 36, no. 12, pp. 7805–7824, 2024, doi: 10.1109/TKDE.2024.3447123.

[129] X. Yang, Z. Song, I. King, and Z. Xu, "A Survey on Deep Semi-Supervised Learning," IEEE Trans Knowl Data Eng, vol. 35, no. 9, pp. 8934–8954, Sep. 2023, doi: 10.1109/TKDE.2022.3220219.

[130] L. Gomez-Zamanillo et al., "Semi-Supervised Approach for Automatic Counting of Whiteflies With Small Annotated Dataset," IEEE Access, vol. 13, pp. 55586–55598, 2025, doi: 10.1109/ACCESS.2025.3552195.

[131] D. J. A. Rustia et al., "Online semi-supervised learning applied to an automated insect pest monitoring system," Biosyst Eng, vol. 208, pp. 28–44, Aug. 2021, doi: 10.1016/J.BIOSYSTEMSENG.2021.05.006.

[132] H. Visserman, "Semi-Supervised Hierarchical Classification for Insect Monitoring," M.Sc. thesis, Radboud University Nijmegen, Netherlands, 2024.

3 CHAPTER III. LOW-POWER EDGE DEVICE FOR HALYOMORPHA HALYS DETECTION

The work in this chapter is adapted from my first-author peer-reviewed articles: 1, 2, 4 as detailed below. From the collaborative project-wide paper (3), only Section V, corresponding to my contribution, is included in this chapter of the thesis. Section 3.2 is primarily derived from article 1, Section 3.3 from article 2, and Section 3.4 from article 3 (only Section V).

1. **Kargar, A**., Wilk, M. P., Zorbas, D., Gaffney, M. T., & O'Flynn, B. (2022), "A novel resource-constrained insect monitoring system based on machine vision with edge AI," In *2022 IEEE 5th International Conference on Image Processing Applications and Systems (IPAS)* (pp. 1-6). IEEE, DOI: 10.1109/IPAS55744.2022.10052895.
2. **Kargar, A**., Zorbas, D., Tedesco, S., Gaffney, M., & O'Flynn, B. (2024). "Detecting Halyomorpha halys using a low-power edge-based monitoring system," *Computers and Electronics in Agriculture*, 221, 108935, DOI: 10.1016/j.compag.2024.108935.
3. Almstedt, L., Betti Sorbelli, F., Boom, B., Calvini, R., Costi, E., Dinca, A., Ferrari, V., Giannetti, D., Ichim, L., **Kargar, A**., et al., (2024), "Beyond the naked eye: Computer vision for detecting brown marmorated stink bug and its punctures," *IEEE Transactions on AgriFood Electronics*, vol. 3, no. 1, pp. 6-17, DOI: 10.1109/TAFE.2024.3429537.

Dataset:

4. **Kargar, A**., Zorbas, D., Gaffney, M., O'Flynn, B., & Tedesco, S. (2024). Image Dataset of Brown Marmorated Stink Bug (BMSB) or Halyomorpha Halys (HH)-V1 [Data set], *Zenodo*, DOI: 10.5281/zenodo.7887046.

3.1 Introduction

This chapter presents the design and implementation of a low-power, MCU-based system for insect detection, focusing on the Brown Marmorated Stink Bug (BMSB). One of the goals of this thesis is to explore the feasibility of deploying a DL

model onto resource-constrained MCUs and assess the system functionality in terms of power consumption, memory requirements, image capture, data transfer, and real-time inference. A lightweight image processing algorithm along with a DL-based classification model was deployed on an MCU-based camera module, and the system was tested in outdoor environments to assess the device performance and understand deployment challenges in real-world environments. Additionally, this phase provided the opportunity to collect real-world insect image datasets under realistic field deployment conditions, which allowed me to refine the algorithm and model design and support the next phase of the project.

This chapter also addresses the first objective of this PhD thesis by developing and validating a low-power edge-based prototype device for insect detection and identification using resource-constrained MCUs.

3.2 Device design for edge-based insect monitoring

This chapter focuses on the development of a stationary edge device, also referred in the text as “end-device” or “sensor unit”. The edge device is a low-cost, low-power, MCU-based system that incorporates an inexpensive camera. It is capable of taking images and processing them. The system can be IoT-enabled so that the output could also be sent to the cloud via a telecommunications link for further processing and, in the long term, shape the growers’ IPM strategy. The proposed general system block diagram is shown in Figure 3.1.

In order to meet the requirements of the autonomous insect identification and the counting of the insect pest of interest, i.e., the BMSB, the proposed device should be equipped with a standard commercially available insect trap that the growers normally use in manual monitoring of their orchards. The proposed system must also meet certain capital expenditure (CAPEX) and operational expenditure (OPEX) requirements, where CAPEX refers to the initial hardware and deployment cost, and OPEX refers to ongoing costs such as power consumption, maintenance, and communication. These include low cost, low power consumption, and low maintenance. These requirements come from the practical needs of agricultural settings where low cost, low power, and minimal maintenance are essential for long-

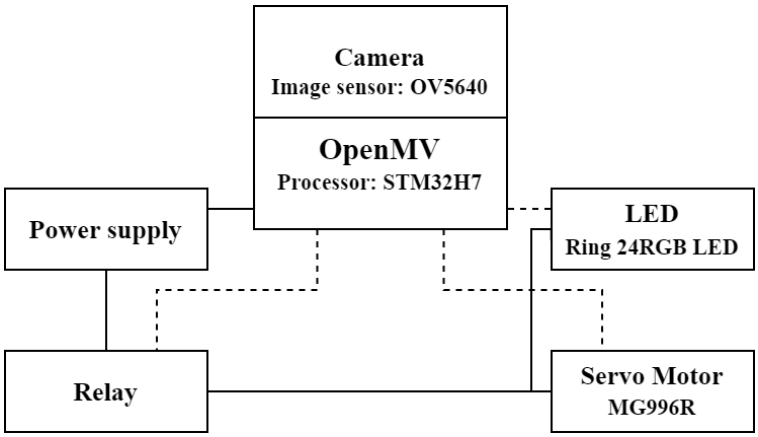


Figure 3.1. Block diagram of the proposed edge-based insect monitoring system showing the main components including the camera, MCU board, LED, servo motor and power management.

term deployment. Based on Table 2.2, MCUs exhibit a much lower cost and power consumption than other solutions, thus, they can be considered as an excellent candidate for insect monitoring applications deployed in remote areas. However, it is important to examine whether their severely limited processing resources are sufficient to provide reliable results for the detection of BMSB.

To achieve image acquisition from insects, an IoT sensor system that incorporates both a trap and a camera was developed. The camera can take images of the trap at regular time intervals. The device processor (e.g., MCU) can process the images to detect and identify the insects. The proposed system details are elaborated in the following subsections.

3.2.1 Hardware platform

One of the main factors that drives many of the system requirements for the final prototype is the camera specification. The balance between camera resolution and energy consumption of the overall system (often battery powered for remote deployment) should be considered in the system design. Similarly, the processor must be powerful enough to process the images, but not too powerful for the needs of the task.

Based on the identified requirements, the OpenMV Cam STM32H7 Plus[9] module was selected for this project (see Figure 3.2). OpenMV is an MCU-based image acquisition board with an STM32H743II ARM Cortex M7 processor with 32

[9] https://openmv.io/collections/products/products/openmv-cam-h7-plus

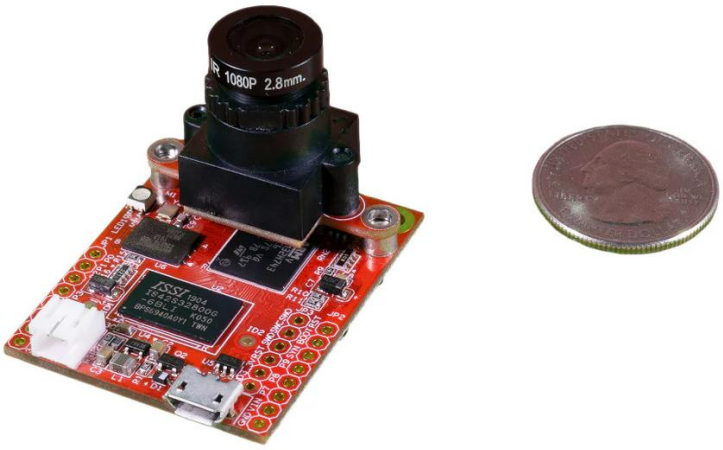


Figure 3.2. The MCU-based OpenMV H7 Plus board used in this work, equipped with an STM32H743II ARM Cortex-M7 processor and a 5 MP camera module. (source: https://openmv.io)

MB of SDRAM, 1MB of SRAM, 32 MB of external flash and 2 MB of internal flash running at 480 MHz. This is a small, inexpensive, and low-power system that incorporates an OV5640 imaging sensor that can acquire a 5MP resolution image. In the developed system, images of 2MP resolution are used to achieve lower power consumption while still being able to capture and identify the smallest features of the BMSB, such as antennas and legs.

In this thesis, all the proposed DL models were implemented using TensorFlow framework. Afterwards, the trained model was exported as a TensorFlow Lite (tflite) model, a compatible format for the OpenMV board to load the model. A MicroPython script was developed to handle all the processes including image capturing, applying the DL model on the captured image, as well as data saving and transferring.

3.2.2 System design

The operation of the device must not have an adverse impact on the trap's fundamental requirement, e.g., its attractiveness to insects. The trap attractiveness to insects could be reduced if the device is not designed appropriately. For example, the effectiveness of the trap can be affected by its positioning or the amount of light it receives. Therefore, these factors must be considered during the design of the system as discussed below [1].

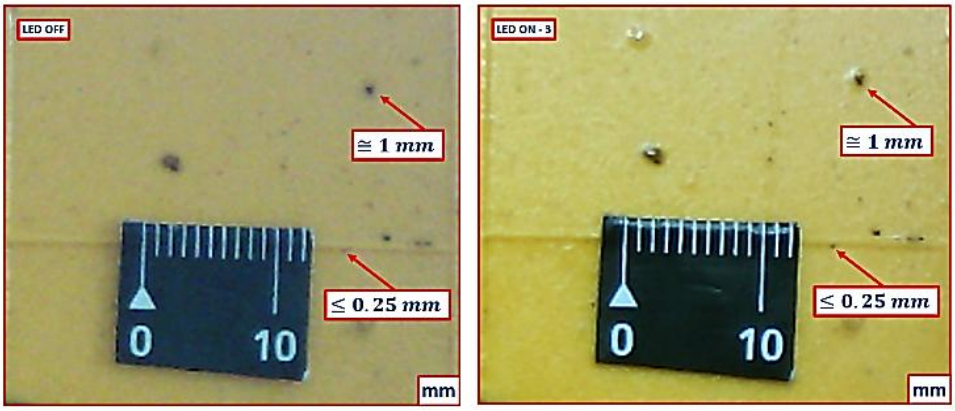


Figure 3.3. Illumination impact on image quality; Left: darkest conditions, Right: brightest conditions (In Lab).

1) Illumination

Vision sensors are designed to capture optical data that their sensor arrays are exposed to. Illumination plays a critical role in the sensors' effectiveness. The less controlled the environment is, the more difficult it is to extract the desired information from the images. One way to control the environment is to introduce artificial illumination to the scene, thus, introducing a level of control over the environment in which the vision sensors operate. This approach is also widely used in machine vision applications, which however, is associated with high power drain, a parameter that needs to be considered in the system design as well [2][3].

Nevertheless, the positive impact of illumination is clearly visible in Figure 3.3, where illumination provides an enhanced image quality that can directly affect image processing and DL performance. Moreover, illumination allows for image acquisition at night. Timing the system to operate at night allows end-users to control lighting conditions and eliminate ambient lights and shadows; therefore, the captured images have the same conditions in terms of lighting parameters, such as light intensity and brightness, which is important for the operation of the DL approach.

2) Servo Motor Integration

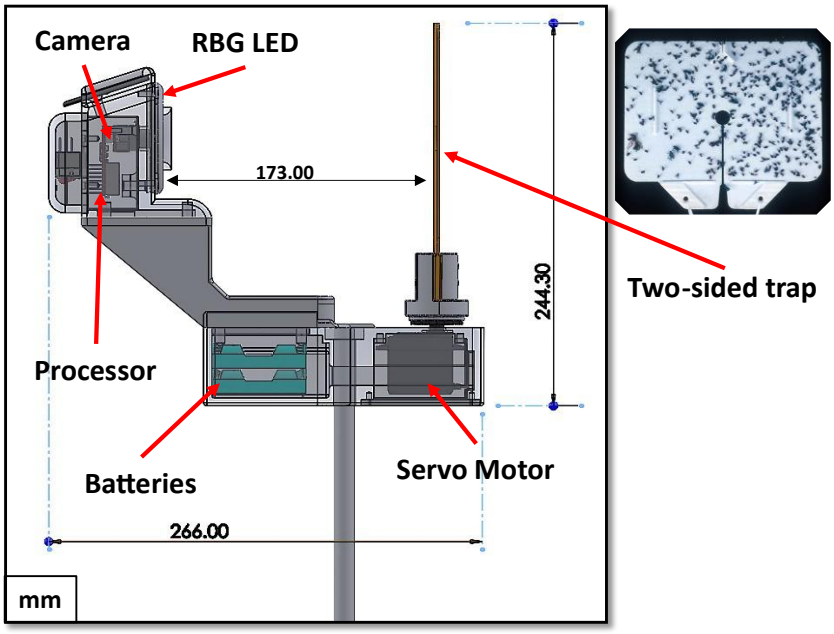
Commercial double-sided sticky traps are widely used by farmers to track insect populations in orchards, i.e., insects are captured on both sides. This can pose a significant challenge for a camera-based sensor. Firstly, both sides of the trap must be captured and analyzed. Secondly, the camera must not be occluding either of the trap's two sides, otherwise its effectiveness could be adversely affected.

The above-mentioned problem can be solved by introducing an approach that utilises a low-cost servo motor. The servo motor can hold and rotate the trap. When the camera is inactive, the trap's surface is parallel to the camera's optical axis, thus avoiding occlusions, as shown in Figure 3.4a. When the system wakes up for image acquisition, the servo motor rotates the trap so that the camera can capture images from both sides of the trap.

The purpose of the double-sided trap mechanism is to increase the probability of capturing insects (as the captured area is doubled) and, at the same time, to make it more approachable to insects since the sticky area is not shadowed by the camera body.

Figure 3.4a illustrates the first version of the image acquisition and data processing system of the project.

This automated insect monitoring tool was developed and deployed in an orchard in Italy for a trial period (see Figure 3.4b). Figure 3.4b (left) shows the first designed and deployed version of the device, while the right image shows the updated version, which was improved based on the experiences from the initial deployment.



(a) System design and its components.

Deployed device in an orchard; left: First version; right: Updated version.

Figure 3.4. Overview of the proposed edge-based insect monitoring device, showing the overall device design and the main components.

The color was changed to white to reduce overheating during outdoor operation, with some minor design adjustments to improve durability and usability. The system consists of a double-sided sticky trap[10] and a pheromone lure is applied on it to attract BMSBs. The pheromone lure attracts only BMSBs, which reduces the probability of other insects being trapped, also known as bycatch.

A white coloured trap was employed because it provides the highest contrast with mottled brownish grey BMSBs and, thus, improves the detection accuracy. More specifically, using a uniform white trap reduces background clutter, which simplifies feature extraction. As a result, a simpler DL model can be used to extract features. This

[10] https://www.andermattuk.com/rebell-white-trap

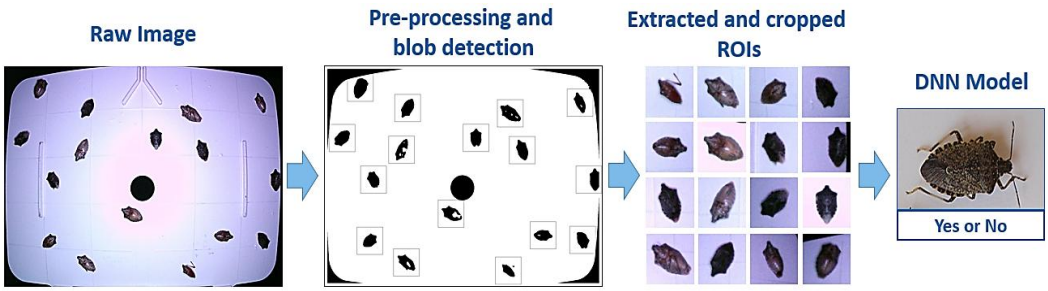


Figure 3.5. System pipeline: from raw input image to ROI classification.

leads to reduced computational requirements and consequently lower power consumption, which makes it more suitable for edge-based deployments [4].

In addition, as intermittent electricity supply may not be available in remote orchards, the system uses batteries, which can be recharged using an integrated solar panel.

In this first version of the device, the captured images are transferred via an LTE (4G) connection to the cloud.

3.2.3 Preliminary image processing system algorithm

The proposed system algorithm is shown in Figure 3.5. This algorithm first attempts to extract the Region of Interest (RoI) or multiple RoIs from the captured image of the trap. It then runs a Convolutional Neural Network (CNN) based model for insect classification. The area of the RoI is such that the insect of interest almost entirely fills its area. In this way, only small portions of captured images (RoIs with suspected BMSBs) are used for further processing and transferring, which reduces system requirements significantly. The steps of the proposed algorithm are shown in Figure 3.5 which depicts the image processing pipeline from the raw input image to the end results using an example input image.

It must be noted that the algorithm under development must be as “light” as possible so as to run on the edge device which is resource-constrained in terms of processing power, memory, and energy. In this context, in the first phase (pre-

processing), two simple image processing techniques were used to prepare the raw input image (see Figure 3.6). These techniques are as follows:

1- Smoothing filter: The smoothing filter/Gaussian blur with the 3x3 kernel was applied to the input image to reduce the high-frequency noise component and details [5].
2- Since there is a strong difference in color and intensity between black (insects) and the white background color (trap), histogram-based thresholding was utilized to divide the image into two classes, i.e. background and foreground. To this end, Otsu's method [6] was used to find the optimal threshold value.

Then, a simple blob detector (available in OpenMV libraries [7]) was utilized to detect and extract possible RoIs from the image. The blob detector finds a blob based on two main parameters, including Pixel Area and Circularity. Pixel Area was set based on the minimum and maximum size of the studied insect which is approximately 12-17 mm in length [8] and Circularity was adjusted to approximate the insect's shape to an Elliptical Shape.

Finally, the extracted RoIs were passed to a lightweight deep learning model for insect classification. In this phase, the extracted small images are fed to the deep learning model to label with the input RoI with either a '1' or '0' depending on whether there is a BMSB present in the extracted image or not. As mentioned before, the deep learning model should be as small as possible so that it can fit in the MCU's memory. For this reason, parameter quantization was performed on the model, which converts floating-point numbers to other data types. In this study, all the weights and biases were converted to 8-bit integers [-128, ..., 127] instead of float32, which significantly

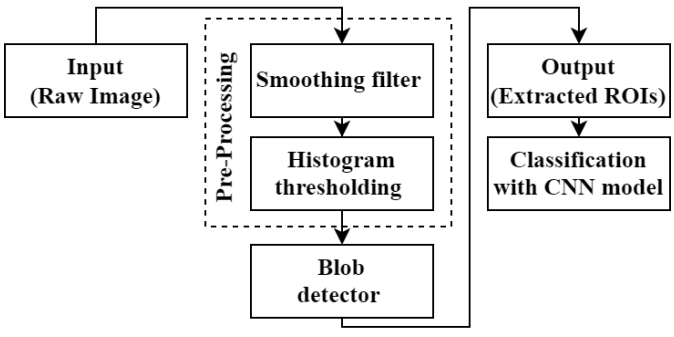


Figure 3.6. Diagram of the proposed image processing and classification pipeline executed on the MCU-based edge device.

reduces memory and computing requirements. The (int8) quantized CNN model is proposed with the model size of just 57 KB to show the feasibility of such systems.

The model architecture is shown in Figure 3.7. This is a CNN-based model comprising two convolutional layers with 32 and 16 channels and a kernel size of 3x3. Each convolutional layer is followed by a max-pooling layer. In the final part, there is a fully-connected layer connected to the output layer (softmax). This model has 51,730 parameters overall.

3.2.4 Experimental results

In this section, the performance of the proposed prototype is evaluated based on preliminary results. It is worth noting that the proposed algorithm was implemented on the OpenMV MCU platform, and all the computation and analysis were carried out on this board. Moreover, it must be noted that this task was carried out before the first field deployment. Thus, real-world images of the BMSB in orchards were not available. For this reason, the initial dataset was manually created in laboratory conditions using dead (deep-frozen) BMSB specimens that had been collected in the field by project collaborators. The specimens were manually glued to the trap, as shown in Figure 3.5, simulating a deployment capture scenario. As mentioned in the previous section, in the first phase, the proposed system finds and extracts RoIs from the whole input image.

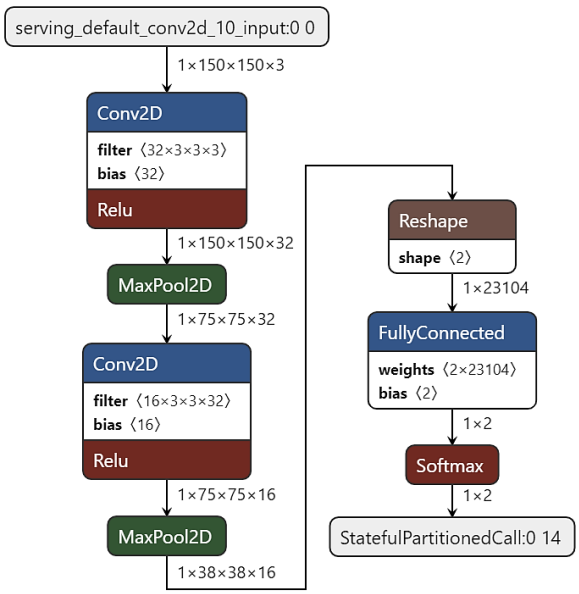


Figure 3.7. Architecture of the proposed CNN model used for insect classification on the MCU platform.

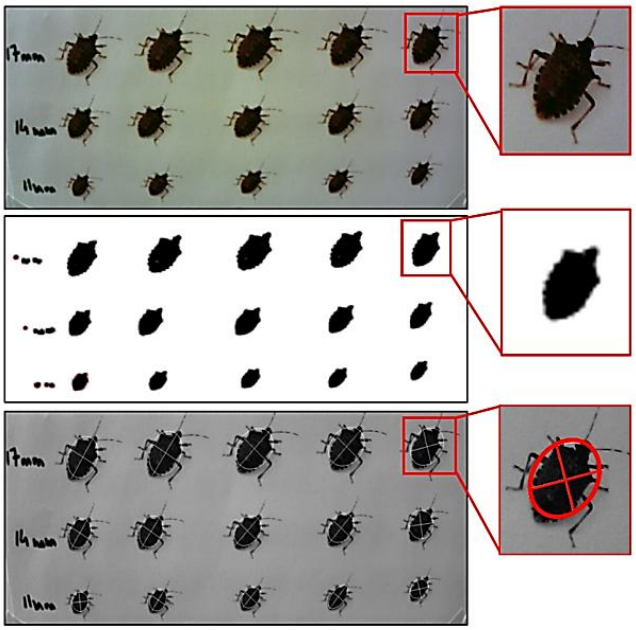


Figure 3.8. a) Original input image from artificial insects, b) Image pre-processing, c) Blob detection on input image in greyscale.

Laboratory experiments (see Figure 3.8) showed that the proposed method for RoI extraction was sufficient for successfully detecting all desired blobs based on Pixel Area and Circularity parameters.

For the DL model, Adam and Binary Cross-entropy are chosen as the optimizer and loss function, respectively. The kernel size of 3x3, same padding with strides 1 are set for 2-D convolutional layers, and their activation functions are Rectified Linear Unit (ReLU). The output layer activation function is Softmax and a dropout layer with the rate of 0.25 is used to tackle the overfitting during the training phase. Figure 3.9 shows the model accuracy and loss curves. Based on this figure, model accuracy becomes steady at around 97% after 15 epochs.

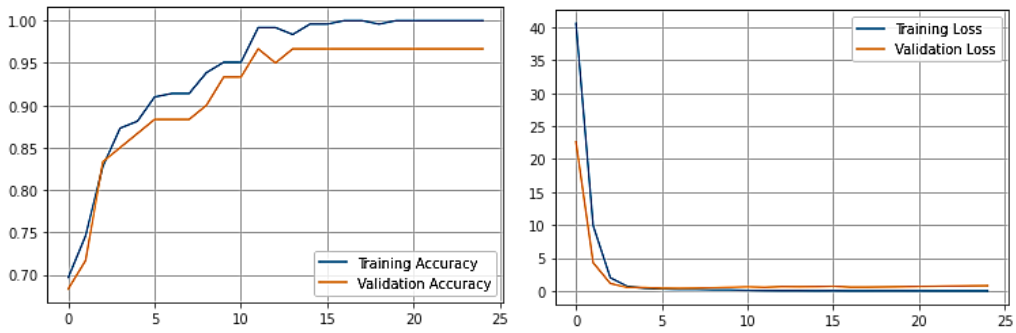


Figure 3.9. a) Training and validation accuracy, b) Training and validation loss.

The model performance is reported in Table 3.1. The model with parameters in float32 data type achieved 96.7% accuracy and over 97% for recall, precision, and F1-score with the size of 210 KB. On the other hand, the quantized model with parameters in the int8 data type achieved lower accuracy, i.e. 88.3%. However, the model size was significantly reduced; by a factor of approximately 3.7, and its size in memory was reduced to 57 KB.

The quantized model was embedded in the MCU's firmware. The proposed prototype system was deployed in an orchard in the Emilia-Romagna region in Italy during the summer months. Figure 3.2 reports the model classification performance from this deployment. The data collection was carried out over a period of two months during which the BMSB was present in the orchard. Figure 3.2 shows that the classification accuracy in real-world field deployment was lower as compared to the corresponding experiments in the lab, i.e. shown in Table 3.1. This result was expected, as the images acquired in the orchard were contaminated by noise and contained by-catch (e.g., random non-BMSB insects on the trap) to a greater extent. In this regard, some of the classified images captured at the deployment site in Italy are shown in Figure 3.10. There are some false positives or negatives in the field deployment results. For example, regarding the second image in the first row of Figure 3.10, it was predicted as BMSB while it is not. This is because it has the same shape as the BMSB, but it has a different colour, and it is not in the dataset used for training. Also, a similar result occurred when BMSB lands unevenly on the trap. Hence, further improvement of the deep learning model and training it with images captured in real-world conditions can further increase the accuracy.

Table 3.1. Proposed CNN model laboratory results.

	Accuracy (%)	**Recall (%)**	**Precision (%)**	**F-score (%)**	**Model Size (kB)**
Float32	96.7	97.6	97.6	97.6	210
Int8	88.3	95.1	88.6	91.8	57

Table 3.2. Proposed CNN model field deployment results.

	Accuracy (%)	**Recall (%)**	**Precision (%)**	**F-score (%)**
Int8	70.1	77.2	81.6	79.4

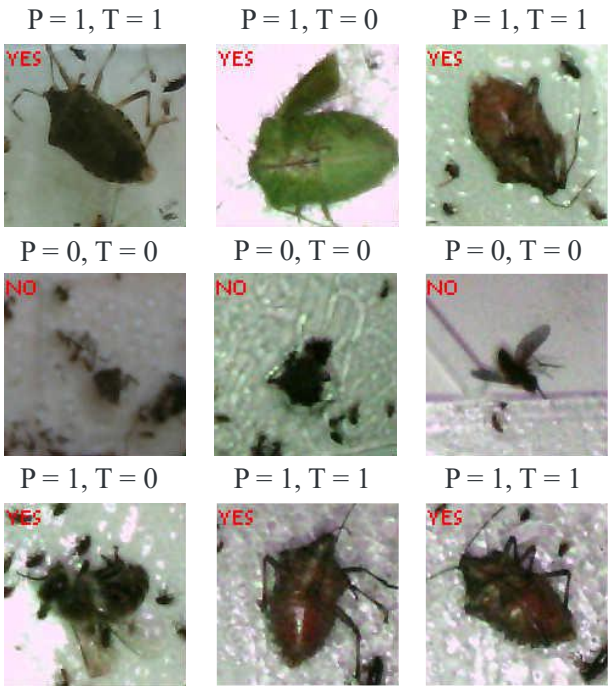


Figure 3.10. Example results obtained from the deployed prototype device showing detected and classified insects.

These results show the output of the early stage, edge-compatible, image processing algorithms. The proposed method has shown promising results. It demonstrates the potential that this system has in the application space considered, i.e. automated pest insect monitoring as part of an IPM strategy.

3.2.5 Discussion

A novel prototype system for automated insect pest monitoring with a commercial trap was proposed. It is a low-cost edge device with Edge AI for detecting and identifying insects of interest, in this case the BMSB. An OpenMV Cam H7 Plus was selected for the proof-of-concept prototype in this study to satisfy the balance between cost, energy consumption and processing power. As this is a resource constrained MCU-based board, a lightweight image processing algorithm is proposed. The proposed system was validated in field deployment in an orchard in Italy during the summer months. The preliminary results showed promising results, with BMSB classification accuracy of over 70 percent.

The accuracy of the Edge AI is expected to increase once the CNN updated model is retrained with a dataset containing images from real-world field deployment conditions. Also, it must be noted that as an edge-based system, the power

consumption aspect is the next stage of this study, even though it is expected the energy consumption of the proposed MCU-based system to be multiple times lower than that of the RPi-based solutions.

3.3 Algorithm design for edge-based insect monitoring

This section focuses on the algorithm design and optimization of the edge-based AI framework. The performance of the proposed model is evaluated from different aspects and compared with existing methods to demonstrate its efficiency and suitability for real-world deployment.

3.3.1 Device operational cycle

The proposed system works in repeated cycles, where in each cycle, a picture of a double-sided trap is taken, and image analysis is carried out using an image processing algorithm and a DL model. The relevant results are transmitted over a communication link. The device then enters into a long deep sleep mode until the next cycle starts. This subsection presents details regarding the proposed system processing implementation, the employed techniques, as well as the DL model.

Figure 3.11 shows the system timing diagram for an image capture and processing cycle. In order to keep track of the insect population trends during the growing season, the system wakes up twice in 24 hours. This happens one hour after sunset and one hour before sunrise. In between cycles, the system remains in deep sleep mode to decrease energy consumption. Since sunset and sunrise vary during the year, they are calculated based on the location and the time zone of the deployment.

The system operation is divided into several stages in each operational cycle as shown in Figure 3.12 and summarized as follows:

Capture of images: The system takes images from both sides of the trap using the integrated RGB camera. As mentioned earlier in the text, images are captured during nighttime hours, one hour before sunrise and one hour after sunset using the LED light. By doing so, the lighting condition is controlled and remains consistent in all images, which is important for the object detection and classification processes. The next step is the image analysis phase, which consists of two sub-steps, including detection and classification.

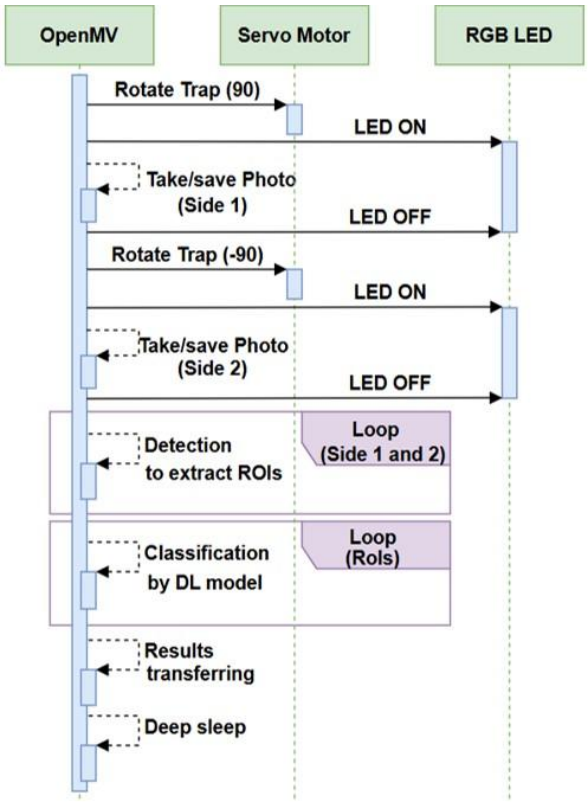


Figure 3.11. Operational cycle of the proposed system, illustrating the sequence of operations including trap rotation, image capture, image processing, results transmission, and sleep mode.

These two sub-steps are explained in the following paragraphs:

Processing and detection: In this step, the system executes an image processing algorithm on the two captured images (i.e., two trap sides) to detect and extract the potential RoIs, where an RoI is an area where the target insect is likely to be present. To this purpose, as shown in Figure 3.12, the algorithm converts the RGB image to grayscale. Then, a smoothing filter (e.g., mean blurring filter) is used to reduce noise and some details, such as insect legs or antennas, windblown dust, leaves, or tiny insects (step 1). This is only used to locate RoIs, the detected RoIs are then cropped from the raw image, which preserves all original details for further analysis by the CNN-based model in the next stage. Since there is a strong difference in color and intensity between dark (insects) and the white background color (trap surface), histogram-based thresholding is utilized to divide the image into two classes, i.e., background and foreground. Therefore, in step 2, Otsu's method is used to find an optimal threshold from the image histogram to convert the image to a black-and-white image and separate the foreground (insects) and background (trap surface). The last stage of the process is the use of a blob detector to locate the potential areas where the

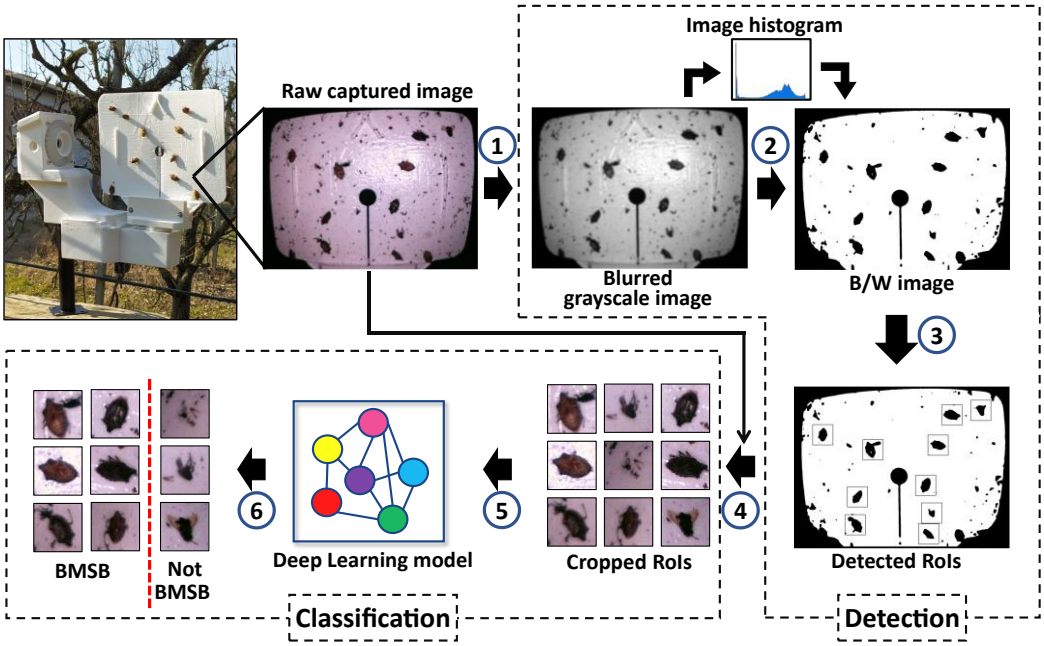


Figure 3.12. Image processing algorithm, from the raw captured image to classified RoIs executed on the MCU-based device.

target insect may be present (step 3). The blob detector detects blobs based on primary criteria, called size thresholds. Blobs are filtered out if their bounding box parameters are lower or higher than these thresholds. These thresholds are defined and adjusted by the minimum and maximum dimensions of the target insect size. Since the distance between the camera and the trap is fixed, these thresholds are valid and reliable. The detected areas (RoIs) are cropped from the original image for further analysis and are sized to 150×150 pixels (step 4).

Deep learning classification: In step 5, the extracted RoIs are fed into a DL model for classification. Based on the model result, the cropped images are labelled as '1' or '0', indicating whether the image belongs to a BMSB or not (step 6). The critical point for this phase is that the model should be optimized, be small, and be sufficiently efficient to run and fit in resource-constrained MCUs.

Results transfer: In this phase, the results (i.e., the number of the target insects on the trap along with the cropped images) are transferred to growers or the cloud for further decision-making. To reduce power consumption associated with communications, instead of sending images of the whole trap, the system only transfers small (in terms of data size) results as regards the presence of the relevant insect species. The JPEG compression method with a quality of 90% is used to compress the images, which results in a typical image size of a few KB (≤4 KB). This

allows the system to send data using a low data rate communication technology, such as LoRa, which is low-cost and long-range, and aligns with the requirements of this application [9].

Sleep mode: Finally, the system goes to deep sleep mode to reduce power consumption until the next scheduled image acquisition round needs to take place.

3.3.2 Classification model architecture

The goal of the classification model is to recognize whether the cropped image contains a BMSB or not. The proposed model is based on a CNN which is specifically designed for image classification. The main part of the CNN model is the convolutional layers, which are responsible for recognizing patterns, edges, and textures as well as for extracting this information from the input image. These layers scan an input image using small kernels (also called filters) and create feature maps, in which each kernel extracts specific features and generates a corresponding feature map. Then, an activation function is applied to the feature maps to introduce non-linearity into the network, and thus, allows the approach to recognize complex patterns and relations in data. In the final stage, which is known as the classifier part, the extracted feature maps are combined and analyzed to make a decision on the content of the input image.

This subsection describes the development of the DL model to meet the criteria of small size and low complexity for deployment on resource-constrained MCUs. For this goal, Depthwise Separable Convolution (DwSConv) was employed instead of Standard Convolution (SConv) which is generally used to capture and extract important features and patterns from an input image. DwSConv is an effective method for reducing computations and parameters while maintaining a high accuracy level [10]. DwSConv factorizes a Standard Convolution into two steps: a depthwise convolution followed by a pointwise convolution. This significantly decreases the amount of computation, as pointed out in [10]. For example, a DwSConv with a kernel size of 3 decreases the computational cost by around 9 times.

Moreover, skip connections were used to improve the proposed model performance by allowing data to flow easily from input to output and combining the

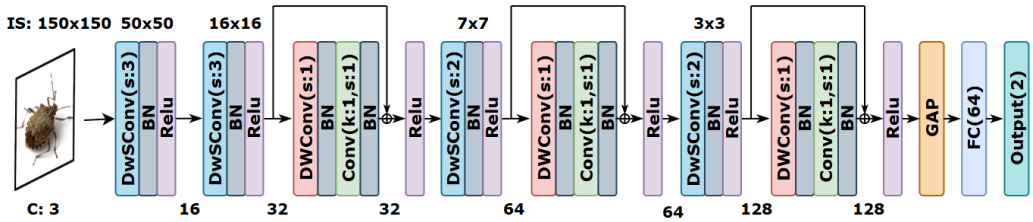


Figure 3.13. Proposed classification model; IS: Input Size, C: Number of Channels, DwSConv: Depthwise Separable Convolution, Conv: Convolution, BN: Batch Normalization, GAP: Global Average Pooling, FC: Fully Connected.

features from different layers and, also, to help reduce the problem of vanishing gradients during network training [11].

The proposed architecture is depicted in Figure 3.13. As mentioned, DwSConv was used instead of SConv to extract features from an input image. In this architecture, four DwSConv (with a kernel size of 3 and a stride of 3 or 2) were used to extract features and reduce the input dimension. A residual/skip block was used after each DwSConv, except for the first one. This block is a DwSConv that adds its input and output before the activation function (ReLU). Therefore, in this architecture, the dimension size decreases from 150×150 in input to 3×3 after the last convolution layer. The channel number correspondingly increases from 16 in the first DwSConv to 128 in the last one. For the classifier part, a Global Average Pooling (GAP) layer was used, followed by a Fully Connected (FC) layer. This is a small and lightweight CNN-based model that has approximately 46,000 parameters overall.

In order to deploy the trained model on MCUs with extremely limited flash storage, it was quantized to 8-bit integers. Quantization reduces the model size, however, it may cause a slight reduction in accuracy compared to the float data type model, which uses 32 bits of precision. Based on experiments, the accuracy difference between the quantized and the unquantized option was approximately 0.2%, thus, the accuracy of the quantized model of the proposed network remained the same as with the unquantized model, while its model size decreased by over two times.

The following bullet points highlight the key considerations of the model:

- Depthwise Separable Convolution instead of Standard Convolution: this reduces computations by factorizing the process, significantly decreasing computational cost.

- Skip connections: skip connections enhance performance by facilitating data flow, combining features from different layers, and addressing vanishing gradient issues during training.
- Global Average Pooling instead of a Flatten layer: this decreases parameter numbers, leading to a substantial reduction in computation and model size.
- Quantizing: the model weights were quantized to 8-bit integers from the original 32-bit float data, which is critical for deploying the model on memory-constrained MCUs.

3.3.3 Experimental setup and configuration

This section describes how the experiments were carried out, presents the evaluation results, and compares them to other similar approaches in literature. In order to create a comprehensive evaluation, several important aspects relevant to edge-based systems were investigated.

To evaluate the power consumption performance of the approaches, six BMSBs were considered on each side of the trap and the device was powered using an Otii Arc Pro power analyzer[11]. The current and power consumption of the system were measured at different phases of operation using the power analyzer. Besides, several CNN-based architectures, such as LeNet [12], MobileNetV1 [10], MobileNetV2 [13], SqueezeNet [14], and VGG16 [15] were trained and evaluated on the BMSB_Dataset (described in Section 3.3.5) and used to compare the performance of the different approaches. MobileNetV1, MobileNetV2 and SqueezeNet are lightweight network models that are widely used in mobile and edge-based systems. Besides these models, LeNet (as one of the simplest network models) and VGG16 (as a bigger and more complex model) were also trained to evaluate their performances and the possibility of using them on MCUs. All models were deployed and evaluated on an MCU-based OpenMV board to assess their feasibility for resource-constrained edge devices.

As there are two classes (‘Yes’ and ‘No’), binary cross-entropy was used as a loss function. To reach the final optimized model, hyperparameters were tuned by varying epochs, batch size, and learning rate. Therefore, the Adam algorithm was used

[11] https://www.qoitech.com/otii-arc-pro

as an optimizer with a learning rate of 0.005, and the epochs and batch size were set to 200 and 64, respectively. ReLU6 was used as an activation function, and the dataset was divided into 80% and 20% for training and test sets, respectively. In addition, dropout and spatial dropout techniques were used to prevent overfitting during model training.

The classification performance and computational resource efficiency of the proposed model are evaluated using standard metrics defined in Appendix A. For the detection phase, the accuracy was calculated by averaging the ratios of the correct number of detected RoIs to the actual number of RoIs on each image as follows:

$$Accuracy = \frac{1}{N}\sum_{i=1}^{N}\frac{d_i}{n_i} \tag{3.1}$$

where N is the number of images, d_i is the number of correctly detected RoIs of the ith image and n_i is the number of actual RoIs of the ith image.

3.3.4 Datasets

As mentioned in the previous section, the image analysis process is divided into two separate functions, including detection and classification. In the detection phase, the system analyzes images sized at 1600×1200 pixels to extract RoIs, while in the classification phase, the system analyzes 150×150 pixels cropped images to recognize BMSBs. Thus, two separate datasets were created for these two phases; the first one was used to evaluate the performance of the detection phase, and the second one was used for the classification phase. The detection dataset was used to evaluate the performance of the image processing algorithm responsible for detecting potential insect areas on the trap surface. For this aim and because the system had only been deployed in the orchard at that stage, dead BMSB specimens were used and glued on the surface of the trap to create the first dataset of insect images in the laboratory. In addition, some images of real BMSB captured by the deployed system were added to this dataset named Trap-Dataset. This dataset contains approximately 40 laboratory-captured images and 70 images from the deployed system in the orchard, all captured at a resolution of 2MP (1600×1200) by the developed system. These images were used to evaluate the ability of the detection algorithm to correctly identify ROIs under both controlled laboratory and real-world conditions.

The second dataset (BMSB-Dataset) was created by using images of insects gathered from the deployed system, available on the internet, and cropped BMSB images from Trap-Dataset. Augmentation methods were also employed to increase the number of images. In this study, rotation, horizontal flipping, and vertical flipping were used for augmentation. To this end, the dataset was first divided into train, validation, and test sets with 75%, 12.5%, and 12.5% of the total data assigned to each set, respectively. Then, augmentation was applied to the train set to decrease the model overfitting and increase the generalization of the model by increasing samples in the train set. Figure 3.14 shows some samples of this dataset.

To maintain consistency with the real deployment conditions of the proposed system, an augmentation strategy reflecting the deployment configuration was adopted throughout this study. In the developed device, the distance between the camera and the trap surface remains fixed during operation. Therefore, augmentation techniques were limited to transformations that preserve this imaging configuration, such as rotation and horizontal and vertical flipping. These transformations represent realistic variations in insect orientation on the trap surface while avoiding changes in scale or perspective that would not occur in the deployed system. Augmentation was applied

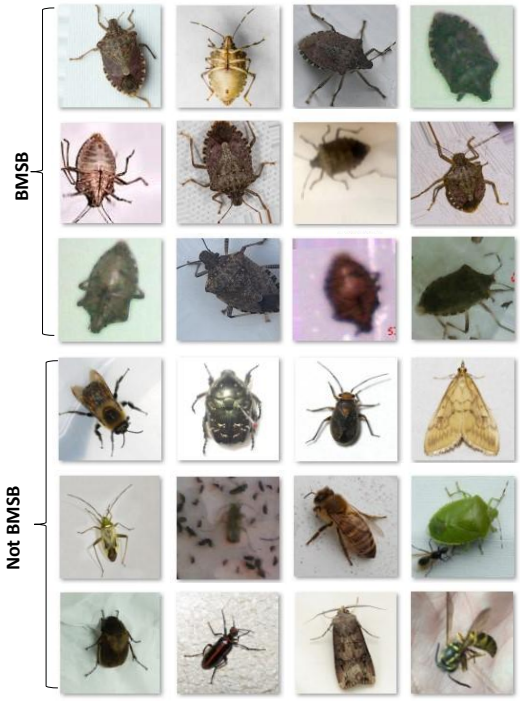


Figure 3.14. Sample images from the BMSB-Dataset, illustrating examples from the two classes used in the binary classification task.

only to the training set to improve model generalization while keeping the validation and test datasets unchanged.

3.3.5 Evaluation and discussion of the results

3.3.5.1 Accuracy & model size

The detection algorithm achieved an accuracy of 93.8% on the OpenMV platform. It was observed that most of the detection errors occurred in the areas located at the edges of the image or trap. The algorithm can detect the black objects that are surrounded by white color (background). These objects could be any object of the same size as the target insects. For example, it could be any insect species, such as a bee or a fly, or even a small leaf. When the trap image is converted to a black-and-white image (see Figure 3.15), insects located at the edges are mixed with the black area around the trap surface, thus, the algorithm is not able to recognize them as a separate object on the trap surface since they are not surrounded by white color.

Figure 3.16 illustrates the performance of the classification models in terms of accuracy, precision, recall, and model size. The accuracy of the DwSConv model was 90.2% and a 95% confidence interval (CI) of [89.3, 91.1], the precision was 93.2% (95% CI, 92.4-94.1), the recall was 90.2% (95% CI, 88.9-91.4), and the F1-score was 91.6% (95% CI, 90.8-92.4).

In comparison with the SConv model, only provided here as a reference, the DwSConv-based model had better performance and raised the accuracy by 2%. Moreover, in comparison with LeNet, the proposed model achieved better accuracy, however, it was lower than the other more complex models, including MobileNet,

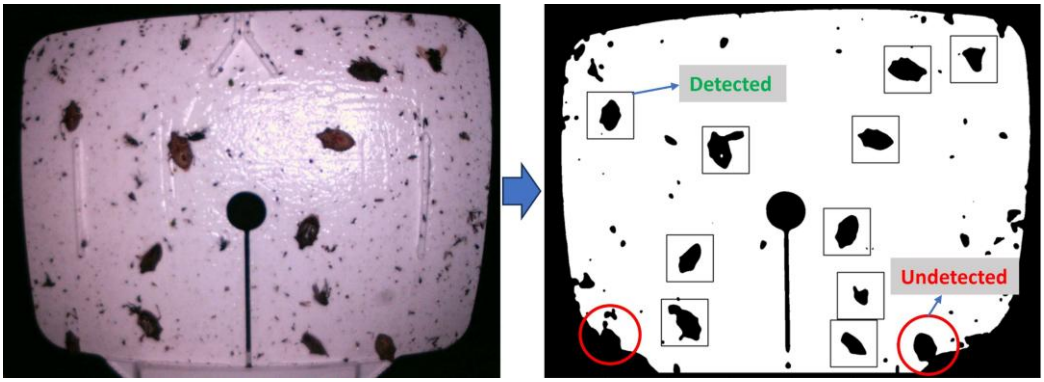


Figure 3.15. Left: Original image; Right: Black-and-white image showing detected and undetected areas.

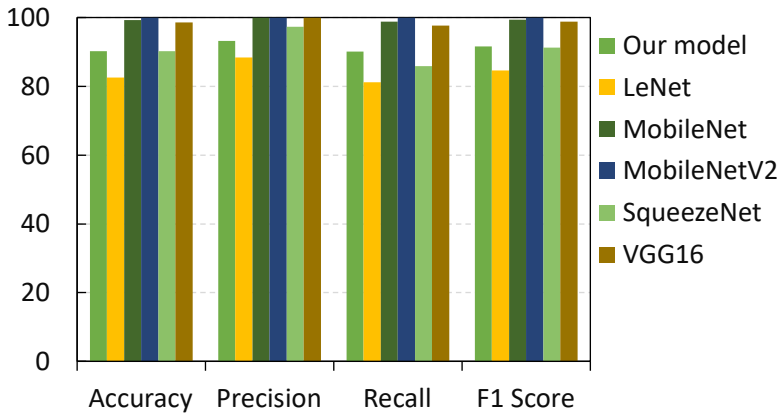


Figure 3.16. Comparison of the proposed system using different classification models based on accuracy, precision, recall, and F1-score.

MobileNetV2, and VGG16. This was an expected result because the proposed model is designed to be as light-weight as possible.

However, as shown in Table 3.3, the model had 0.046 million parameters and its size was only 80 KB which is much smaller than the other models' size, over 10 times less than SqueezeNet and approximately 30 times less when compared to other models. This is important as the amount of flash storage in typical MCUs is less than 1MB. This means that the developed model can easily fit in memory-constrained MCUs without any significant loss of accuracy, which is critical for edge-based systems.

Moreover, the proposed architecture achieved considerably lower inference time compared to other models. The proposed DwSConv model needed 0.07 seconds for each prediction. This is 3.6 times faster than LeNet, 20 times faster than MobileNet, 16.7 times faster than MobileNetV2, 27 times faster than SqueezeNet, and

Table 3.3. Performance of the proposed classification model and comparison with other models.

DL model	Model size (MB)	Parameters (M)	Inf. time(s)
Proposed model (DwSConv)	0.08	0.046	0.07
Proposed model (SConv)	0.3	0.3	0.1
LeNet	2.3	2.4	0.25
MobileNet	3.4	3.2	1.41
MobileNetV2	2.6	2.3	1.17
SqueezeNet	0.8	0.8	1.88
VGG16	14.5	14.7	48.22

approximately 688 times faster than VGG16. However, the inference time is not a critical factor for insect monitoring applications, but it has a direct impact on energy consumption. Since computational resources are used for a longer period of time, increasing inference time directly correlates with an increase in power consumption, as further explained later.

3.3.5.2 Memory usage & complexity

Memory usage is a key factor when a classification model is supposed to be implemented on a resource-constrained MCU. The peak memory usage of different models is shown in Figure 3.17 with the size of the bubbles representing the memory usage in conjunction with levels of accuracy associated with the different models as well as their complexity. The peak memory usage of the proposed model was 75 KB which was 2.7, 9.7, 14.8, 8.4, and 38 times less than LeNet, MobileNet, MobileNetV2, SqueezeNet and VGG16, respectively. Therefore, the proposed model can easily run on general-purpose MCUs with less than 512 KB of RAM. This is not possible for the other models except for LeNet.

The same figure can also reveal the relative complexity of the different approaches. The model complexity was evaluated based on the number of MACs needed for one inference. As Figure 3.17 illustrates, the proposed model had 1.3 million MACs in one inference, while this factor for other models was much greater, being 24.8 M for LeNet, 231 M for MobileNet, 146.6 M for MobileNetV2, 305 M for SqueezeNet, and over 6 billion MACs for VGG16. The impact of this factor is

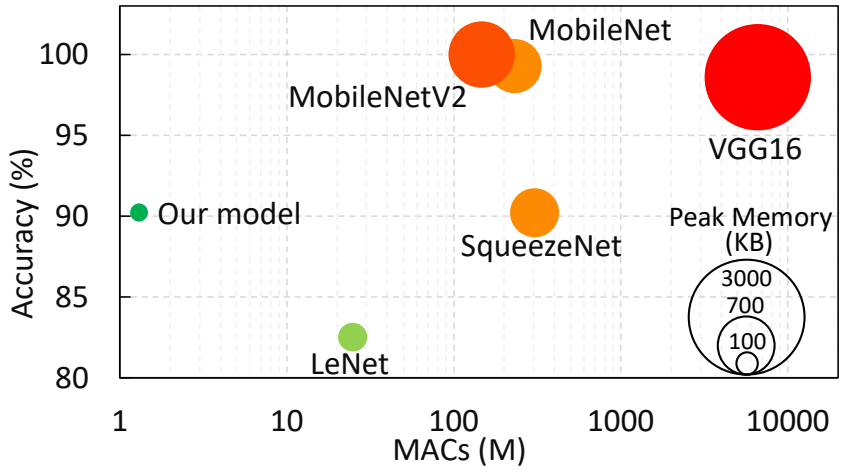


Figure 3.17. Comparison of accuracy, complexity (MACs) and peak memory usage; bubbles size represents the peak memory usage, and its color indicates the possibility of running the corresponding model on typical MCUs.

reflected in the inference time and consequently to energy consumption. The results clearly indicate that the proposed DwSconv architecture has a reasonable balance between accuracy, MACs, and peak memory usage when it is supposed to run on resource-constrained MCUs.

3.3.5.3 Power Consumption & Run time

Power consumption is another critical factor which affects both the deployment feasibility and the cost of such a system as the one described. Keeping the system in low-power sleep mode for as long as possible between image acquisition cycles will significantly reduce the power consumption, but low-power consumption algorithms are essential to enable scenarios that require more frequent sampling duty cycles.

The OpenMV current consumption over a single operational cycle is illustrated in Figure 3.18. The voltage was set to 5V. Since the camera should capture two images (from both sides of the trap), there are two peaks in the current consumption profile associated with image capture. Also, there are two peaks associated with image analysis for each of the images, for the two sides, respectively. In addition, the current consumption during the classification component reaches the highest value, thus reducing the classification time significantly reduces the total energy consumption of the system. Moreover, the system's current consumption was about 7mA in the deep sleep mode, which is almost 30 times lower compared to the active mode (see Figure 3.18).

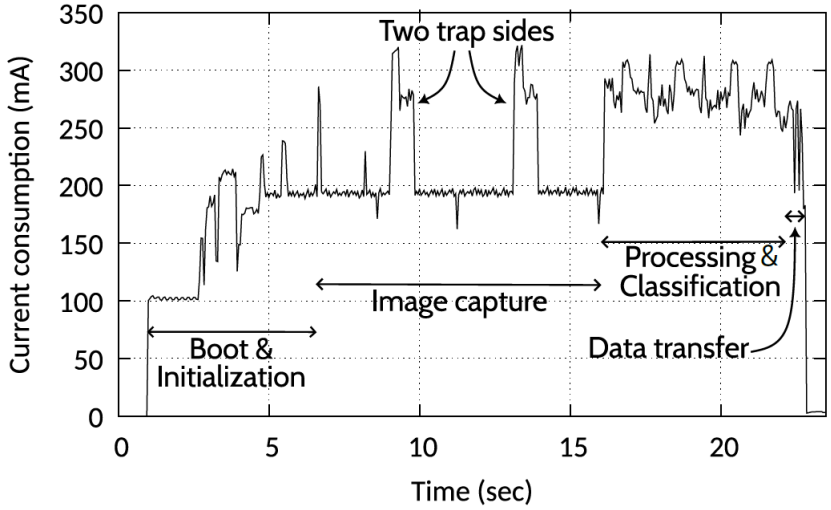


Figure 3.18. Proposed system's current consumption over different phases of a cycle.

A comparison of the energy consumption of the proposed system and another edge-based pest detection system [16] for each of the five operations is depicted in Figure 3.19. In the work reported by Albanese et al., the authors utilized a RPi for image capture, processing and classification. Another difference between their system and the one proposed here is that their system only takes and analyzes one image, while the proposed system works on a two-sided trap, thus it takes and analyzes two images in one operational cycle. Based on this fact, the proposed system consumed more energy in capturing mode as the trap needs to rotate two times during this task, taking 9.75 seconds in total. Therefore, in capturing function, the proposed system consumed 12.09 J (of which 2.7 J was consumed by the servomotor and LED) while Albanese et al.'s device consumed 1.67 J. Overall, the proposed system consumed less energy in one processing cycle, particularly in the boot and classification parts. The proposed device needed 4.63J to boot, while Albanese et al.'s device needed 40.7 J. In addition, in the classification function, the proposed system consumed just 3.24 J while Albanese et al.'s system needed 57.4 J, meaning that around 18 times more energy was expended by their system. Finally, in data transfer mode, 0.6 J and 2.15 J were consumed by the proposed system (wired serial connection) and Albanese et al.'s device which used LoRa, respectively.

As previously mentioned, the classification component's energy consumption is dependent on the network model used. It is worth noting that the classification time

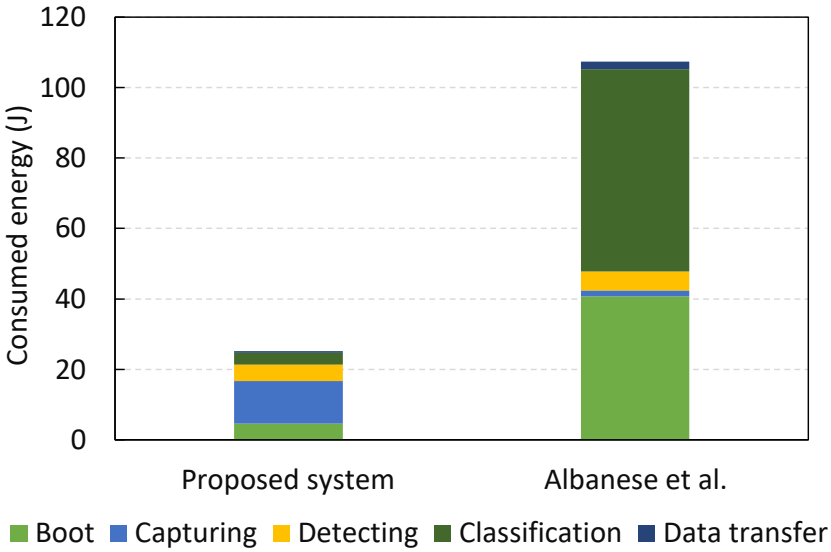


Figure 3.19. A breakdown energy consumption of the proposed system and [16].

depends on the inference time, meaning that a lower inference time leads to a lower classification time as well. In this regard, the classification time and energy consumption for different models on the OpenMV system are reported in Table 3.4. As can be observed, the system using the DwSConv model consumed only 17.9J. This is 1.3, 2.5, 2.4, 2.8, and 52 times less than LeNet, MobileNet, MobileNetV2, SqueezeNet, and VGG16, respectively. As compared to the total energy consumption, the classification component corresponds to only 18.1% of it when DwSConv was used. Remarkably, by increasing the complexity and, consequently, the inference time (see Table 3.3) the contribution of the classification in the energy consumption increases significantly for the other models. For example, when MobileNet was used, more than 50% of the energy consumption was dedicated to classification. As a result, the classification model plays a key role in such a system's energy consumption.

3.3.5.4 System Lifetime

System lifetime is a critical factor in insect monitoring systems, since they should last for at least a growing season when they are deployed in orchards. In this version of the device, two packs of four series-connected batteries (AA NiMH rechargeable battery, 2.45Ah, 1.2V) are connected in parallel so the battery bank voltage and capacity are 4.8V and 4.9Ah, respectively.

To calculate the battery lifetime, the energy consumption of the device in one day should be computed. As mentioned earlier, the system goes into active mode twice a day, and it remains in sleep mode for the rest of the day. In each of the active mode periods, it is assumed that the system works for 30 seconds during which it consumes an average of 300 mA (see Figure 3.18). Moreover, the system consumes 7 mA in

Table 3.4. Energy consumption for different models running on OpenMV.

DL models	Total runtime [b] (s)	Total energy (J)	CL time [a] (s)	CL energy [a] (J)
Our model (DwSConv)	16.37	17.9	2.52	3.24
LeNet	18.34	24	4.95	6.26
MobileNet	31.5	44.4	18.5	26.8
MobileNetV2	30.3	42.4	17.2	24.6
SqueezeNet	36.2	50.6	22.5	34.5
VGG16	618.5	932	605.4	914.1

[a] Classification (CL) time/energy consumption for 12 images.
[b] The booting is not considered in this table.

sleep mode. Hence, every 24 hours, the system consumes 300 mA for 60 seconds and 7 mA for the remainder of the day. According to these values, the total energy consumption for one day is about 0.83 Wh, and as the battery capacity is 23.52 Wh (4.9Ah x 4.8V), the system could continuously work for over 28 days without any charging activity.

In addition, the system benefits from a solar panel to charge the battery bank. The typical charging voltage of the solar panel is 5.5V, the current is 100 mA, and the conversion rate is up to 17%. Thus, based on these values and considering that 0.83 Wh of energy is consumed per day, the charging time needed to restore the battery to its full capacity is around 8.9 hours. However, this is a theoretical estimation, and it is expected to be higher in cloudy weather conditions. Based on the fact that the day is much longer than the estimated charging time, the battery bank is capable of being fully charged within a sunny day, meaning that the system can work on the battery for a long time. It is worth mentioning that by using bigger solar panels, the charging time can be decreased. For example, the charging time can be decreased to 4.5 hours by using two solar panels (the one used in this study) in parallel.

3.4 Accuracy evaluation on real-world orchard data

As mentioned, the device was deployed in an Italian orchard infested with *Halyomorpha halys* in 2022 (a pear orchard in the Emilia Romagna region in northern Italy). Images were collected and used to create a dataset of HHs. This version of the dataset was created using images captured from the trap surface at night. Here, the proposed detection and classification algorithm is applied to the collected data to evaluate its performance under field conditions.

The analysis of real-world data revealed that the detection phase was unable to properly detect insects. This was due to the fact that the algorithm could not properly convert RGB images into black-and-white where the foreground should be separated from the background, as shown in Figure 3.20 (left). This indicates that the Otsu's method failed to find the optimal threshold. To address this issue, a contrast limited adaptive histogram equalization (CLAHE) technique [17] was used to improve local contrast, ensuring that the histogram is more uniform across different areas of the image. This helps the Otsu's method select the optimal threshold to effectively separate foreground and background, as shown in Figure 3.20 (right). Consequently,

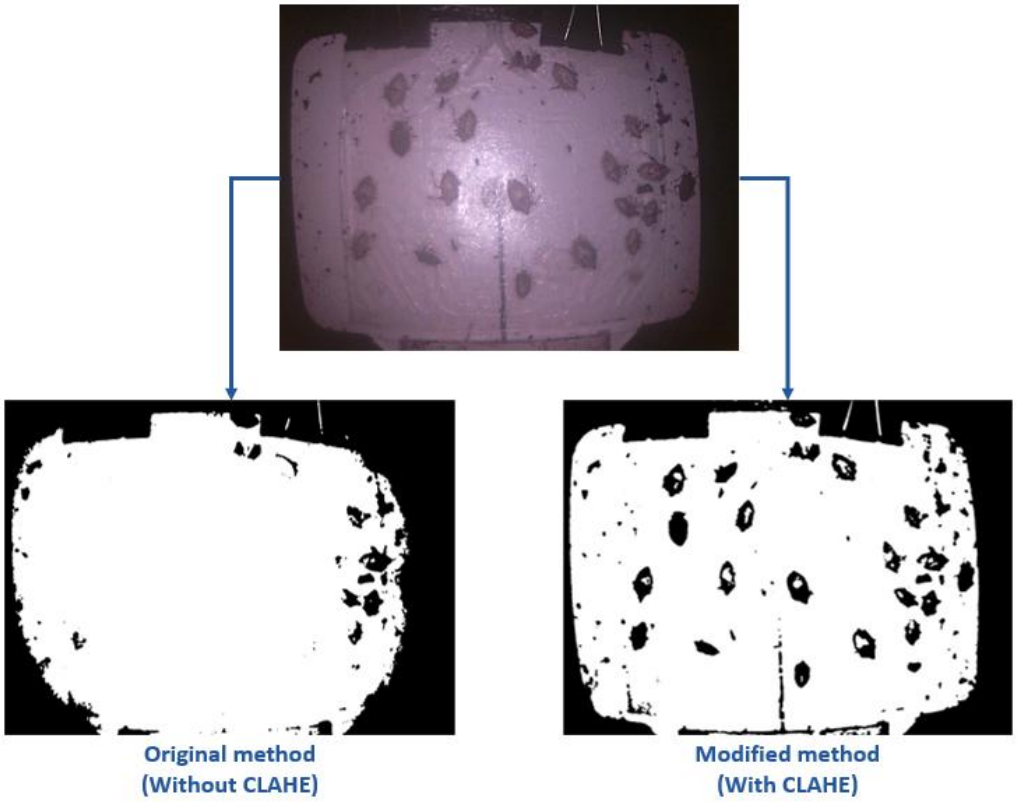


Figure 3.20. The impact of CLAHE technique on converting RGB images to black-and-white ones.

the detection phase was modified and enhanced by adding the histogram equalization technique at the beginning of this phase.

This highlights how important it is to incorporate a wide range of real-world images into algorithm development and help the algorithm adapt to the complexities of real-world situations.

In Figure 3.21, the modified image processing pipeline is illustrated. In the detection phase, the process begins with the application of the CLAHE histogram equalization to the raw captured image (step 1). Then, the image is converted to a gray-scale image, and a smoothing filter is applied to eliminate small objects (step 2). Then, the image is converted to black-and-white using Otsu's method (step 3), and finally, the suspected area that might be filled by the target insect is detected using a blob detection algorithm (step 4). In the next stage, these suspected areas are cropped from the captured image (step 5) and fed to the classification phase, where a lightweight CNN-based image classification model is used to classify the images (step 6). Finally, the results are sent to the growers (step 7).

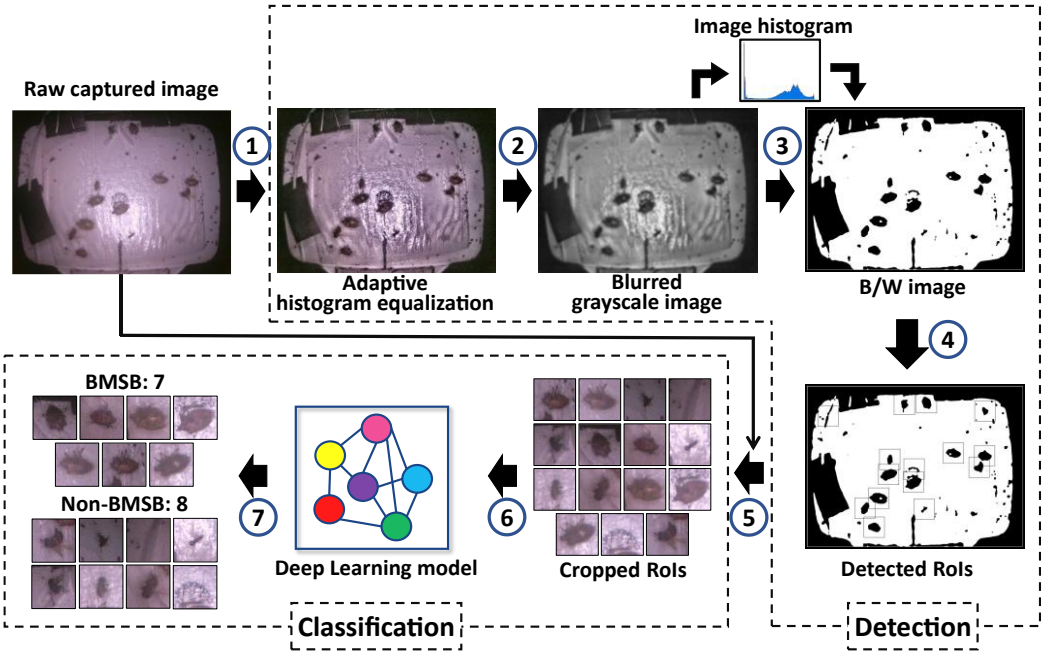


Figure 3.21. Modified image processing algorithm, from the raw captured image to classified RoIs executed on the MCU-based device.

Table 3.5 reports the accuracy of the modified method based on images from the real-world orchard as compared to the original method and YOLO methods that are widely used in insect detection systems.

The modified algorithm compared to the original one decreased MSE from 50.8 to 13.6 and MAE from 5.1 to 2.1, and increased Recall around 34%, indicating a reduction by a factor of approximately 3.7 in MSE and approximately 2.5 in MAE. In comparison with YOLO methods, although the accuracy of YOLO methods is better

Table 3.5. Comparison of different models on the data collected from the real-world orchard for BMSBs counting.

Model	MAE	MSE	P (%)	R (%)	MCU Comp.
YOLOv8n	0.71	1.33	86.6	98.5	No
YOLOv5n	0.88	0.55	85.1	97.6	No
YOLOv3n	0.5	0.40	88.9	98.8	No
Proposed-Original	5.07	50.79	88.5	36.7	Yes
Proposed-Modified	2.07	13.55	84.0	70.6	Yes
Proposed-Modified (7-day)	1.05	1.81	83.0	80.2	Yes

than the proposed one, they are computationally intensive, large in size, and unable to run on resource-constrained MCUs.

In terms of running time, the added step in the detection phase of the image processing algorithm takes up to 0.5 seconds per image. Therefore, the total runtime for a complete system operation, including image capturing (from two sides of the trap), detection (on two captured images), classification, and results, is approximately 17.5 seconds.

Moreover, the analysis revealed that the presence of a high level of overlap among the captured insects in the obtained images, which occur when new insects are trapped physically stuck next to or over the existing ones trapped previously, increases the error of the detection phase. This is due to the fact that the detected blobs are filtered out if their size is much larger than that assigned to the presence of a BMSB (overlapping increases the blob size).

To evaluate the impact of insects overlapping each other, the data set was altered by removing the images that exceeded seven days between trap replacements. This led to a significant reduction in error by decreasing the MAE from 2.07 to 1.05 and MSE from 13.55 to 1.81. It also increased the recall by almost 10%. These results prove the negative impact of overlapping on the method performance. Therefore, it is important to replace the trap in a timely manner to prevent excessive insect overlap, which can negatively impact detection and counting performance.

3.5 Conclusion

This chapter completes the first part of the first objective of this thesis by proposing a low-cost and low-power monitoring system for the detection of *Halyomorpha halys*, an invasive insect which causes a lot of damage to fruit. The system employs light but efficient image processing and a DL mechanism that can run on very low-cost microcontrollers.

A series of experiments were run on a blend of real and artificial datasets. The experimental results confirmed that it is feasible to run complex deep learning approaches on MCU-based systems on the edge. Compared to other approaches in the literature, the comparison showed significant gains in terms of energy consumption and model size, which greatly affect the cost of the deployment. The compromise in

terms of detection accuracy was only a few percentile units away from models much larger in-size.

To achieve low memory usage, a separate detection and classification approach was used in the suggested algorithm. Thanks to this approach, the system could load images from the memory several times (i.e., load captured images from both sides and run a detection algorithm to detect, crop and save potential areas; then again load cropped images for classification). The process of loading images from memory increases the image processing time as well as power consumption.

Based on the data collected and insights from real-world deployment, it became evident that a more complex image processing algorithm was needed. Therefore, in the next chapter, a single-stage DL-based algorithm is proposed that does the detection and classification simultaneously in order to decrease image loading time, and thus, energy consumption and improve accuracy.

3.6 References

[1] M. Preti, F. Verheggen, and S. Angeli, “Insect pest monitoring with camera-equipped traps: strengths and limitations,” J Pest Sci (2004), vol. 94, no. 2, pp. 203–217, Mar. 2021, doi: 10.1007/S10340-020-01309-4/TABLES/2.

[2] S. K. Kopparapu, “Lighting design for machine vision application,” Image Vis Comput, vol. 24, no. 7, pp. 720–726, Jul. 2006, doi: 10.1016/J.IMAVIS.2005.12.016.

[3] A. Novini, “Fundamentals of machine vision lighting,” Proceedings of NORTHCON 1993 Electrical and Electronics Convention, Conference Record, pp. 38–46, 1993, doi: 10.1109/NORTHC.1993.505029.

[4] A. Goel, C. Tung, Y. H. Lu, and G. K. Thiruvathukal, “A Survey of Methods for Low-Power Deep Learning and Computer Vision,” IEEE World Forum on Internet of Things, WF-IoT 2020 - Symposium Proceedings, Jun. 2020, doi: 10.1109/WF-IOT48130.2020.9221198.

[5] R. C. Gonzalez and R. E. Woods, Digital Image Processing (3rd Edition). Pearson, 2007. Accessed: Sep. 10, 2025. [Online]. Available: http://dl.acm.org/citation.cfm?id=1076432

[6] N. Otsu and others, “A threshold selection method from gray-level histograms,” Automatica, vol. 11, no. 285–296, pp. 23–27, 1975.

[7] “MicroPython libraries — MicroPython 1.25 documentation.” Accessed: Oct. 10, 2025. [Online]. Available: https://docs.openmv.io/library/index.html

[8] A. L. Acebes, “Host plant effects on the biology, behavior and ecology of brown marmorated stink bug, Halyomorpha halys (Stål)(Hemiptera: Pentatomidae),” Virginia Tech, 2016.

[9] L. J. Zurek, A. Kargar, B. O’Flynn, D. Niederprüm, L. Wolf, and D. Zorbas, “Evaluation of Wireless Technologies for an Embedded Camera-based Pest Monitoring System,” 2023 IEEE International Workshop on Metrology for Agriculture and Forestry, MetroAgriFor 2023 - Proceedings, pp. 449–454, 2023, doi: 10.1109/METROAGRIFOR58484.2023.10424396.

[10] A. G. Howard et al., “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” Apr. 2017, Accessed: Jul. 25, 2025. [Online]. Available: https://arxiv.org/pdf/1704.04861

[11] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2016-December, pp. 770–778, Dec. 2016, doi: 10.1109/CVPR.2016.90.

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2323, 1998, doi: 10.1109/5.726791.

[13] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 4510–4520, Jan. 2018, doi: 10.1109/CVPR.2018.00474.

[14] 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 <0.5MB model

size," Feb. 2016, Accessed: Aug. 06, 2025. [Online]. Available: https://arxiv.org/pdf/1602.07360

[15] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings, Sep. 2014, Accessed: Jul. 25, 2025. [Online]. Available: https://arxiv.org/pdf/1409.1556

[16] A. Albanese, M. Nardello, and D. Brunelli, "Automated Pest Detection with DNN on the Edge for Precision Agriculture," IEEE J Emerg Sel Top Circuits Syst, vol. 11, no. 3, pp. 458–467, Sep. 2021, doi: 10.1109/JETCAS.2021.3101740.

[17] K. Zuiderveld, "Contrast limited adaptive histogram equalization," in Graphics Gems IV, USA: Academic Press Professional, Inc., 1994, pp. 474–485.

4 CHAPTER IV. SEMIY-NET: A TINY DEEP LEARNING MODEL FOR ON-DEVICE INSECT SEGMENTATION AND COUNTING

The work in this chapter is adapted from the first author peer-reviewed articles as follows: Section 4.2 is primarily derived from article 1, and Section 4.3 from article 2.

1. **Kargar, A.**, Zorbas, D., Gaffney, M., O'Flynn, B., & Tedesco, S. (2024), "Y-Net: Insect Counting and Segmentation using Deep Learning on Embedded Devices," In *2024 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)* (pp. 1-6). IEEE, DOI: 10.1109/I2MTC60896.2024.10561007.
2. **Kargar, A.**, Zorbas, D., Gaffney, M., O'Flynn, B., & Tedesco, S. (2025), "Tiny deep learning model for insect segmentation and counting on resource-constrained devices," *Computers and Electronics in Agriculture*, 236, 110378, DOI: 10.1016/j.compag.2025.110378.

Dataset:

3. **Kargar, A.**, Zorbas, D., Gaffney, M., O'Flynn, B., & Tedesco, S. (2024). Image Dataset of Brown Marmorated Stink Bug (BMSB) or Halyomorpha Halys (HH)-V2 [Data set]. Zenodo, DOI: 10.5281/zenodo.14051319.

4.1 Introduction

Building on the prototype device developed and described in Chapter 3, this chapter continues with the second objective of this thesis by the development of a tinyDL-based model with the ability to count and segment HH from captured images on an IoT edge-based device. As mentioned in the previous chapter, the device was designed around an MCU, meaning that all the counting and segmentation computation should be performed on a resource-constrained device. In this chapter, the target size in terms of RAM and flash memory storage is 1MB and 2MB, respectively, which corresponds to the capability of a typical modern MCU device.

An image dataset for the target insect, *Halyomorpha halys*, was first created using images captured by the introduced device deployed in a fruit orchard.

Then, a Y-Net model inspired by U-Net [1] was developed with the capability of insect counting in addition to segmentation. In this design, a lightweight MobileNet-based encoder was used to reduce computational and memory requirements, making it suitable as a baseline for edge-based deployment. The performance of this model was assessed using a variety of different metrics, and the results demonstrated the feasibility and effectiveness of the model in counting and segmentation of insects using Edge-AI algorithms capable of running on embedded devices such as smartphones or Raspberry Pi boards.

Since the goal for this thesis is to perform the counting and segmentation on MCUs, SemiY-Net was then proposed, which is suitable for battery-powered resource-constrained MCU-based edge devices. The previously developed Y-Net model is considered as a lightweight baseline for this task, and the design of SemiY-Net follows a hardware-aware approach where this baseline is further optimized to meet strict MCU constraints. However, not all optimization techniques are equally effective in this context, therefore, a combination of methods was employed, carefully selected based on the capabilities of the target hardware and their impact on model accuracy. For this aim, several strategies were used to minimize model size and memory usage, such as structural pruning, skip connection management, and architecture simplification, to meet the resource constraints of an MCU. The proposed model was investigated from different aspects to evaluate its performance and the feasibility of its implementation on MCU-based edge devices.

4.2 Y-Net: A model for segmentation and counting

4.2.1 Y-Net architecture

Image segmentation is widely used to distinguish object regions in an image. One of the most popular image segmentation architectures is U-Net [1] which is widely used for this purpose, as shown in Figure 4.1. This model has two main components, an encoder and a decoder, which are also known as the contracting and expansive paths respectively. The encoder part consists of convolutional layers that are widely used to extract important features from the input and decrease the

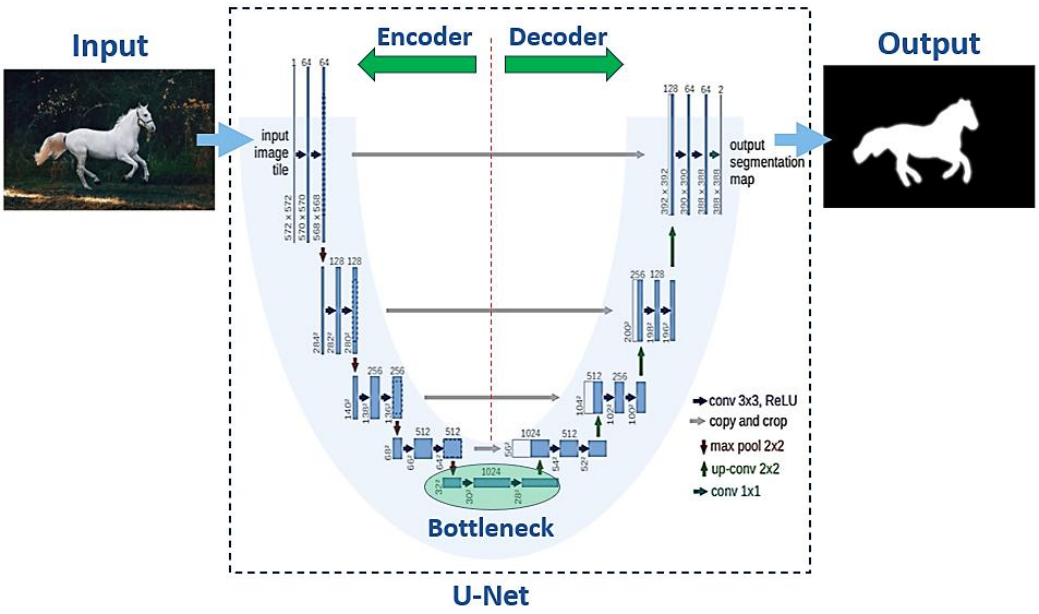


Figure 4.1. U-Net architecture for image segmentation, a widely used DL model for image segmentation.

dimensions of the input image. The decoder component is responsible for creating a segmentation map using upsampling layers and increasing the dimensions. The bottleneck is a central point where these two parts are connected to each other. It also has four skip connections that copy information from the encoder to corresponding layers of the decoder, where the copied information concatenates to the current data [1].

In a binary U-net, the segmentation output generates a grayscale image with the same dimensions as the input image, in which the target objects (e.g., HHs in this case) appear as lighter pixels, while the background is represented by black pixels. However, for insect monitoring, the number of insects on the image is also required in addition to segmented areas. To count the objects from the segmented binary image generated by U-Net, methods such as Connected-component labeling (CCL) [2] can be used to count the connected pixels in the binary image. However, such approaches impose extra stages on the system algorithm that can increase processing time and power consumption; also, using CCL may be inaccurate in counting insects in cases of overlap among insects, which is frequent in this application and is evident in the collected dataset, since overlapping insects are considered as a single connected component and counted as one object. To tackle these limitations, a new architecture named Y-Net, inspired by U-Net, was proposed to count the target insects and segment them on the input image in one stage. This architecture is depicted in Figure 4.2. As

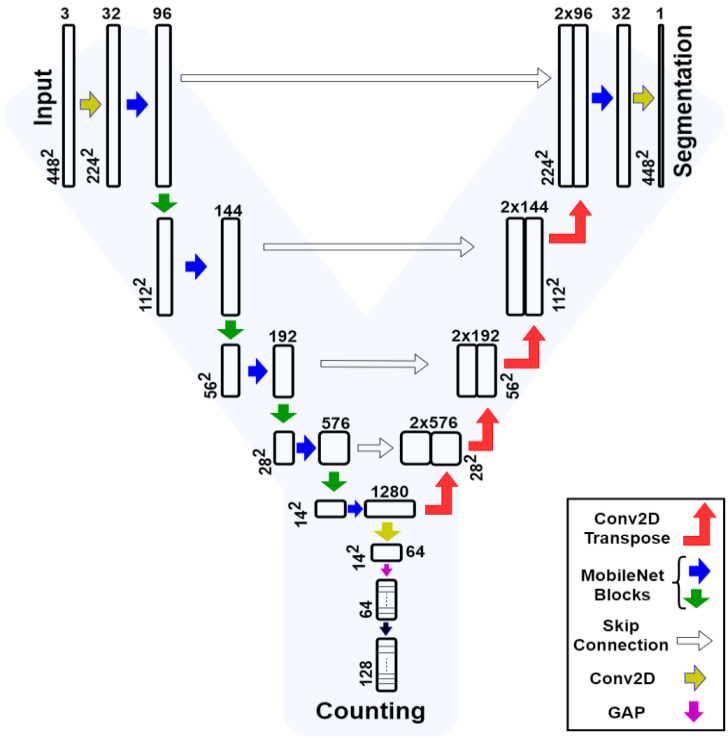


Figure 4.2. The proposed Y-Net architecture with two outputs: counting and segmentation, where MobileNetv2 with α=1 is used in the encoder part.

shown in this figure, an auxiliary branch was taken from the bottleneck of the U-Net model for the counting output. As a result, following the encoder part, the model presents two branches, one used for insect counting while the other one is used for the segmentation task. This network was named Y-Net because of its Y-shaped structure.

Indeed, after the encoder path, which is responsible for extracting important features from the input image, the model branches into two paths. The first path uses the extracted features to create the segmentation output, while the second path uses the features to generate the counting output. This allows each path to separately use the extracted features and analyse them for their specific purpose in their own way. Other parts of the model were investigated for the counting part; for instance, if the model performs the counting on the returned segmentation output, the counting part would not have direct access to the extracted features' information (i.e., provided by the encoder part) and must only rely on the one-channel segmentation output. This could also increase the risk of error propagation, where inaccurate segmentation results negatively affect counting by carrying forward the error from segmentation to the counting part. Analysis indicated that in this scenario the model could not train well, especially for the segmentation output, and could not achieve a balance between

the segmentation and counting outputs since both are part of the same branch and their interactions affect learning during back-propagation.

In this model, the lightweight but powerful MobilenetV2 [3] architecture was used for the encoder component to efficiently extract features. MobileNetV2 was used due to its low complexity and lightweight structure, which makes it well-suited for embedded devices with restricted computational resources. To this aim, the pre-trained MobilenetV2 available in Keras (e.g., trained on the ImageNet dataset) was used.

The decoder consists of several transposed convolution or deconvolution layers to upsample the feature maps and reconstruct high-resolution spatial information. Like U-Net, this Y-Net architecture benefits from skip connections that copy and concatenate encoder layers' information to the corresponding decoder layers.

In the counting part, as illustrated in Figure 4.2, the extracted features are passed through a convolution layer and then a GAP layer to reduce the dimensions of the extracted 2D feature maps and convert them into a 1D feature vector. This vector is then fed to a fully connected layer (with a ReLU activation function) which is connected to the counting output. The counting output layer is a single neuron without an activation function, since this is a regression task.

It should be noted that using GAP instead of a fully connected layer significantly reduces the parameter numbers, leading to a significant reduction in model size and computation; this is critical when the model is required to be run on embedded devices. Besides, this layer helps the network avoid overfitting issues since it averages all elements of each feature map while the fully connected layer flattens all elements of each feature map [4].

4.2.2 Datasets

The most important first step in developing DL models for insect monitoring systems is to have an appropriate dataset of the target insect. In this study, a dataset for HHs was created using images captured by the device described in the previous chapter. This device was deployed in a pear orchard in the Emilia Romagna region in northern Italy, which was infested with *Halyomorpha halys*.

In this version of the dataset, an image dataset of *Halyomorpha halys* insects captured from two-sided sticky traps was created. The dataset supports object

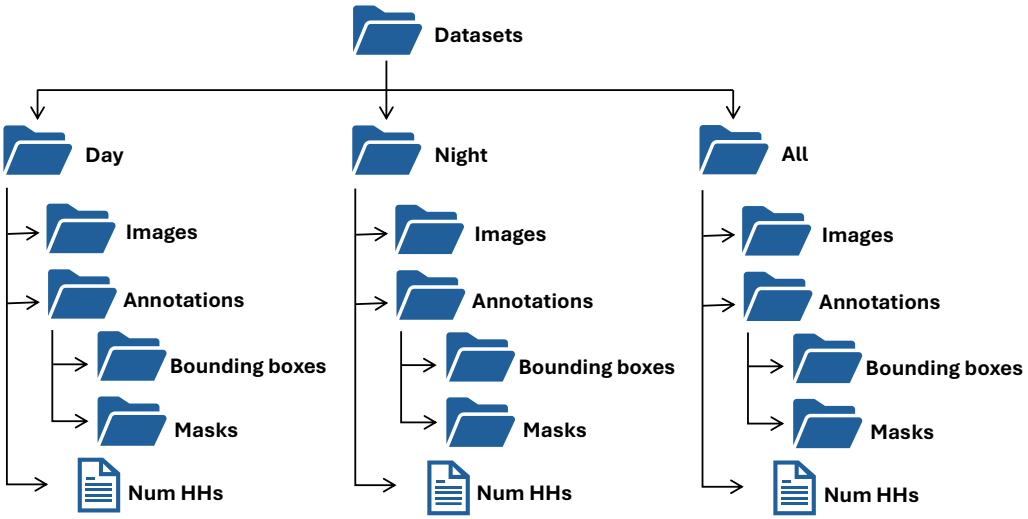


Figure 4.3. Dataset structure showing Day, Night, and All subsets with images, annotations (bounding boxes and masks), and the number of HHs instances.

detection and segmentation tasks by providing two different types of annotation. The dataset structure is shown in Figure 4.3. As is clear from the figure, it consists of three main sub-datasets named 'Day', 'Night', and 'All', which respectively contain images captured at the night, daytime, and combination of night and daytime. The images have a resolution of 1600×1200 pixels and were captured using the IoT device developed and described in the previous chapter, deployed in an orchard in Italy infested with the target insect during the growing season of 2023. The 'Day' dataset has 256 images with a total number of 2,222 HHs, and the 'Night' one has 267 images with 2,452 HHs. Thus, the 'All' dataset has 523 images with 4,674 HHs in total. Each sub-dataset includes three folders as follows:

- **Images**: This contains the original images in JPEG format from the sticky trap's surface that might include the HH as well as other insects and objects such as windblown dust or tiny leaves.
- **Annotations**: This folder contains the annotations related to HH in the images. These annotations are provided in two different formats:
    - Masks, which are binary images, highlighted the HH in the original images. Pixels belonging to HH insects were set to white, while the rest were set to black.
    - Bounding Boxes, which are JSON/XML files indicating the HH locations in images.

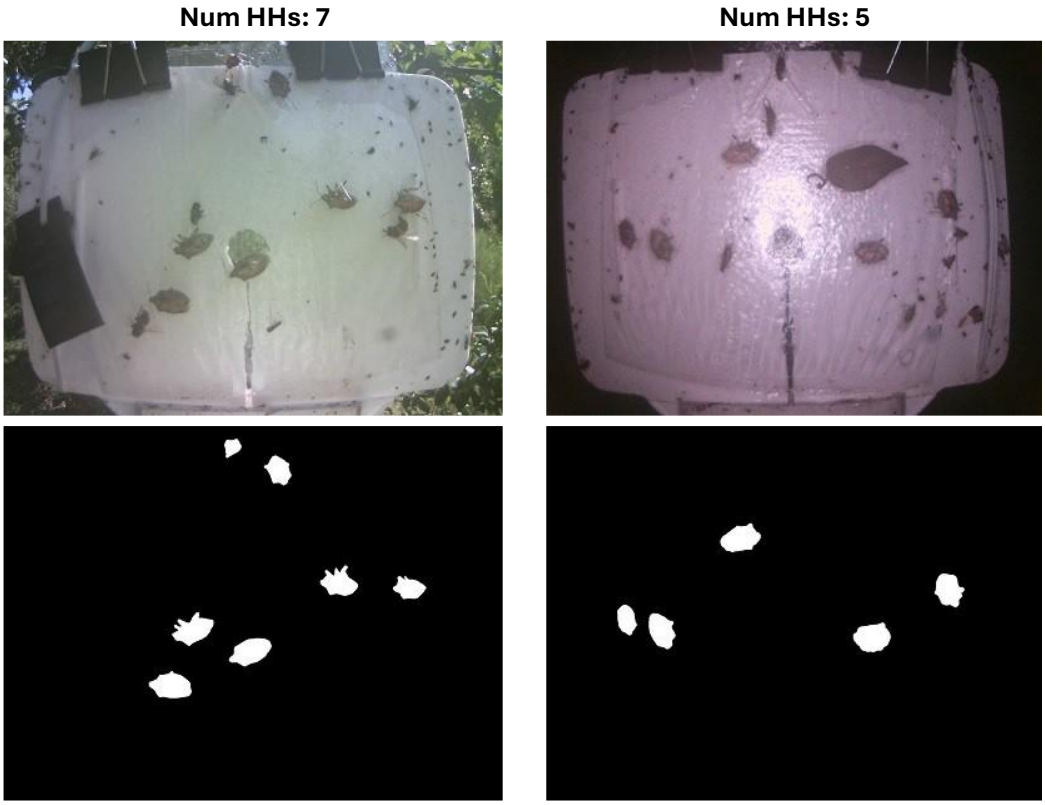


Figure 4.4. Two samples from 'Night' and 'Day' datasets; the first row is the captured images and the second row is the corresponding mask.

- **Num HHs**: This folder contains a CSV file that indicates the number of HH present in each image.

Figure 4.4 shows two samples of images from 'Day' and 'Night' datasets, along with their corresponding masks and the number of HH on the images.

4.2.3 Datasets splitting strategy

For this study, the dataset is split into training, validation, and test sets to train and evaluate DL models. As mentioned, the images were captured from two-sided traps. Figure 4.5 shows the distribution of HH numbers for each side of the Night dataset. It is clear that the distributions of these two sides are different in the dataset.

Thus, it is important to preserve this distribution during the splitting of the dataset. Note that, in these images, the number of HHs were binned with a bins size of four.

Figure 4.6a shows each side distribution when the dataset was randomly split into training, validation, and test sets. It should be noted that in these charts, the orange bars are overlaid on the blue bars; the blue bars show the proportion of HHs in the original dataset that falls into each bin, while the orange ones show the proportion of HHs in each set that falls in the bins. Therefore, comparing the overlay of the orange and blue bars indicates how accurately each set reflects the original dataset; the closer the bars are to each other, the better the set represents the original dataset.

Based on Figure 4.6a, there is no guarantee to preserve the distribution of each side if the images are randomly split. To overcome this issue, stratified sampling based on the HH numbers was used. For this aim, the data was first divided into side1 and side2. Then, stratified sampling was applied to each side separately to split the data into training, validation, and test sets. Finally, the training sets of each side (training_side1 and training_side2) were combined to create the final training set. The same was carried out for the validation and test sets. Figure 4.6b shows the strategy used for dataset splitting; it is evident that this strategy could reflect the original dataset distribution to each set better than the random splitting. The same strategy was also used for the 'Day' and 'All' datasets.

In addition to the provided visual inspection, the proposed splitting strategy was statistically analysed and the Chi-Square Goodness of Fit test was selected for this purpose. This is a suitable test for categorical data to check if the frequency of the

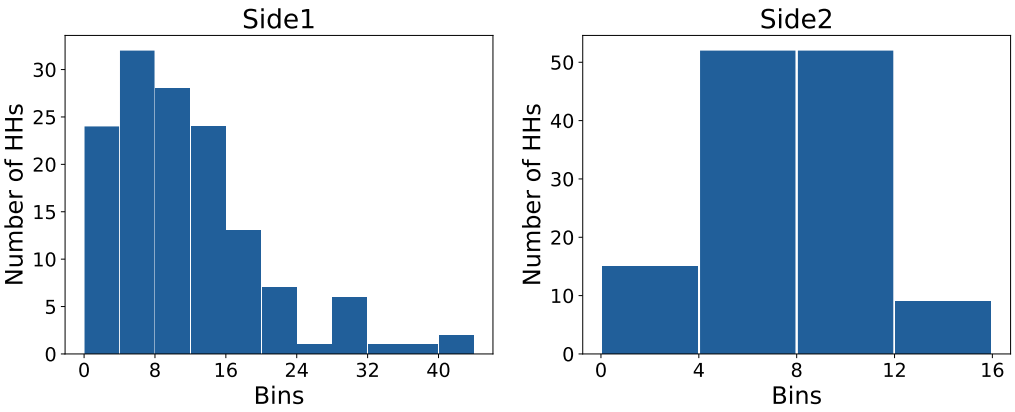


Figure 4.5. Distribution of the target insect numbers for each side of the Night dataset; the number of HHs were binned with the bins size of four.

observed data matches the frequency of expected data, where a higher p-value shows a higher degree of matches. For this test, the original dataset was considered as expected data, and each set was considered as observed data; moreover, each bin was considered as a category.

The P-value of this test for random splitting and the proposed strategy is reported in Figure 4.6. Based on this figure, it is evident that the proposed strategy has a higher p-value for all sets compared to random splitting, indicating a stronger match.

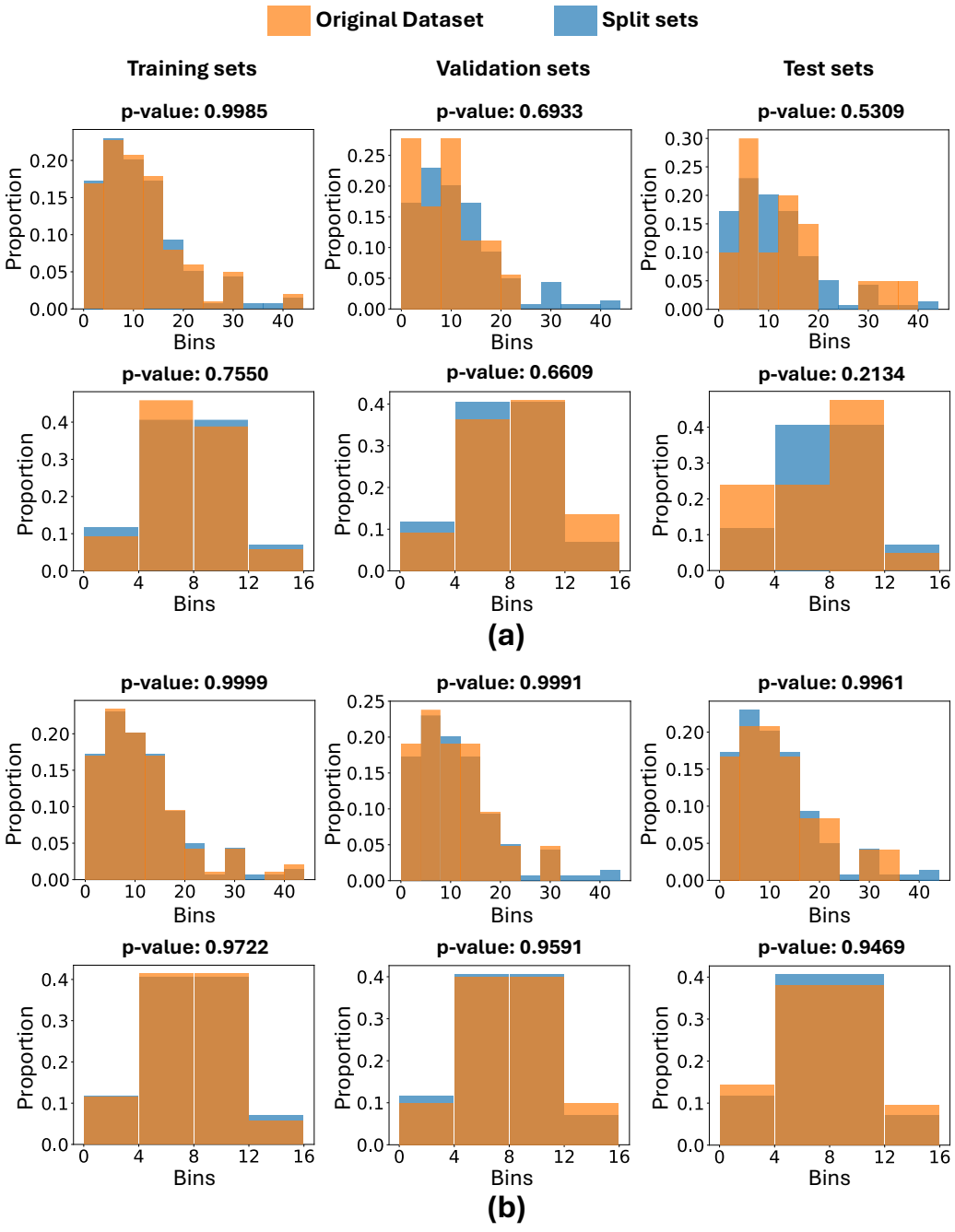


Figure 4.6. Distribution of the target insect numbers on each side for training, validation and test sets with the corresponding p-value of chi-square goodness of fit test; a: random splitting, b: stratified-based splitting.

4.2.4 Hyperparameters and training details

The proposed Y-Net model was implemented and evaluated using TensorFlow on Google Colaboratory. For training the model, Binary Crossentropy was selected as the loss function for the segmentation task since it involves two classes (HHs and background), and MSE was used for the counting task. Moreover, Adam optimizer was utilized for model training for 100 epochs with a learning rate of 0.0001 and a batch size of 2. The MobileNetV2 backbone pre-trained on the ImageNet dataset was used as the feature extractor, and all of its layers were frozen during training.

Furthermore, the performance of the proposed Y-Net was investigated for two different input sizes including 448×448 and 320×320 to compare the input size impact on the segmentation, counting and efficiency of the model. Moreover, the performance of the model was assessed by changing the α parameter of MobileNetv2; α is a parameter defined to control the model width by scaling the channel number of each layer, which impacts the model complexity and accuracy. In this study, four different α values (0.35, 0.5, 0.75, and 1.0) were evaluated. The inference time of the model was measured by running the model on a Samsung A54 Android phone. It was measured using the TensorFlow Lite Benchmark Tool, which executes the model multiple times and reports the average latency on the device.

Finally, several augmentation techniques were employed on the training dataset to enhance the model generalization and decrease the risk of overfitting issues. The augmentation techniques used were vertical and horizontal flip, rotation, and random brightness and contrast.

4.2.5 Evaluation and discussion of the results

Table 4.1 reports the values of these metrics for different configurations of the proposed Y-Net. It is evident that the model with the alpha value of 0.75 and input size of 320×320 had the lowest value of MSE and MAE at around 1.86 and 0.98, respectively. Based on the results, it was observed that the counting was affected when there was a multitude of HHs on the trap, specifically when the insects were laid over each other, since it increased the complexity of extracting features of distinct insects for the model.

Based on Table 4.1, the models with the input size of 448×448 px (α=1) and 320×320 px (α=1) have the best performance in terms of insect segmentation at approximately 85% for IoU and 91% for DSC. Additionally, these two models obtained a recall and a precision of about 92% indicating that the model is able to correctly predict positive cases which are actual positive and also minimises false positives and negatives. However, the model with an input size of 448×448 px requires 0.94 seconds to analyse one image, while the model with an input of 320×320 px needs 0.48 seconds.

As expected, by decreasing α, the model performance decreases for both counting and segmentation, but it also results in a significant reduction in the number of model parameters and inference time. For example, by using a model with an alpha value of 0.35, IoU and DSC are respectively approximately 4% and 2% lower than a model with α=1, but it has a model parameter and inference time of nearly three times less. There is therefore a trade-off between model efficiency and accuracy which required to be carefully balanced for practical applications of automated insect monitoring on embedded devices.

Moreover, the number of parameters is independent of the input size, and this is attributed to the use of a GAP layer instead of a fully connected layer that made the model size and parameters unaffected by the input size. However, the input size has a direct impact on the model inference time. As mentioned, a model with an input size of 448×448 px needs approximately twice as much time as a model with an input size of 320×320 px for each prediction. Thus, a larger input size leads to higher execution time since a larger image needs more computations to be analysed by the model.

Table 4.1. Y-Net Performance on the test set for two different input sized and four different alpha values of Mobilenetv2 with Mean value and 95% Confidence Interval (CI) of segmentation and counting metrics.
IoU: Intersection over Union , DSC: Dice Similarity Coefficient, MSE: Mean Squared Error, MAE: Mean Absolute Error.

Model		**Segmentation metrics**				**Counting metrics**		**Efficiency metrics**	
Input Size	**α**	**IoU**	**DSC**	**Precision**	**Recall**	**MSE**	**MAE**	**Model parameters**	**Inference time (s)**
448×448	1	84.60 (84.39, 84.82)	91.56 (91.42, 91.70)	92.39 (91.97, 92.81)	91.65 (91.18, 92.12)	3.12 (2.64, 3.60)	1.34 (1.23, 1.46)	12,446,226	0.94
	0.75	84.01 (83.71, 84.30)	91.22 (91.03, 91.40)	91.90 (91.07, 92.72)	92.02 (90.86, 92.82)	2.38 (1.98, 2.78)	1.11 (1.00, 1.22)	8,897,226	0.75
	0.5	82.38 (82.07, 82.68)	90.18 (89.97, 90.39)	92.79 (92.42, 93.16)	89.21 (88.69, 89.74)	3.13 (2.59, 3.76)	1.20 (1.07, 1.36)	5,535,154	0.34
	0.35	81.21 (80.86, 81.56)	89.51 (89.27, 89.75)	93.97 (93.38, 94.47)	87.43 (86.41, 88.45)	2.58 (2.36, 2.88)	1.18 (1.09, 1.28)	3,842,194	0.31
320×320	1	84.96 (84.79, 85.10)	91.78 (91.68, 91.87)	91.90 (91.48, 92.31)	93.01 (92.68, 93.35)	2.89 (2.31, 3.47)	1.26 (1.15, 1.38)	12,446,226	0.48
	0.75	84.33 (84.14, 84.52)	91.43 (91.32, 91.54)	92.02 (91.37, 92.66)	92.77 (92.12, 93.43)	1.86 (1.65, 2.18)	0.98 (0.91, 1.08)	8,897,226	0.39
	0.5	82.73 (82.61, 82.84)	90.45 (90.37, 90.52)	92.34 (91.81, 92.87)	90.32 (89.61, 91.03)	3.14 (2.82, 3.45)	1.28 (1.20, 1.37)	5,535,154	0.18
	0.35	80.77 (80.56, 80.97)	89.22 (89.09, 89.35)	93.42 (93.16, 93.69)	87.91 (87.38, 88.45)	2.73 (2.41, 3.05)	1.22 (1.10, 1.34)	3,842,194	0.14

Finally, Figure 4.7 depicts how the model (input size of 448x448 px with α=1) works by showing two samples of the model outputs, the first row is an image with three HHs and the second row is an image with 27 HHs, some of which overlap. The first column shows the original image, the second column shows the model output including the segmented image and number of HHs, the third column shows the true mask and the actual number of insects on the input, and the last column visualizes segmentation results on the input image.

As expected, the increase in the number of insects increases the chance of overlapping among insects, which negatively affects the performance. It should be noted that Y-Net is much less impacted by this issue compared to the method suggested in the previous chapter, as will be demonstrated in the next section.

The proposed Y-Net was tested on a smartphone. However, the main goal is to implement the model on resource-constrained MCU-based devices at the edge, enabling practical applications of automated insect monitoring. For this purpose, the model needs to be optimized, particularly in terms of memory usage and energy consumption. Therefore, in the following subsection, Y-Net is analyzed and optimized to make it suitable for deployment on resource-constrained MCUs.

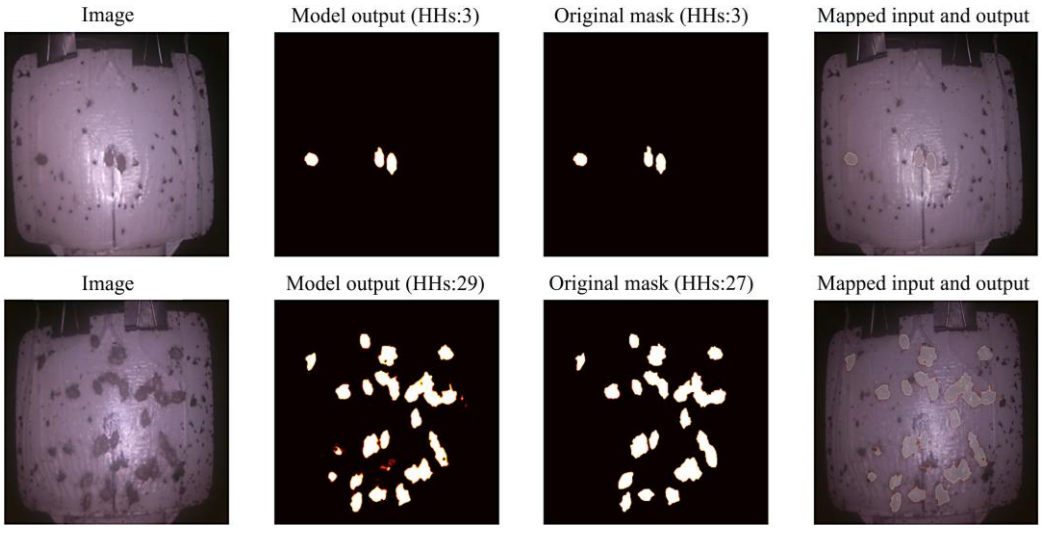


Figure 4.7. Example outputs of the proposed model showing predicted segmentation masks and insect counts for sample images, compared with the corresponding ground truth.

4.3 SemiY-Net: Squeezed Y-Net for Resource-Constrained MCUs

The use of resource-constrained devices imposes strict limitations on the design of an accurate DL model, specifically in terms of model size and RAM usage. In order to identify the critical layers of Y-Net, which highly affect RAM and model size, the original Y-Net was analyzed in terms of:

- Peak memory usage.
- Number of parameters directly affecting the model size.

The previous section examined several configurations of the Y-Net model; in the following, Y-Net with alpha value of 0.5 and input size of 320×320 was selected for analysis since it has a proper balance in terms of model accuracy and efficiency metrics (see Table 4.1).

4.3.1 Peak memory analysis and optimization

To analyse the Y-Net memory usage, the model was divided into several Blocks (B) for simplicity and better visualization. Figure 4.8 depicts the Y-Net model with MobileNetV2 with alpha value of 0.5 as its encoder and input size of 320×320 px. The mentioned division was performed based on the feature map shapes; for example, the input has the shape of 320×320px, Block 1 (B1) has the shape of 160×160 px and so on.

Figure 4.9 depicts the peak memory usage of the selected Y-Net. According to Figure 4.9, it is obvious that Block 10 allocates the highest memory usage to itself, at nearly 5MB, which is 5 times the available RAM on an MCU. Moreover, the majority of the blocks need more than 2MB of RAM for their computations.

Note that each block consists of several layers, in which one of them has the highest memory usage. This layer is selected as the critical layer and its memory usage is reported as the block peak memory usage. For instance, Block 2 with a peak memory of 2.65 MB has eight layers, one of which needs 2.65 MB while the other layers alone need nearly 2 MB on average. To identify the critical layers and factors that contribute to the peak memory usage of the model, the structure of the model was investigated as follows:

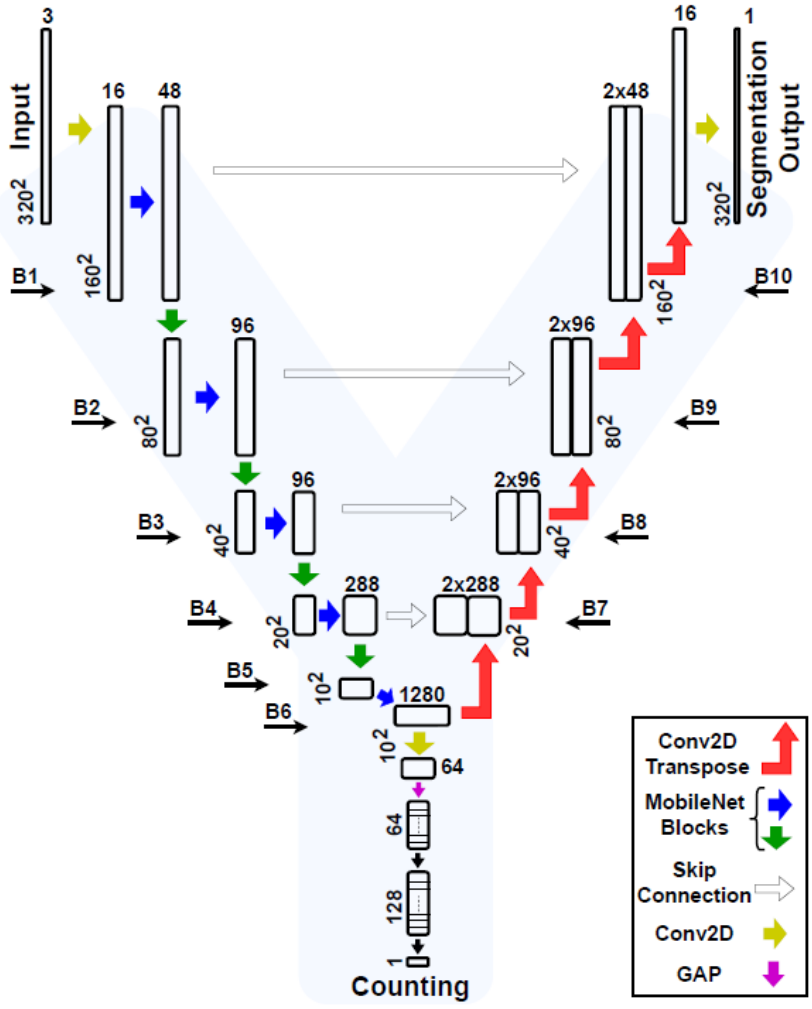


Figure 4.8. The proposed Y-Net architecture for input size of 320×320; Blocks (B) are defined for the simplicity of the model analysis.

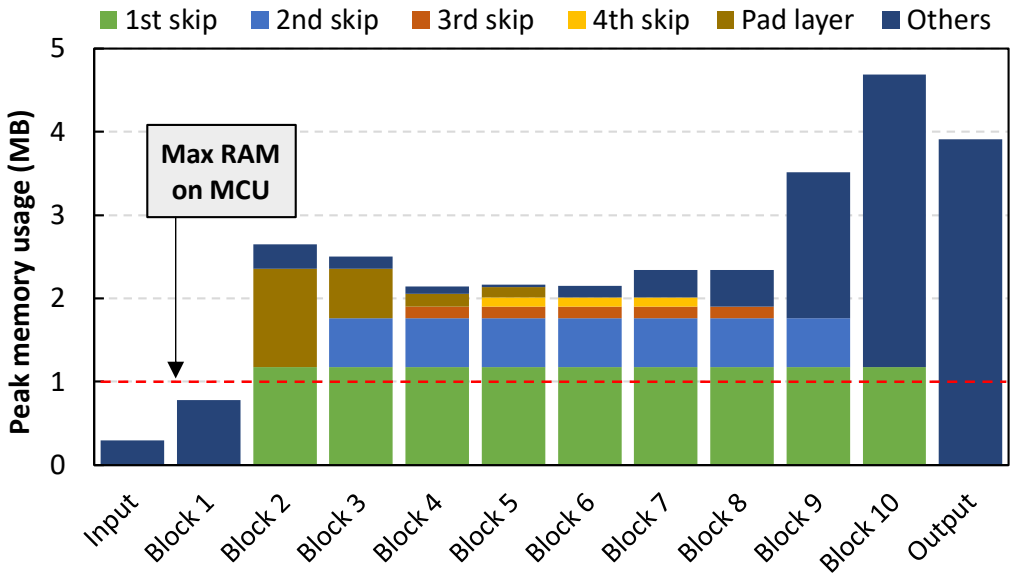


Figure 4.9. Peak memory usage of the proposed Y-Net model across different blocks during inference.

- Based on Figure 4.9, it is obvious that skip connections (in particular the first and the second ones) occupy a significant amount of block memory usage; for example, the first skip connection occupies approximately half of the memory usage of Blocks two to eight, which is around 1.2 MB. Skip layers copy information from a layer in the encoder to the decoder, meaning that this information should remain in the RAM until the computation reaches the destination layer of the skip connection. Thus, skip connections occupy RAM throughout the network until they reach the destination layer, where they copy the data.
- In addition to the skip connections, the padding layer also needs a considerable amount of RAM to perform its computation. Specifically, in Block two and three, this layer needs 1.2 MB and 0.6 MB, respectively. These padding layers belong to the pre-trained MobilnetV2 used for feature extraction in the encoder part.
- Moreover, the shape of the feature maps has a direct impact on the required RAM; for example, in Block 10, the output of a layer has the shape of 160×160×96 which alone occupies 2.3 MB of RAM.

By considering the mentioned points, SemiY-Net, shown in Figure 4.10, was proposed. As such, the following steps were adopted to design SemiY-Net based on Y-Net:

- Padding layers were removed from the encoder part in MobilenetV2.
- The first two skip connections were removed since they occupy a considerable amount of RAM.
- Channel numbers for a number of layers in Block 2 and decoder blocks were reduced to decrease memory usage below 1 MB.
- A convolution layer was used after each concatenation layer in the decoder part to reduce the channel number. In fact, the concatenation layer merges the skip connection and current feature maps, leading to an increase in channel number. Therefore, this convolution layer reduces the channel number and at the same time learns new features from the merged data.
- The two last blocks were removed from the model so that Block 9 would generate an 80×80 segmentation output.

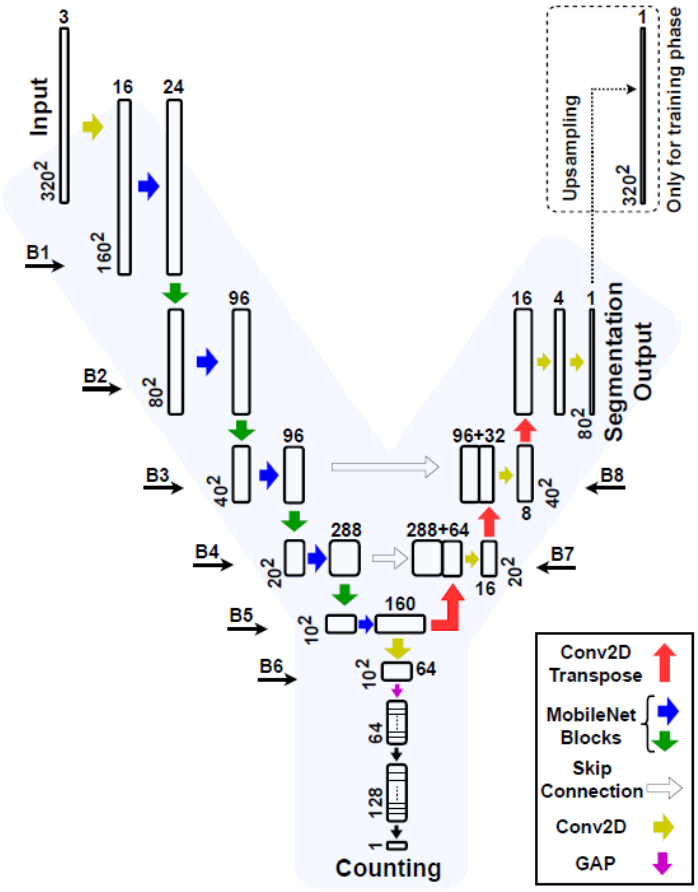


Figure 4.10. The proposed SemiY-Net architecture, a lightweight model designed for joint insect segmentation and counting on resource-constrained devices.

- An upsampling layer with the bilinear interpolation was added after the segmentation output to resize it from 80×80 to 320×320 px. This layer was only used during the training phase to resize the segmentation output image to the original size for the purpose of comparing the model accuracy with the original Y-Net and other models provided in Section 4.3.5. This layer was removed after model training; if needed, a simple image resizing method (e.g., with a nearest neighbour or bilinear interpolation) can optionally be used after the inference time as a post-processing step to resize the segmentation output.

Figure 4.11 shows the SemiY-Net's peak memory usage. Based on the points mentioned above, the entire model peak memory usage decreased to near 900 KB, less than the 1 MB limit.

Generally, in image segmentation tasks, the output is the same size as the input, but the proposed SemiY-Net does not follow this procedure where the input size is

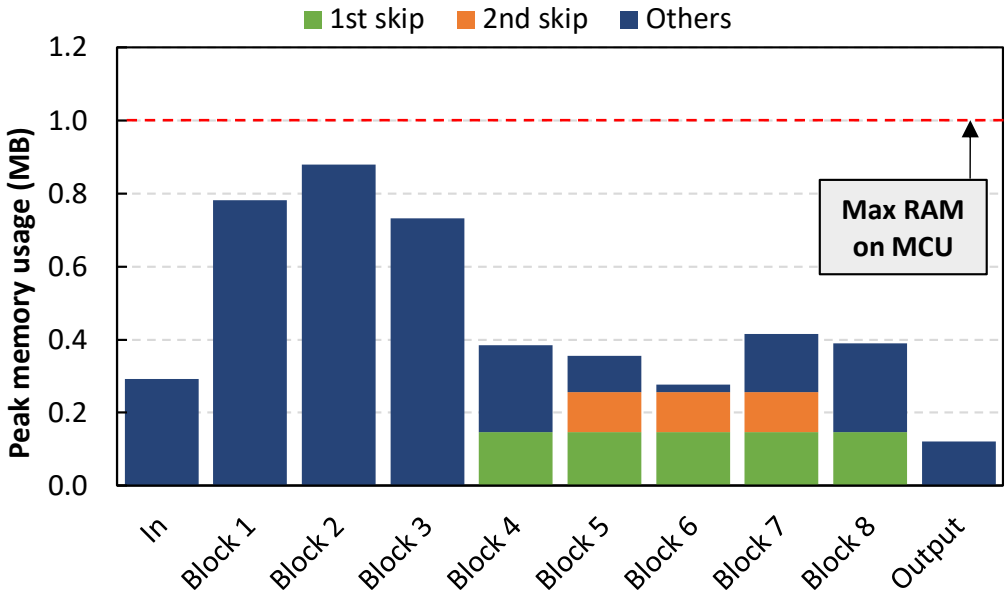


Figure 4.11. Peak memory usage of the proposed SemiY-Net model across different blocks during inference.

320×320 px while the output size is 80×80 px. Therefore, it is needed to resize the output from 80×80 px to 320×320 px to map the model output to the input image. This helps decreasing the RAM usage and, more importantly, makes it possible to run complex image segmentation tasks on MCUs with a RAM of only 1 MB.

4.3.2 Model size analysis

Regarding the model size, the Y-Net model had 5.5 million parameters. By using quantization techniques, the model size decreased from about 21 MB to 5.6 MB, which is still 3.6 times larger than the available storage on the MCU. The proposed SemiY-Net has 0.75 million parameters and the int8 model size is 0.94 MB, less than the 2 MB storage limit. As a result, it is possible to implement the proposed SemiY-Net on the MCU while the Y-Net model does not fit on the MCU storage.

4.3.3 Hyperparameters and training details

The dataset was split into training, validation, and test sets as discussed in Section 4.2.3. Additionally, images and masks were resized to 320×320 px. A histogram equalization technique (i.e., CLAHE) was also used to improve the image contrast, which has a significant impact on the image analysis algorithm [5].

Since the model has two outputs, two different loss functions were used for the model training. For the counting part, MSE was used as the loss function. For the

segmentation, BCE-Dice, which is the combination of Binary Cross Entropy (BCE) and Dice loss, was used as the loss function; BCE tries to minimize the pixel-wise differences while the Dice loss focuses on the overlap of the predicted and the true mask. The final segmentation loss was implemented as a weighted combination of BCE and Dice loss as follows:

$$Loss = (w \times BCE) + ((1 - w) \times Dice) \quad (4.1)$$

where w is the weighting factor controlling the contribution of the two losses. In this study, w was set to 0.6 after exploring various values during preliminary experiments.

As described, a pre-trained MobileNetV2 was used in the Y-Net encoder part for feature extraction, while some layers of the encoder part were modified to design SemiY-Net. Therefore, the original MobileNetV2 weights were loaded to the corresponding layers of the modified MobileNetV2 in the encoder of the SemiY-Net as initial weights, and then all layers of the modified MobileNetV2 were unfrozen. This allows the layer weights to be fine-tuned and updated during the training phase to enhance the modified architecture capability in extracting important features more effectively.

In addition, the learning rate of Adam optimizer was set to 0.0002, the epoch number to 500 and the batch size to 8 during the network training after exploring various values of these hyperparameters during preliminary experiments.

Moreover, augmentation techniques were applied on each batch during the training phase, including rotation, horizontal and vertical flip, random brightness and contrast.

4.3.4 Evaluation setup

In this section, the proposed SemiY-Net model is evaluated and compared to other models from different aspects. For this aim, several other models were trained to compare performance, including Y-Net, DeepLabV3+ [6], YOLOV8n-det and YOLOV8n-seg [7], and the method suggested in the previous chapter. It should be noted that all the models perform the segmentation task except YOLOV8n-det and the previous method. YOLOV8n-det is a detection model that provides bounding boxes for objects as an output, while the segmentation models provide a mask output which

is a pixel-wise prediction. For YOLO-det, these metrics are computed based on bounding boxes of detected objects while, for the segmentation models, they are calculated based on pixel-wise predictions. For this reason, these values may not be directly comparable but still provide valuable insight into the model performance; however, the counting metrics are directly comparable. In addition to this, unlike segmentation models that use binary masks for the training phase, YOLOV8n-det used the bounding-box labels for the model training. Regarding the previous method, in this chapter, the modified version proposed in Chapter three, Section 3.4 was employed.

Moreover, hardware efficiency metrics, including current and energy consumption during model inference, were measured using an Otii Arc Pro power analyzer that powered the OpenMV MCU-based board while the board executed the entire algorithm and the DL inference.

4.3.5 SemiY-Net evaluation

First, the impact of the segmentation output size on the model performance is investigated to find the best output size offering an optimal trade-off between the mentioned metrics, including model accuracy and implementation metrics. Table 4.2 reports the proposed model performance based on different segmentation output sizes. As the table shows, all SemiY-Net models have the same peak memory usage at 900 KB. As expected, increasing the output size (i.e., from 20×20 px to 160×160 px) directly affects and increases the MAC and inference time. Besides, it increases the segmentation accuracy, but it does not have a noticeable effect on the model size and counting metrics. Therefore, SemiY-Net-80 with the segmentation output size of 80×80 was selected as the optimal model since it has a reasonable balance between the accuracy performance metrics and the implementation metrics. Henceforth, in the remainder of this thesis, SemiY-Net will be denoted as SemiY-Net-80, corresponding to the proposed model configured with a segmentation output size of 80×80 px.

Moreover, to evaluate the impact of the added counting part on a segmentation-based model (U-Net), the proposed SemiY-Net without the counting part was assessed, where the counting was performed by applying a simple blob detection algorithm to the segmentation output. The results are reported in the last row of Table 4.2. In comparison with SemiY-Net, the results show that adding the counting part leads to a lower than 5% reduction in segmentation accuracy. At the same time, it significantly

improves the counting performance by decreasing MSE from 8.8 to 1.3. For the SemiY-Net without counting part, which used a blob detection algorithm for insect counting, the counting error rate is higher than the others. The reason is that when the overlap rate among the objects (i.e., insects) is high, the blob detection algorithms cannot accurately distinguish between objects and thus consider them as one object, increasing error. In terms of peak memory usage, the footprint of the counting part is less than 300 KB (see Block 6 in Figure 4.11), thus the peak memory usage of the whole model is 900 KB, the same as SemiY-Net.

Besides, the added counting part increases the DL model size by about 100 KB and MAC by approximately 9 million, leading to around a 20 ms rise in the inference time. Therefore, these costs allow the proposed SemiY-Net model to perform the counting more accurately in addition to the segmentation, which is necessary for insect monitoring applications.

To investigate the potential impact of removing pretrained layers, another version of the Y-Net model was implemented in which only the MobileNetV2 layers were modified and named "Y-Net (Modified-encoder)" and then trained on the dataset. The results are reported in Table 4.3. As shown in the table, by modifying the MobileNetV2 only, the peak memory usage of the entire model decreased to about 3.2 MB, while the original Y-Net was around 4.8 MB. Although this shows 1.6 MB reduction in peak memory usage, it is still higher than 1 MB of the available memory on the MCU. It should be noted that modifying the MobileNetV2 part only in the Y-Net led to a lower than 6 percent reduction in segmentation accuracy but did not have a significant effect on the counting performance. It also led to a threefold reduction in model size and a 1.7 fold reduction in MAC. To fit the model on an MCU, other parts of the Y-Net model were modified as explained in Section 4.3.1. Therefore, SemiY-Net meets the MCU limitations, however, this led to a reduction in the model accuracy as expected. Therefore, accuracy is partially sacrificed for a smaller model (i.e., in terms of size and memory usage) to be able to fit the model in the MCU. This highlights that there is a performance cost to implement DL models at the edge on resource-constrained MCUs [8][9].

Moreover, the proposed SemiY-Net was also trained from scratch (i.e., without loading the pretrained weights to the modified MobilenetV2 as the initial weights in

the encoder part) and named “SemiY-Net-Scratch”. As reported in Table 4.3, it is evident that the DSC metric increased around 5 percent and the MSE decreased by a factor of 1.6 from approximately 2.1 to 1.3. This shows that the encoder part can better extract the features when pretrained weights are used as initial weights.

Table 4.3. SemiY-Net evaluation based on different segmentation output resolutions on the test set of the 'Night' dataset.

Model	Segmentation metrics (%)				Counting metrics			Efficiency metrics			
	DSC	IoU	P	R	MSE	MAE	R^2	Size (KB)	MAC (M)	Peak mem. Usage (KB)	Inf. Time (s)
SemiY-Net-160	86.42	76.09	84.60	88.43	1.52	0.83	0.97	967	273	900	2.95
SemiY-Net-80	84.69	73.45	83.21	86.38	1.32	0.78	0.97	964	258	900	2.6
SemiY-Net-40	79.33	65.74	77.75	81.07	1.52	0.84	0.97	960	247	900	2.47
SemiY-Net-20	62.38	45.33	60.89	64.09	1.55	0.83	0.97	943	225	900	2.28
SemiY-Net-80 w/o-counting [a]	87.97	78.52	87.19	88.77	8.77	1.36	0.82	863	249	900	2.58

[a] Since the counting part was removed from the model, the counting was performed using a simple blob detection algorithm.

Table 4.2. The impact of encoder modification on Y-Net and SemiY-Net.

Model	Segmentation metrics (%)				Counting metrics			Efficiency metrics			
	DSC	IoU	P	R	MSE	MAE	R^2	Size (MB)	MAC (M)	Peak mem. Usage (KB)	Params (M)
Y-Net	92.34	85.78	91.63	92.94	1.1	0.78	0.96	5.55	7,231	4,800	5.54
Y-Net Modified-encoder	88.26	79	83.78	93.29	1.15	0.75	0.98	1.88	4,198	3,200	1.73
SemiY-Net	84.69	73.45	83.21	86.38	1.32	0.78	0.97	0.94	258	900	0.75
SemiY-Net Scratch	79.79	66.37	78.80	80.90	2.11	0.95	0.96	0.94	258	900	0.75

4.3.6 Performance metrics evaluation

Table 4.4 reports the models' performance in terms of their accuracy in counting and segmentation of the insect. The YOLOv8n-det has the best performance for both metrics; however, as mentioned, this model is a detection model and the output is bounding boxes. In terms of counting, the high level of R^2 for all models indicates a good match between the actual and the predicted number of insects. The proposed SemiY-Net outperforms the previous method in insect counting since the previous one depends on an image processing technique which is negatively impacted by the insect overlap, as discussed in the previous chapter. In terms of segmentation, apart from YOLOv8n-det which is a detection model, YOLOv8n-seg has the highest performance followed by Y-Net. DeepLabV3+, despite having large numbers of parameters (about 11.9 million), does not show comparable performance compared to Y-Net, YOLOv8n-det, and YOLOv8-seg with approximately 5.5, 3 and 3.3 million parameters, respectively. The proposed SemiY-Net achieved a DSC of 85%, an IoU of 73%, a precision of 83% and a recall of 86% which is less than others, but it only has 0.75 million parameters which is 7 times less than Y-Net, 4 times less than YOLOv8n-seg/det, and 16 times less than DeepLabV3+. This is a critical factor since the model parameters directly impact the model size. As aforementioned, accuracy was partially traded off to enable the implementation of the model on an MCU. Apart from reducing the model parameters in the proposed model, int8 post-training quantization was used, which led to around a 3 times reduction in model size from 2.8 to 0.94 MB. However, this improvement in model size came at a cost and led to less than a 1% reduction in segmentation metrics and less than a 0.05 increase in counting errors.

Table 4.4. Mean and 95% Confidence Interval (CI) of the accuracy metrics of the proposed SemiY-Net in comparison with other models for test set of the 'Night' dataset.

Model	Segmentation metrics				Counting metrics			
	DSC (%)	IoU (%)	P (%)	R (%)	MSE	MAE	R^2	Params (M)
SemiY-Net	84.69 (84.48,84.90)	73.45 (73.14,73.76)	83.21 (82.36,84.07)	86.38 (85.35,87.41)	1.32 (1.13,1.50)	0.78 (0.73,0.83)	0.97 (0.97,0.98)	0.75
Y-Net	92.34 (92.29,92.38)	85.78 (85.70,85.86)	91.63 (91.38,91.88)	92.94 (92.69,93.18)	1.1 (0.98,1.22)	0.78 (0.74,0.81)	0.96 (0.96,0.97)	5.54
Previous method	–	–	–	–	21.62 (19.10,24.13)	2.72 (2.56,2.88)	0.69 (0.67,0.71)	-
YOLOv8n-seg	94.80 (94.56,95.04)	90.12 (89.68,90.55)	94.56 (94.68,95.42)	94.80 (94.10,95.03)	–	–	–	3.26
YOLOv8n-det [a]	96.00 (95.83,96.18)	92.32 (92.00,92.64)	97.33 (97.07,97.60)	94.72 (94.38,95.07)	0.72 (0.62,0.83)	0.37 (0.35,0.40)	0.99 (0.99,0.99)	3.01
DeepLabV3+	89.46 (89.38,89.53)	80.97 (80.85,81.09)	93.8 (93.44,94.15)	80.81 (80.24,81.37)	–	–	–	11.85

[a] This is a detection model, while others are segmentation models.

Figure 4.12 shows the models' peak memory usage vs the models' size along with complexity (MAC) represented by the bubbles size. The green area in the figure highlights the regions in which a model can be implemented on the MCU (e.g., 1MB of RAM and 2MB of flash storage). It is evident that only the proposed SemiY-Net can run on the MCU, with a peak memory usage of 0.88 MB and a size of 0.94 MB. The figure clearly depicts the amount of compression from Y-Net to SemiY-Net; a 5.3-fold reduction in peak memory usage, decreasing from 4.69 to 0.88 MB, and a 5.9-fold reduction in model size, decreasing from 5.55 to 0.94 MB. For other models, the peak memory usage of YOLOV8n-det, YOLOV8n-seg, and DeepLabV3+ are 1.17, 1.34, and 26.56, respectively. In terms of model size, YOLOV8n-det and YOLOV8n-seg are 2.96 and 3.22 MB while it is 11.70 MB for DeepLabV3+. Note that DeepLabV3+ is not shown in the figure for better visualisation since its value is relatively large compared to other models. In terms of the models' complexity (which is reported by the MAC and shown by the size of the bubbles in the figure) the proposed SemiY-Net has about 258 million MAC which is approximately 4, 6, 28, and 57 times less than YOLOV8n-det, YOLOV8n-seg, Y-Net, and DeepLabV3+, respectively. This reduction in MAC is important since it directly affects inference time and power consumption. A higher MAC means more calculations that require more clock cycles to complete, which increases the inference time and results in higher power consumption.

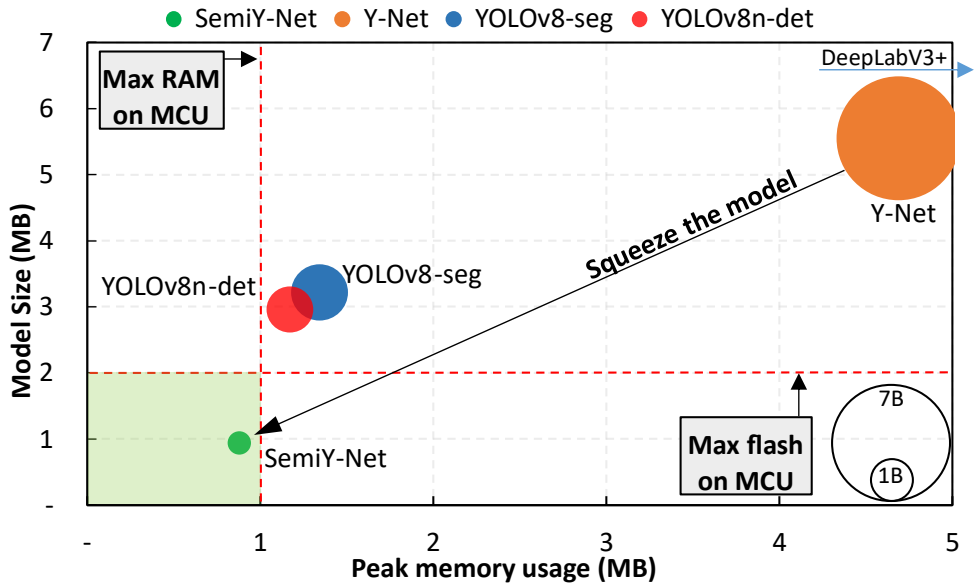


Figure 4.12. Comparison of models size (y-axis), peak memory usage (x-axis) and MAC (bubbles size); green region illustrates an area in which a model can be implemented on the MCU.

4.3.7 Efficiency metrics evaluation

Based on the results presented, the only model that can be successfully implemented on MCUs is SemiY-Net.

Since the model input size is 320×320 px, the board was adjusted to capture RGB images with this size. In one operation cycle, after the booting phase, the device captures an image, then the captured image is fed to the SemiY-Net model, and the results are saved in memory. Afterwards, the results including the captured image, mask and number of estimated insects are transferred via Wi-Fi to a nearby workstation. This is an optional step that can be followed to collect a dataset. In real-life deployment, the transfer of the decision over a long-range radio technology would be enough. Finally, the device goes into deep sleep mode until the next operation cycle to reduce the overall energy consumption of the system. Figure 4.13 shows the current consumption of the board over one cycle of operation. The starting point of each stage is shown by a vertical line and the average elapsed time is reported in this figure.

Around 44% of one operation cycle is accounted for system booting, while the proposed SemiY-Net inference time is 2.6 seconds which is 24% of the operation time. The image capturing and results saving phases only needed 0.2 seconds. Finally, the data transfer phase needed 3.25 seconds which is around 30% of the total time; it is

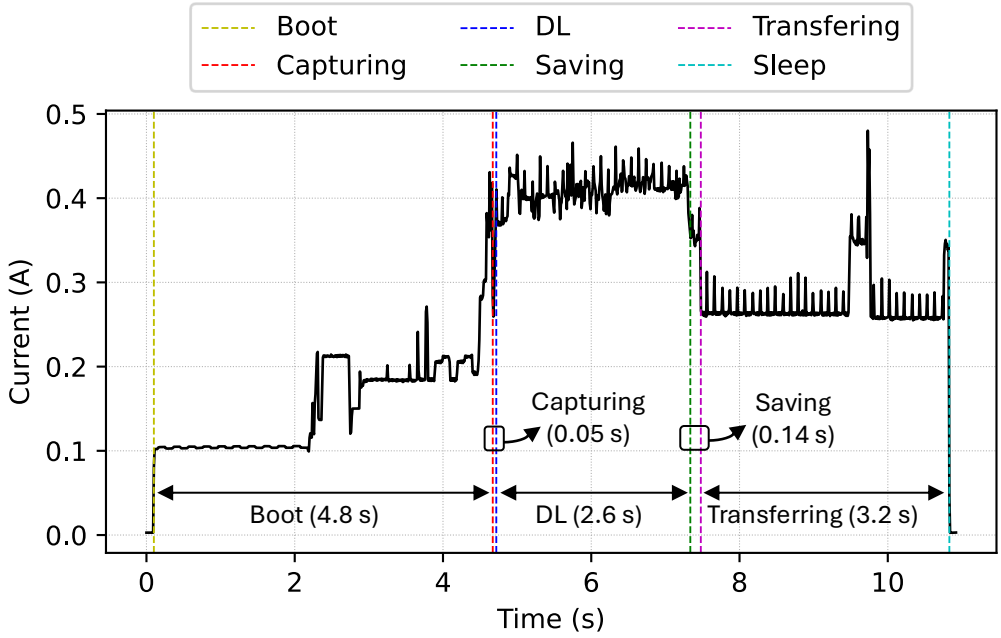


Figure 4.13. Current consumption of the proposed SemiY-Net during an operational cycle on the OpenMV MCU-based board.

worth mentioning that in this phase, a significant amount of time and energy consumption is associated with the Wi-Fi connection.

Moreover, Figure 4.14 illustrates the energy consumption of each phase separately. This figure shows that the proposed DL model consumed the highest proportion of the energy at 4.9 J while the boot and data transfer phase consumed 3.2 and 4.2 joules respectively. Although the boot and data transfer stages have the highest elapsed time, the DL model consumes more energy during operation than the others due to its higher current consumption. The average current consumption of the boot up stage and data transfer stage are 156 mA and 275 mA, respectively, while it is 409 mA for the DL phase. This fact clearly demonstrates the impact that DL models have on the overall energy consumption in battery-powered edge-based devices.

4.3.8 'Night', 'Day', and 'All' datasets

The performance of the proposed SemiY-Net is investigated on each dataset introduced in Section 4.2.2, including the 'Day', 'Night', and 'All'. The segmentation and counting performances are shown in Figure 4.15. These bar charts indicate that the model has the best performance on the 'Night' dataset, while the 'Day' one achieved the lowest performance among these three datasets. These results support the idea that taking images at night improves image processing performance in this task, as suggested in Chapter three.

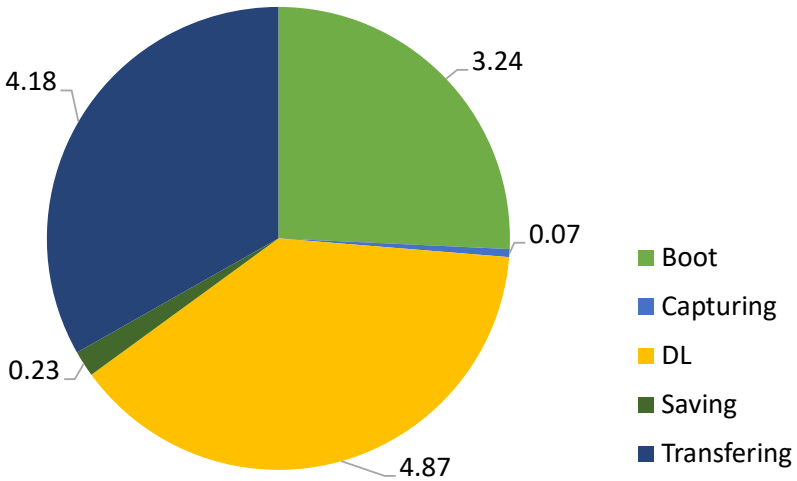


Figure 4.14. Energy consumption of the proposed SemiY-Net in joules during an operational cycle on the OpenMV MCU-based board.

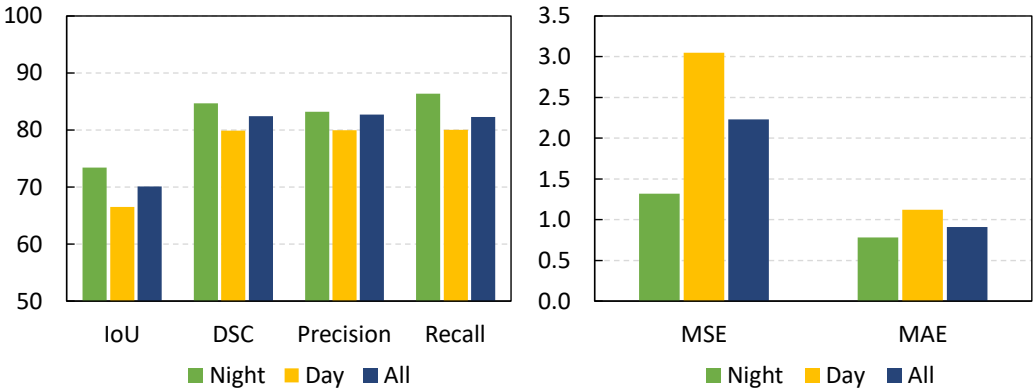


Figure 4.15. The proposed SemiY-Net model performance on the 'Night', 'Day', and 'All' datasets.

Figure 4.4 compares two samples of the 'Day' and 'Night' datasets. It is evident that the 'Day' image has a more complex background around the trap surface (e.g., trees and leaves) while the 'Night' one has a simple black background around the trap surface. Therefore, the impact of other sources of light is minimised, resulting in simpler images which facilitate model training.

Moreover, the 'All' dataset, which is the combination of the 'Day' and 'Night' datasets, has better performance compared to the 'Day' dataset. This shows that increasing the dataset size and including simpler night samples improves the model's generalization ability, even with the added complexity of daytime backgrounds. The 'All' dataset has two times more images than the 'Day' dataset. In addition to this, it has a lower performance compared to the 'Night' dataset since it has images from the 'Day' with a more complex background.

4.3.9 Discussion

Comparing DSC and IoU values of SemiY-Net shows that the DSC is approximately 10% higher than IoU, meaning that the model is less able to detect object (HHs) boundaries since the IoU metric is more sensitive to small mismatches, as mentioned in Section 4.2.4. When it comes to insect monitoring, precision at boundaries is not as critical as it is in some other applications such as medical imaging, since the main objective is to detect and count insects to track their population trend in the orchards. Based on this fact, the output segmentation size in SemiY-Net was reduced from 320×320 px to 80×80 px. By doing so, the model was simplified by removing the first skip layer and the last two blocks of the Y-Net, which have the

highest peak memory usage among the skip layers and the blocks, respectively. In addition to RAM usage reduction, this helps to reduce the inference time and the power consumption, which are critical factors for battery-powered IoT-based devices that should be deployed in remote areas with limitations in power resources.

In terms of the battery lifetime of such a device, the analysis revealed that the current consumption of the device in sleep mode is a critical factor since the device would be in this mode for almost the whole day. The average operation time of the device is approximately 11 seconds (see Figure 4.13), considering that the device operates two times a day, the device would be in sleep mode for almost the entire day except those 22 seconds of its operation time. The average current consumption in operation time is approximately 255 mA resulting in an energy consumption of approximately 1.6 mAh per day. For the sleep time (i.e., when the current consumption is about 7 mA), the energy consumption would be approximately 168 mAh which is 107 times more than the energy consumed during the operation time. For a device with a 5Ah battery capacity, the battery lifetime would be around 29 days. But if the sleep mode current consumption is decreased to 1 mA, the battery lifetime would be increased to approximately 196 days. This analysis clearly shows the impact of current consumption during sleep mode on the overall battery life of the device. Note that this is a theoretical estimation and several factors, such as battery self-discharge, can affect the actual battery lifetime. Moreover, since such devices are deployed outdoors, solar panels could be used to recharge the battery as suggested in the previous chapter.

Moreover, some examples of the proposed model output are shown in Figure 4.16. In this figure, the first column is the input image, the second one is the actual mask, and the third one is the detailed model output in which the true positive, false positive, and false negative pixels are represented by green, red, and blue colours. This figure shows three images with three different rates of overlap between insects, from low to high. It is evident that a higher rate of overlap increases the overall error; therefore, it is suggested to replace the trap more regularly and at shorter intervals, especially in the peak season. Besides, most of the unrecognized HHs were related to non-adult HHs and to HHs that were stuck on their sides; since the frequency of these occurrences was not sufficiently high in the dataset, the model has not yet been trained well to distinguish them.

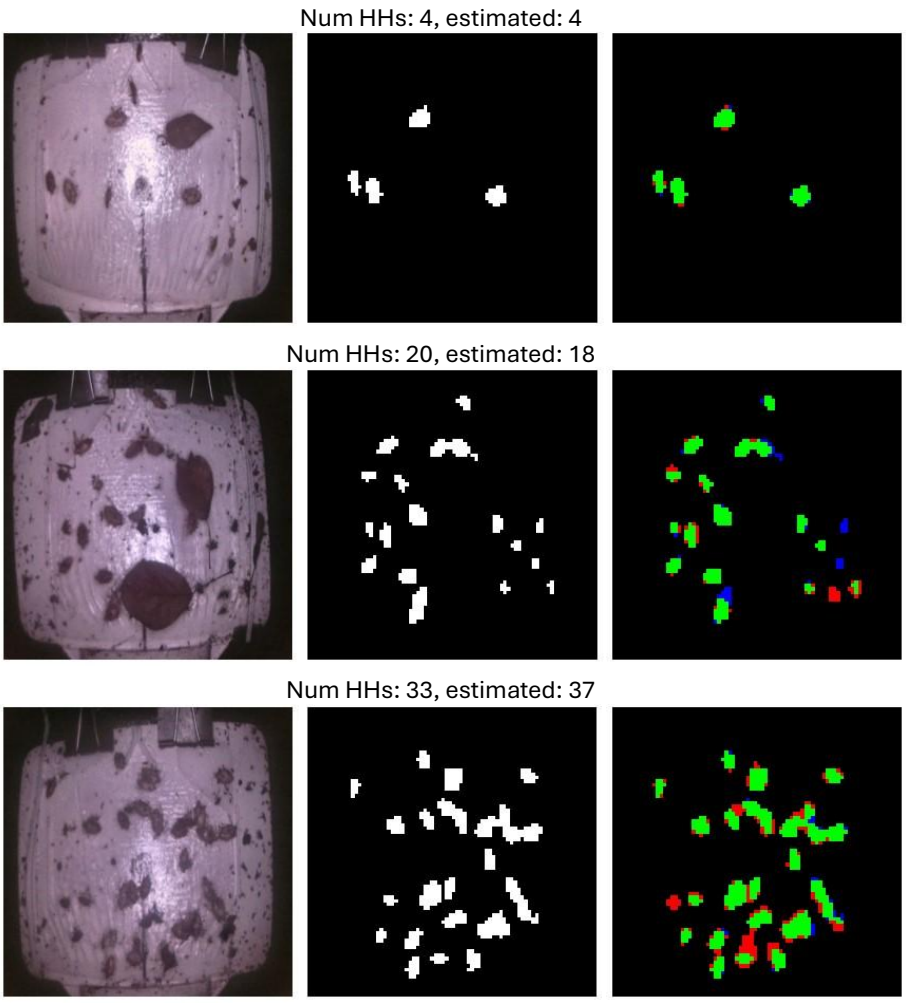


Figure 4.16. Samples of SemiY-Net outputs; first column is the images, the second column is the true mask, and the third column is the detailed model output (Green: TP, Red: FP, Blue: FN).

In terms of adaptability, the insect of interest in this study is a relatively large insect, meaning that the 320×320 resolution of the input image is acceptable for detecting and identifying the insect. For smaller insects, resizing the input image to 320×320 px could lead to removing their important features during image downscaling. By using image tiling which splits a high-resolution image into smaller tiles or patches (e.g., 320×320 px) instead of downscaling to reduce the input size no feature elimination would occur during the pre-processing. Therefore, the model could be adapted for even smaller insects using image tiling. However, this increases the processing time and consequently the energy consumption required since the model should run on each tile separately.

4.4 Conclusions

This chapter completes the second objective of this thesis by introducing and validating a custom TinyDL model that can run entirely on MCU-based edge devices without needing cloud or server support. This chapter presented a super lightweight SemiY-Net model for insect segmentation and counting on images. The proposed model was optimized to be implemented on resource-constrained hardware such as MCUs with up to 1 MB of RAM memory and 2 MB of flash storage. This enables battery-powered IoT-based edge devices to use low-power and low-cost MCUs as their processors to count and segment insects in remote orchards suffering from a lack of reliable Internet access and power resources.

However, it is needed to improve the robustness of the model to low-frequency samples in the dataset, which is critical for enhancing accuracy given the lack of large, well-balanced datasets for this application. In addition, after real-world deployment, environmental conditions and insect patterns change over time, leading to data shift and consequently a potential decrease in the model accuracy. To address these challenges, Chapter 5 presents a framework that enables the model to update and improve itself in the field using newly captured data, while minimizing human intervention.

4.5 References

[1] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9351, pp. 234–241, 2015, doi: 10.1007/978-3-319-24574-4_28.

[2] L. He, X. Ren, Q. Gao, X. Zhao, B. Yao, and Y. Chao, “The connected-component labeling problem: A review of state-of-the-art algorithms,” Pattern Recognit, vol. 70, pp. 25–43, Oct. 2017, doi: 10.1016/J.PATCOG.2017.04.018.

[3] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 4510–4520, Jan. 2018, doi: 10.1109/CVPR.2018.00474.

[4] M. Lin, Q. Chen, and S. Yan, “Network In Network,” 2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings, Dec. 2013, Accessed: Sep. 12, 2025. [Online]. Available: https://arxiv.org/pdf/1312.4400

[5] Y. Yoshimi et al., “Image preprocessing with contrast-limited adaptive histogram equalization improves the segmentation performance of deep learning for the articular disk of the temporomandibular joint on magnetic resonance images,” Oral Surg Oral Med Oral Pathol Oral Radiol, vol. 138, no. 1, pp. 128–141, Jul. 2024, doi: 10.1016/J.OOOO.2023.01.016.

[6] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, “Encoder-decoder with atrous separable convolution for semantic image segmentation,” in Proceedings of the European conference on computer vision (ECCV), 2018, pp. 801–818.

[7] G. Jocher, A. Chaurasia, and J. Qiu, “Ultralytics YOLOv8,” 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[8] M. S. Diab and E. Rodriguez-Villegas, “Embedded Machine Learning Using Microcontrollers in Wearable and Ambulatory Systems for Health and Care Applications: A Review,” IEEE Access, vol. 10, pp. 98450–98474, 2022, doi: 10.1109/ACCESS.2022.3206782.

[9] F. Svoboda, J. Fernandez-Marques, E. Liberis, and N. D. Lane, “Deep learning on microcontrollers: A study on deployment costs and challenges,” EuroMLSys 2022 - Proceedings of the 2nd European Workshop on Machine Learning and Systems, pp. 54–63, Apr. 2022, doi: 10.1145/3517207.3526978.

5 CHAPTER V. SELF-IMPROVING MECHANISM FOR INSECT IDENTIFICATION ON RESOURCE-CONSTRAINED EDGE DEVICES

The work in this chapter is adapted from the first author peer-reviewed articles as follows: Section 5.2 is primarily derived from article 1, and Section 5.3 from article 2.

1. **Kargar, A.**, Zorbas, D., Gaffney, M., O'Flynn, B., & Tedesco, S., "Concept Drift Mitigation on Resource-Constrained IoT Devices via Self-Learning," *in IEEE Sensors Applications Symposium (SAS)*, Newcastle, UK, Jul. 2025, DOI: 10.1109/SAS65169.2025.11105109.
2. **Kargar, A.**, Zorbas, D., Gaffney, M., O'Flynn, B., & Tedesco, S. (2025), "Self-Supervised Incremental Learning for Insect Segmentation and Counting on Resource-Constrained Devices," *IEEE Internet of Things Journal,* DOI: 10.1109/JIOT.2025.3630062.

Dataset:

3. **Kargar, A.**, Zorbas, D., Gaffney, M., O'Flynn, B., & Tedesco, S. (2025). Image Dataset of Brown Marmorated Stink Bug (BMSB) or Halyomorpha Halys (HH)-V3 [Data set], *Zenodo*, DOI: 10.5281/zenodo.16088064.

5.1 Introduction

The final step in this thesis is to move beyond a static model and build a comprehensive framework that can adapt and improve itself over time after deployment. Building on the TinyDL developed in chapter 4, this chapter continues with the third objective of this thesis by developing and validating a practical and comprehensive edge-based insect monitoring framework that addresses two fundamental issues of edge AI systems including data scarcity and post-deployment data drift.

In AI-powered IoT systems, a TinyDL model is typically trained offline using a labeled dataset and deployed on the edge nodes for online predictions [1]. The lack of suitable (e.g., sufficiently large and representative) labeled datasets is a common challenge during the initial implementation of these IoT nodes. Even when such datasets are available, the characteristics of the collected data may change over time due to data drift [2]. Therefore, it is necessary for the TinyDL model to be self-sufficient without relying significantly on model updates over the cloud.

Incremental learning (IL), also known as continual learning, is a technique that suggests retraining a DL model on batches of newly captured data. In a typical IL setup, newly captured data should be transferred to a cloud or remote server [3], where it is labeled by an operator and then used for model retraining. This approach is inefficient for real-world scenarios, especially when dealing with image data, due to limited communication bandwidth and battery capacity of IoT systems. In addition, manual labeling of data is time-consuming, tedious, and often requires expert knowledge [4]. Moreover, it raises privacy concerns regarding sharing local data with third-party platforms.

In this chapter, first, an on-device self-improving method is proposed for binary classification tasks based on K-Means clustering, a memory and computation efficient method, added on top of a TinyDL model, suitable for resource-constrained MCU-based devices. In this approach, features extracted using a DL-based model are utilized for classification and pseudo-labeling using K-Means, providing the core idea for handling data drift using lightweight operations. This allows edge IoT devices to effectively adapt to the dynamic environment they have been deployed in, without

transferring data to third-party platforms or requiring any user to label the new incoming data.

Building on this, the second part extends the same concept to a practical and more advanced framework to enable the proposed SemiY-Net, deployed on resource-constrained MCUs, to adapt over time to data drift. To this end, a comprehensive self-supervised IL framework suitable for resource-constrained IoT systems is proposed to fully perform on the edge and tackle the data drift issue for insect counting and segmentation. The proposed framework distributes the responsibilities between: 1) resource-constrained edge nodes, which perform inference using a TinyDL model (e.g., SemiY-Net), and 2) an edge server which performs operations requiring more computational resources, such as pseudo-labeling, image reconstruction, mask generation, and model retraining. The pseudo-labeling process in this framework is based on the approach introduced in the first phase, where the generated pseudo-labels are used to retrain and adapt the model over time.

5.2 On-device self-improving method for binary classification tasks

5.2.1 System architecture

The proposed method diagram is depicted in Figure 5.1. The method consists of three phases, the first two are carried out offline on a powerful device, either in the cloud or on a server, and the third phase is performed on resource-constrained IoT edge devices. In Figure 5.1, the individual sub-elements of these phases are also numbered, and these are described in the following sections. In the offline phase, the model is trained using labelled data and deployed on the IoT device. Then, in the on-device phase, in addition to the prediction, the model updates itself based on the newly captured data over time, which helps the model to adapt itself to data shifts.

5.2.1.1 Offline step

Training Feature Extractor (FE) (Phase 1): In this phase, a DL model was selected and trained on the available labelled data in the training set to extract relevant features by using a transfer learning-based DL model. Since the model should be able to run on resource-constrained IoT devices, a pre-trained MobileNetv2 [5] was selected with an alpha value of 0.35 and an input size of 160×160 as a main feature extractor (❶). The extracted feature map dimensions were then reduced using a GAP

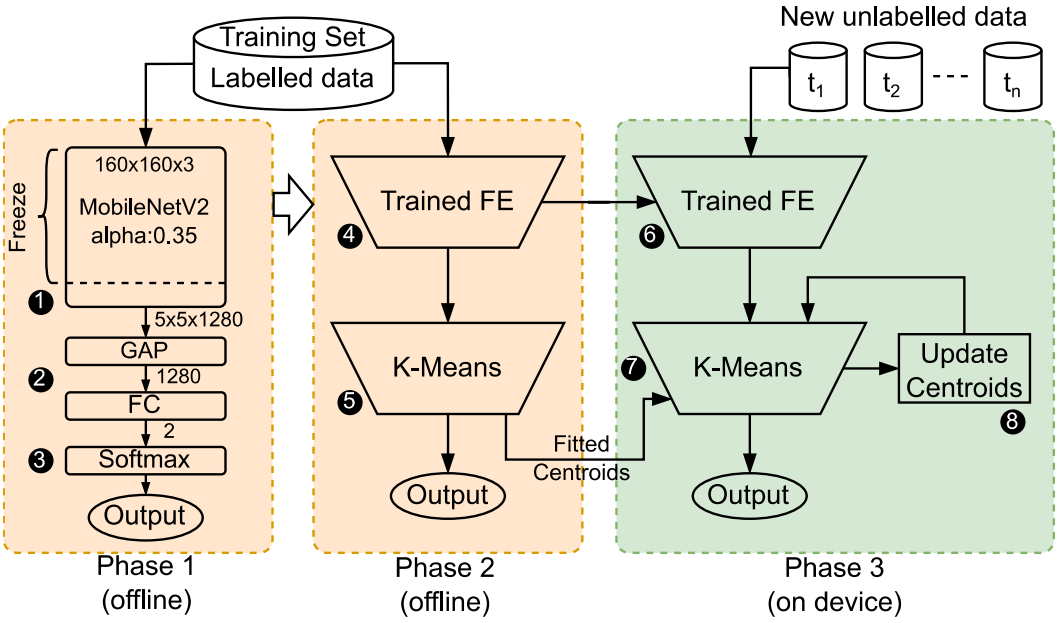


Figure 5.1. Overview of the proposed on-device self-improving method for binary classification.

layer and were then connected to a fully connected (FC) layer with two neurons (❷). Finally, the activation function of softmax was used for classification (❸). During the training, all layers of the pre-trained MobileNetv2 were frozen to prevent training and only the last layers were unfrozen to be updated and fine-tuned to better extract features from images of the used dataset. This is a general procedure that is widely used in such applications, but because of the concept drift that occurs in real-world scenarios, the model accuracy decreases over time. To overcome this accuracy reduction, the next phases were added to improve the model accuracy.

K-Means fitting (Phase 2): In this phase, the fine-tuned model from Phase 1 was used as a feature extractor (❹) to extract the features from the training set, then the K-Means method was applied and fitted to the extracted features for clustering and prediction (❺). This model was then exported to the IoT device to perform predictions on newly captured data. The exported model consists of the trained FE, used for feature extraction, and the fitted centroids of the K-Means, which are then used for clustering. It is worth noting that K-Means is a simple memory- and computationally-efficient method that is suitable for resource-constrained IoT devices.

5.2.1.2 On-device step

Phase 3: In this phase, IoT devices capture new data over time and perform the prediction task using the provided model from the offline phase, which includes

the trained FE (❻), responsible for feature extraction from the input, and the K-Means to perform clustering (❼). To decrease the accuracy loss due to concept drift, K-Means needed to be updated on new data reflecting the data shifts (❽).

Generally, K-Means is updated on batches of data, but this is not efficient or even possible to implement on resource-constrained devices. Therefore, updating the K-Means centroids on each individual collected sample was suggested. Therefore, for each new sample, the algorithm calculates the distance (e.g., Euclidean distance) between the sample point and each centroid and assigns the sample to the nearest cluster. Afterwards, the K-Means centroid to which the sample was assigned is updated using the following equations. Eqs. 5.1-5.2, which are based on the learning rate decay concept, along with Eq. 5.3, collectively guide K-Means smooth learning and prevent drastic shifts in centroid updates, mitigating the possible impact of outliers in the incoming data.

$$c_i = (1 - \eta) \times c_{i-1} + \eta \times x_i \tag{5.1}$$

$$\eta = \frac{lr}{n_s + 1} \tag{5.2}$$

$$c_n = c_n - M(c_n - c_{n-1}) \tag{5.3}$$

Where lr is an initial learning rate, which is a constant value, c_i is the K-Means centroids at iteration i, x_i is the features at iteration i (new data), n_s is the number of samples per class seen so far, and M is a constant value that limits the centroids to prevent drastic shifts.

5.2.2 Datasets

Two datasets were used in this study to investigate the performance of the proposed method:

***Cats and Dogs Dataset** [6]:* This dataset consists of around 25,000 images of cats and dogs. In this study, in order to simulate real-world scenarios where the captured image quality decreases over time, the dataset was split into 20 subsets and then different levels of noises were applied to each set (i.e., by applying Gaussian noise, simulated blurriness through resizing and colour shifts), the first subset includes original images without any noises while the last subset consists of images with the

Figure 5.2. Several samples from the Cat and Dog dataset with artificially added concept drift. The level of concept drift increases from the top-left corner to the bottom-right corner.

highest level of noises. Therefore, a dataset with concept drift was artificially generated to mimic real-world scenarios. The first subset was used for the DL model training and calculation of the K-Means centroids as labelled data, and the rest of the sets were used as an unlabelled data stream. Examples of the dataset are shown in Figure 5.2, including images in which the noise level increased from the top left corner to the bottom right corner.

***HALY.ID Dataset** [7]:* This is a binary classification dataset consisting of 3,899 images, out of which 1,777 images belong to a target insect, *Halyomorpha Halys* (*HH*), and the rest are other insects named Non-HH (2,122 samples). Images were captured two times a day from a pear orchard infested with the HH in 2023 using an IoT-based device described in detail in chapter 3. The dataset is split into 8 subsets based on the capturing dates and device setup. Visual inspection of images from these subsets clearly shows that concept drift has occurred over time, as is also evident in Figure 5.3.

5.2.3 Experiment setup and configuration

In this section, the performance of the proposed method is evaluated on two different datasets. For this purpose, two different FEs created from the trained model in Phase 1 were investigated:

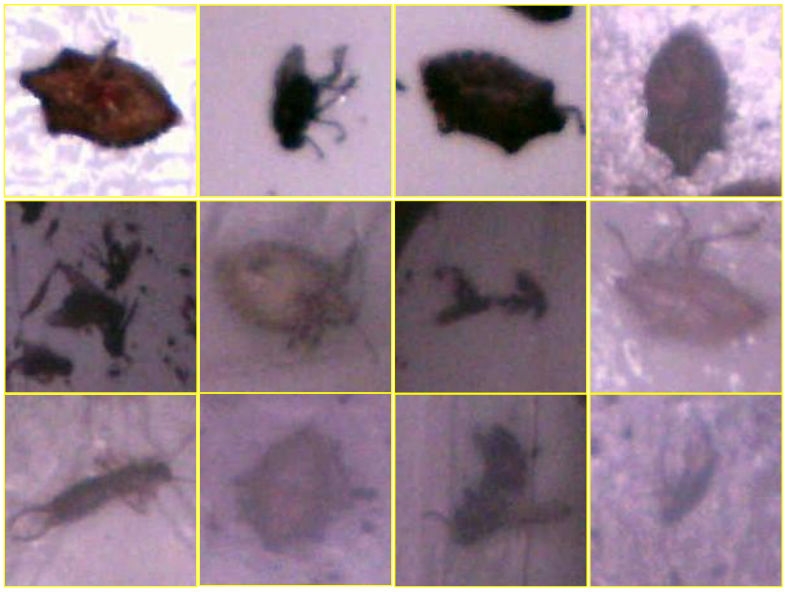


Figure 5.3. Samples from HALY.ID dataset which suffers from concept drift. The level of concept drift increases from the top-left corner to the bottom-right corner.

4. ***FE-GAP***: In this FE, the output features were taken after the GAP layer and layers after the GAP were removed from the FE (see Figure 5.1). With this FE the model extracts 1280 features.
5. ***FE-FC***: In this FE, the FC layer was selected as the output layer and the layers after it were removed (see Figure 5.1), thus the number of extracted features would be two.

To evaluate the performance of the proposed model, its accuracy was compared in three different modes against the baseline. These three modes along with the baseline are as follows:

- **KM (W-U)**: The proposed K-Means-based method with updating centroids.
- **KM (W/O-U)**: The proposed method without updating centroids (e.g., removing step 8).
- **KM (SK)**: It is the same as the KM (W-U) except that for the updating part, the partial-fitting method provided by the Scikit library [8] was used; this can be used on more powerful hardware such as Raspberry Pi boards that can load and run the Scikit library.
- **DL**: The baseline is the classical DL model trained for classification on the train set which is in fact the model trained in Phase 1.

Moreover, to train the DL model in Phase 1, the learning rate was set to a small value of 0.0001 since the last layer of the pre-trained MobileNetv2 was unfrozen to be fine-tuned on the new dataset. Additionally, the number of clusters in K-Means was set to $k = 2$, corresponding to the two classes in the dataset. The constant values in Eqs. 5.1-5.3 were adjusted by trial-and-error at $lr = 0.1$ and $M = 0.01$. An MCU-based OpenMV board was used to measure the power consumption and inference time of the proposed method in the device mode (Phase 3).

5.2.4 Results

In this subsection, the performance of the proposed method was evaluated on the two datasets in terms of accuracy, and hardware and implementation metrics.

5.2.4.1 Cats and Dogs dataset

Figure 5.4 shows the extracted features from the training set samples using both trained FE-GAP and FE-FC where the two classes are shown in different colours. Principal Component Analysis (PCA) [9] projection was adopted to reduce the FE-GAP dimension from 1280 to 2 for visualisation purposes. The first row of the figure illustrates that K-Means can be used for clustering on this dataset since samples

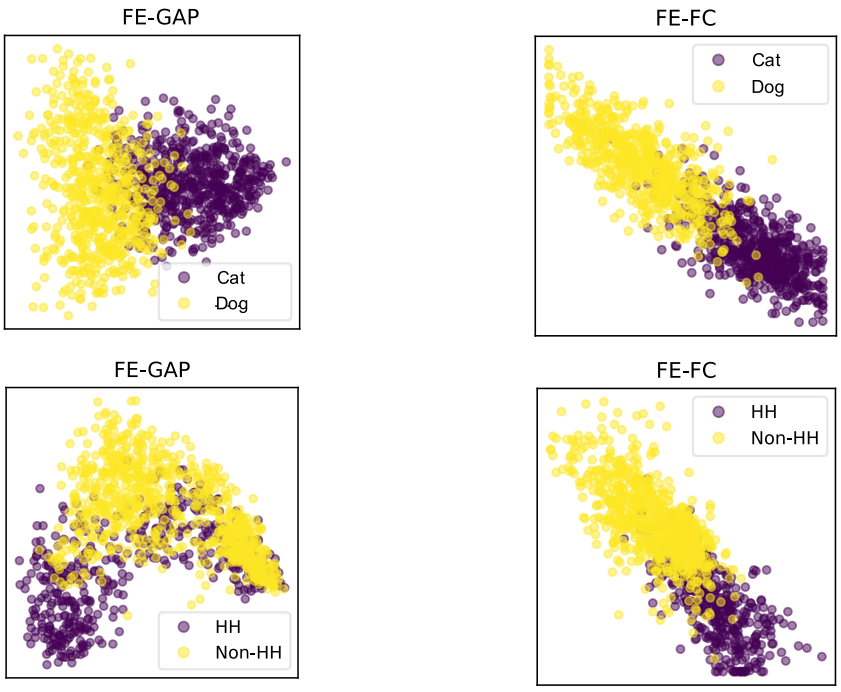


Figure 5.4. PCA-based visualization of the extracted features in 2D space. The first row belongs to the Cat and Dog dataset and the second row belongs to the HALY.ID dataset.

belonging to the same class are close to each other, while they are far apart from samples belonging to another class.

Figure 5.5 shows the performance of different modes of the proposed method by using FE-GAP and FE-FC compared to each other, as explained in Section 5.2.3; this is the average of ten iterations with different initial conditions, such as random weight initialization and data shuffles during training and fitting for all methods. As evident from the figures, the proposed method had higher accuracy compared to the baseline. Moreover, the baseline showed a higher reduction rate in accuracy over time, meaning that the proposed method can better adapt to the concept drift that occurs over time. Comparing FE-FC and FE-GAP reveals that the proposed method in all three modes had the same performance by using FE-FC while by using FE-GAP the proposed KM (SK) had better performance. Additionally, a five percent higher accuracy was obtained over all samples by the proposed method compared to the baseline.

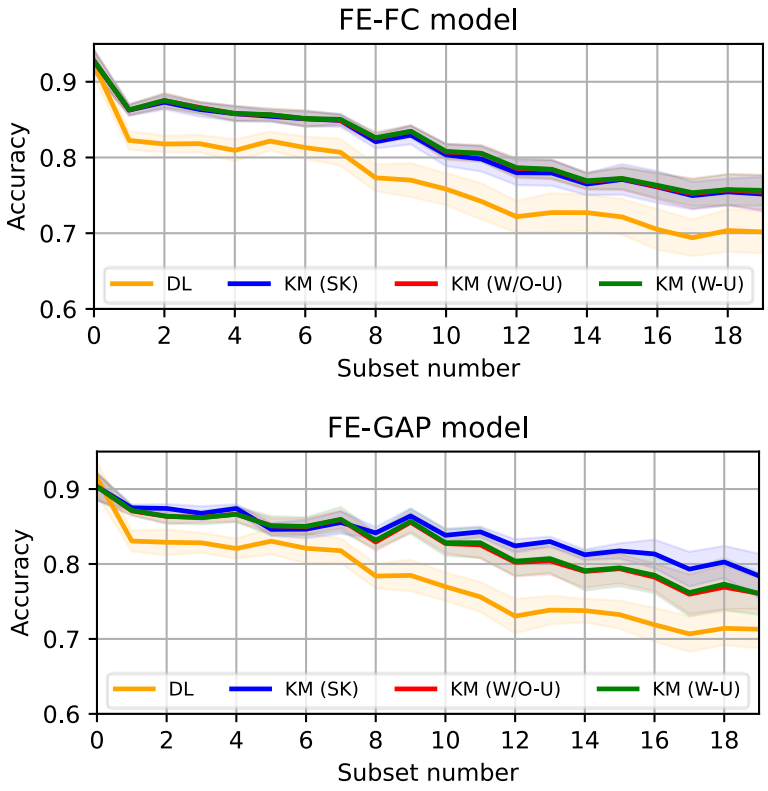


Figure 5.5. Accuracy performance of the proposed method by using FE-GAP and FE-FC compared to the baseline on the Cat and Dog dataset over time. DL: Baseline deep learning model trained in Phase 1; KM (SK): K-Means with Scikit-learn partial-fitting; KM (W/O-U): K-Means without centroid updates; KM (W-U): K-Means with centroid updates.

5.2.4.2 *HALY.ID dataset*

The extracted features of the HALY.ID Dataset are also illustrated in Figure 5.4 for both FE-GAP and FE-FC. Figure 5.6 shows the accuracy of different modes of the proposed method compared to the baseline. As opposed to the Cat and Dog dataset, using FE-GAP did not perform well, while the FE-FC worked better with a six percent increase in the accuracy over all samples in comparison with the baseline. This is expected because, based on the second row of Figure 5.4, there is a significant overlap between the two classes for FE-GAP, resulting in K-Means not being able to perform clustering accurately. Moreover, similarly to the previous dataset, the different modes showed equivalent performance.

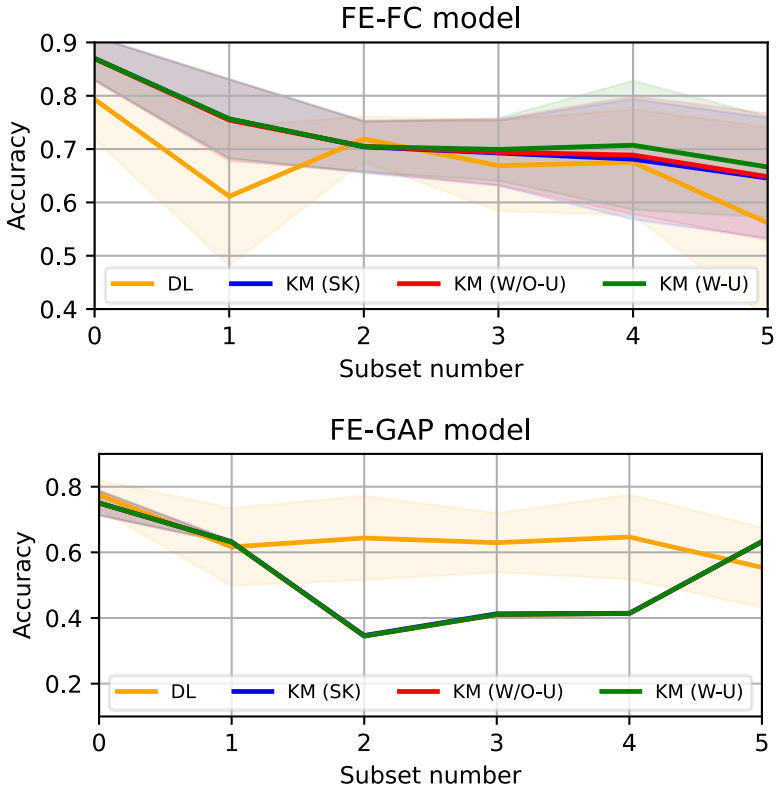


Figure 5.6. Accuracy performance of the proposed method by using FE-GAP and FE-FC compared to the baseline on HALY.ID dataset over time. DL: Baseline deep learning model trained in Phase 1; KM (SK): K-Means with Scikit-learn partial-fitting; KM (W/O-U): K-Means without centroid updates; KM (W-U): K-Means with centroid updates.

Table 5.1. Mean accuracy of the proposed methods using continuous learning compared to the baseline DL model without learning over the entire dataset. DL: Baseline deep learning model trained in Phase 1; KM (SK): K-Means with Scikit-learn partial-fitting; KM (W/O-U): K-Means without centroid updates; KM (W-U): K-Means with centroid updates.

	Cat and Dog		**HALY.ID**	
Method	**FE-GAP**	**FE-FC**	**FE-GAP**	**FE-FC**
DL	0.78	0.77	0.64	0.67
KM (W-U)	0.83	0.82	0.53	0.73
KM (W/O-U)	0.83	0.82	0.53	0.73
KM (SK)	0.84	0.82	0.53	0.73

To compare the performance of continuous learning with the baseline DL model without learning, Table 5.1 reports the mean accuracy of the proposed methods using continuous learning versus the DL model without learning. The table shows that the proposed methods increased classification accuracy by at least five percent, except for the GAP-based model on the HALY.ID dataset, as discussed above.

5.2.4.3 Hardware metrics and performance on MCU

In this work, quantization was used to reduce the model size and memory usage. Table 5.2 reports the performance of the proposed method in terms of hardware metrics on the MCU-based board. The model size and peak memory usage of the DL part used in the proposed method were approximately 610 KB and 608 KB, respectively, which are in the range of most modern MCUs available on the market. Therefore, the proposed method could be stored and run on MCU-based IoT devices.

Table 5.2. Proposed method hardware and implementation metrics on MCU-based board.

Method	**Inf. time (ms)**	**Energy (mJ)**	**Peak mem usage (KB)**	**Model size (KB)**	**Num params**
Full DL (baseline)	442	499	608	610	412,770
K-Means FE-FC	565	612	608	610	412,770
K-Means FE-GAP	641	688	608	607	410,208

In addition, the number of parameters for the DL model and FE-FC was about 412 k, while it was 410 k for FE-GAP since the classifier part was removed in this model. Regarding execution time and energy consumption per image, the baseline method needed 442 ms and 499 mJ to analyse each image, while these factors increased to 565 ms and 612 mJ for K-Means with FE-FC and 641 ms and 688 mJ for K-Means with FE-GAP, respectively. This is expected since the proposed method needs to perform K-Means prediction and update on the features extracted by the FE model, which increases the execution time and consequently the energy consumption.

Despite the similar structure of the proposed method in both FE-GAP and FE-FC, the method with FE-GAP had higher values of execution time and energy consumption. The reason is that FE-GAP requires the K-Means to be applied to 1,280 features, while FE-FC requires the K-Means to be applied to a vector with only two features.

5.2.5 Discussion

Based on the results, it is clear that the simple K-Means without updating, KM (W/O-U), had higher accuracy compared to the baseline DL model. This reveals that replacing the classifier with a simple K-Means clustering method can better deal with shifts in data. K-Means tends to achieve higher accuracy since it assigns the nearest cluster to the new data based on the extracted features. Therefore, it is more flexible to unseen data than the DL classifier, which generally suffers from poor generalization or overfitting, especially when the training set is small or lacks diversity.

Moreover, comparing the performance of the proposed method on both datasets highlighted the real difference between real-world scenarios and the artificially created ones. This is due to several factors: firstly, the device was deployed in an actual orchard and captured images two times a day. Images were captured during the night using an LED light to optimize the lighting conditions. However, as orchards are harsh environments, several factors affected the camera and consequently the captured images, from dust, dirt, and rain on the camera lens to high temperature fluctuations during the day that affect the camera sensor. These factors led to a high concept drift rate on the dataset that affected the performance of the proposed method. Besides, the target object was an insect, thus, their frequency varies over time (i.e., their population varies during the growing season), which makes the dataset and its

subsets imbalanced, while this is not generally considered in synthetically created datasets where different types of noises are added to public datasets. Thus, actual concept drift-affected datasets should be used in similar analyses rather than synthetic data. A potential solution to overcome such problems could be an adaptable learning rate mechanism in which the model analyses the input features and based on the level of shift adjust the learning rate. Although the learning rate was controlled, especially based on the number of samples for each class (n_s in Eq. 2), it could be more effective to adjust the learning rate based on shift level by automatically adjusting the constant values of the suggested learning rate (Eq. 1-3), which could be a future work.

Moreover, in this study, an MCU-based OpenMV board was used; however, more powerful options equipped with dedicated ML hardware accelerators can be explored in future work to achieve better performance. These are expected to reduce inference time and power consumption, which are critical factors for such applications.

In this section, a lightweight on-device self-improving method for binary classification tasks was proposed which was suitable for resource-constrained MCU-based devices. This allows edge IoT devices to effectively adapt to the dynamic environment they have been deployed without transferring data to third-party platforms or requiring a user to label the new incoming data. In the next section, a comprehensive but lightweight edge-based framework for the insect monitoring application is proposed to overcome both data scarcity and post-deployment data drift issues.

5.3 Self-supervised incremental learning for insect segmentation and counting on resource-constrained devices

This section introduces a lightweight edge learning framework suitable for resource-constrained IoT setups. It has two types of edge components: 1) Edge nodes, which are low-power MCU boards, and 2) an edge server, which is a more powerful node, for example, a Raspberry Pi. Initially, a compact DL model is trained on a cloud server using a small labeled dataset and deployed to the edge nodes. These nodes then perform on-device inference and periodically transmit pieces of data to the edge server. The edge server then reconstructs the input data, labels it using a lightweight algorithm, and incrementally retrains the original model. The updated model is subsequently sent to the edge nodes. This continuous loop enables on-site model adaptation, minimizes data transmission, and ensures autonomous operation without human involvement.

5.3.1 System architecture

Figure 5.7 depicts the proposed framework architecture and Figure 5.8 illustrates the main components output to provide a better understanding of the proposed framework. This framework consists of three main phases divided according to the hardware platform used for each of them. Phase A is carried out offline on a powerful server or cloud, while phases B and C are performed on the edge using low-power MCU-based nodes and a more capable node acting as the server node, respectively. The following subsections describe each component of these phases.

5.3.1.1 Phase A: Model training on the cloud server (Offline)

This phase, which consists of step ❶, is carried out only once on a powerful device, such as cloud or server, to train base models using limited available labeled data. Then, the trained models are deployed on the edge devices, including the MCU-based nodes and an edge server, in order to perform inference and retrain the model using the new captured data, respectively. For this application, the SemiY-Net model introduced in the previous chapter is used. In addition to this model, there is another DL-based model used for pseudo-labeling by the edge server node. In this framework, SemiY-Net is used for segmentation and counting on the edge nodes, while a separate classifier is employed on the edge server to support pseudo-labeling for SemiY-Net retraining. This separation allows efficient on-device inference while enabling

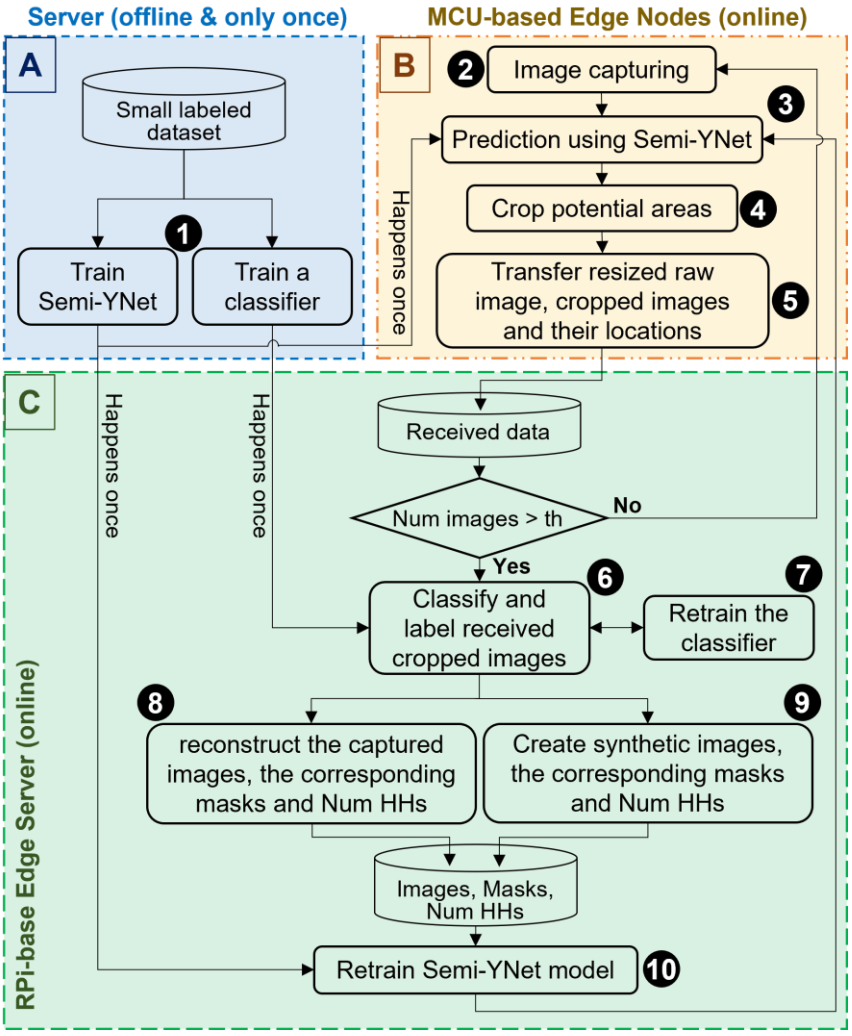


Figure 5.7. Flowchart of the proposed edge-based self-supervised incremental learning framework, showing the main components and the workflow from data acquisition to pseudo-label generation and model updating.

incremental learning and model adaptation. These models are initially trained on a server using available labeled data and then deployed on edge devices to perform inference and retraining during live operation.

5.3.1.2 Phase B: Inference on MCU-based Edge Nodes (Online)

This phase, which includes basic inference tasks, is performed on low-power MCU-based boards, such as the OpenMV board [10], deployed in the operational field. These edge nodes are responsible for two primary tasks described in the following.

5.3.1.2.1 Image Acquisition and Prediction

In step ❷, each edge node captures an image, and then (step ❸) the trained SemiY-Net model is applied to the captured image for prediction, providing the segmentation mask and number of the target insects on the image.

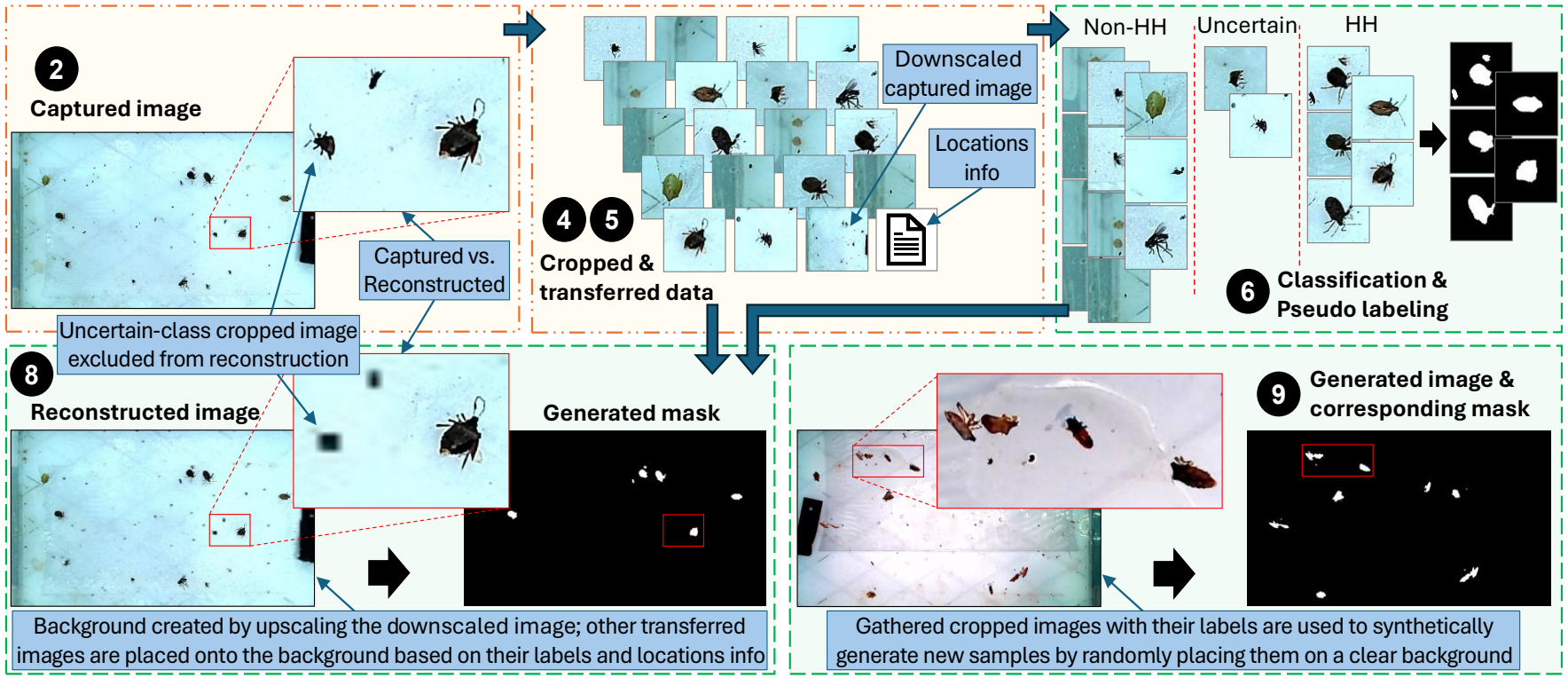


Figure 5.8. Main components visualization of the proposed framework. Steps 2 and 4&5 are performed on edge nodes (orange background), while the remaining steps are executed on the edge server (green background).

5.3.1.2.2 Transferring Key Regions of the Captured Image

Besides the prediction, the image is also required to be transferred to the edge server for retraining. But transferring the raw captured image is not practical and is not efficient in terms of time and power consumption, especially for edge nodes over low data-rate networks (e.g., LoRa networks) because of the limited data rate and bandwidth [11]. Therefore, in this study, instead of the entire image, only the key regions of the image are extracted and transferred to the edge server, where the images are reconstructed.

To this aim, in step ❹, potential areas of the captured image that might be occupied by insects are extracted and cropped using image processing operations, owing to the high contrast between the insect color (especially the brownish target insect) and the white background, which is a feature intentionally included in the imaging system design discussed in chapter 3. For this aim, the RoI detection algorithm explained in Chapter 3 was also used here, and then the detected RoIs cropped to 150×150 pixels and saved with JPEG compression to reduce the cropped image size. In the next step ❺, the original captured image is resized to 150×150 pixels (i.e., the same size as the cropped regions) and sent to the edge server along with the cropped images and their corresponding locations in the original image. Later, on the edge server, this data is used to reconstruct the captured images. This helps the system significantly reduce the transferred data, resulting in lower transfer time and power consumption.

5.3.1.3 Phase C: Edge Server Image Reconstruction and Retraining (Online)

The edge server is a more powerful node and is responsible for retraining the deployed SemiY-Net model using the newly captured data. This node collects and stores all received data and starts its task once the received data exceeds a predefined threshold (which is set as the number of images in each subset of the used dataset in this case) and is responsible for four main tasks described in the following.

5.3.1.3.1 Classifying the Received Cropped Images

Classifier architecture: The classifier architecture is illustrated in Figure 5.9. The classifier was relied on transfer learning using the MobileNetV2 model (i.e., $\alpha = 0.35$) [5] pretrained on the ImageNet dataset. The MobileNetV2 output is passed through a GAP layer to reduce extracted features dimensions from $5 \times 5 \times 1280$ to

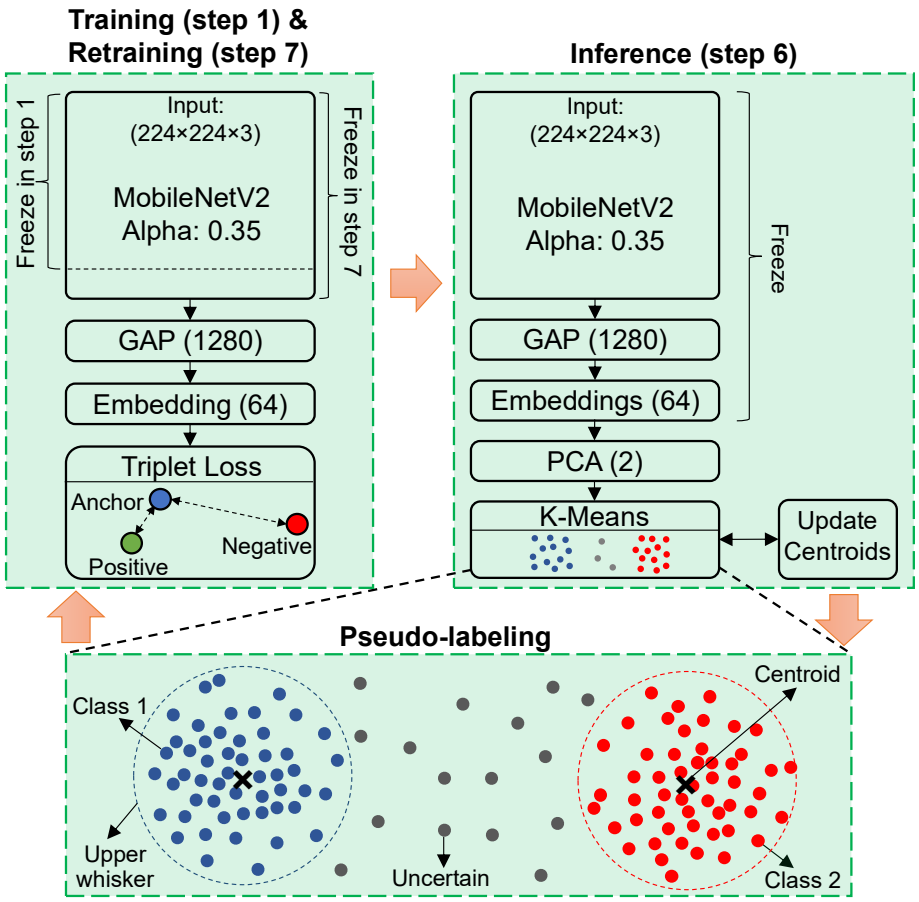


Figure 5.9. Structure of the proposed classifier and its main components, used for pseudo-label generation using K-means and triplet-loss-based feature learning.

1280. The features are then further reduced to a 64-dimensional embedding, and the model weights are optimized using a triplet loss function [12] in steps 1 and 7. This loss function attempts to minimize the distance between samples of the same class while maximizing their distance from samples of other classes. Triplet loss is employed because it effectively separates features of different classes in the embedding space, which is critical for the proposed approach, since the final classification is performed using K-Means. Indeed, the classification and labelling of each new sample is performed by assigning it to the class of the nearest centroid. Moreover, it is considered as a promising approach for scenarios where a model needs to be updated incrementally over time [13]. During the inference time, the 64-dimensional embedding is reduced to 2 features using Principal Component Analysis (PCA) [9] projection, and K-Means clustering is applied to the PCA output to assign each new sample the class of its nearest centroid. Moreover, as described in section

5.2, to better adapt the system to data shift, the K-Means centroids are updated using the newly incoming data.

Inference and training phases: This model is initially trained on a server using available labeled data in step 1 and then deployed on the edge server to perform inference and retraining during live operation. In step ❻, the framework classifies and labels (i.e., pseudo-labels) all the received cropped images using the classifier. Afterward, in step ❼, the DL part of the classifier is retrained using the labeled (i.e., pseudo-labeled) data to adapt itself to the newly incoming data.

During the initial training phase (step 1), all layers of the pretrained MobileNetV2 are frozen except for the last three ones in order to better fit the model to the dataset used in this study. During the retraining (step 7), instead, only the embedding layer is unfrozen to be adapted to the new incoming data, as shown in Figure 5.9. In addition, the initial K-Means centroids are fitted using the small dataset used to train the classifier model as part of step 1.

5.3.1.3.2 Pseudo-Labeling Mechanism

To retrain the classifier, labeled data is required. In this study, pseudo-labeling was leveraged to automatically label new images and consequently eliminate humans from the labelling process. Therefore, the new images in step 6 are classified (pseudo-labeled) using the suggested model (see Figure 5.9), and then, are used to retrain the classifier in step 7. To achieve the highest possible accuracy, it is important to ensure the precision of the pseudo-labels. Thus, the labels were filtered to remove those that might be inaccurate from the training phase. To this end, during the K-Means step (Figure 5.9), the distance of each sample to the centroid of its assigned cluster was calculated to remove samples whose distance exceeded a predefined threshold. In this study, the threshold was set as the upper whisker of the distance distribution observed during the training phase. Samples exceeding this value (i.e., outliers) were assigned to an “uncertain” class and excluded from classifier retraining. In the remainder of the paper, this K-Means-based mechanism is referred to as the modified K-Means version (M-K-Means).

5.3.1.3.3 Image Reconstruction & Mask Generation

In step ❽, the edge server reconstructs the captured images and generates the corresponding masks and number of the target insects using the received data,

including cropped images, their pseudo-labels and locations, and the resized background. To this end, the resized background image is rescaled back to the original size and used as the background. Then, the cropped images are placed onto the background based on their original locations to generate an image that is almost identical to the original captured image, effectively reconstructing the image with important regions. It is worth noting that cropped images labeled as “uncertain” are excluded in this step and not placed onto the background since their labels are not reliable. Moreover, all the cropped images classified as the target insect are converted to black-and-white images using the same algorithm described in Step 5 to generate masks. These masks are then placed onto a black background (i.e., based on their corresponding image locations) to form the final mask for the reconstructed image.

5.3.1.3.4 Synthetic Images & Their Corresponding Masks Generation

In step ❾, since many cropped images with their labels and masks are available in this stage, additional images can be synthetically created by randomly placing the cropped images on a clear background to generate more samples along with their corresponding masks, which improves the model generalization and robustness. Similar to the previous step, cropped images labeled as “uncertain” are excluded and not used in this step.

5.3.1.3.5 DL Model Retraining

Finally, in step ❿, the SemiY-Net model is retrained using all images, including both the reconstructed captured images and the generated synthetic ones, and then deployed to the edge nodes for prediction. During the retraining, all layers are frozen except for the last few layers (e.g., those in the segmentation and counting branches).

5.3.2 Dataset

The image dataset used in this study was gathered using an IoT device deployed in an orchard infested with the target insect (HH) in Italy during the growing season of 2024. Images were captured from a sticky sheet that attracted the target insect using a specific pheromone. The gathered images were then labeled and annotated for the target insect. The dataset consists of 476 images with a resolution of 1920 × 1080 pixels and a total number of 3,813 HHs, a mean of 8 ± 5 HHs per image, and a range of [0, 29] objects per image. Since the SemiY-Net model is used for binary

segmentation and counting, the dataset includes images with their corresponding black-and-white masks, along with a CSV file reporting the number of target insects in each image. During the image acquisition, the sticky sheet was replaced eight times, therefore, the dataset was split into nine subsets chronologically based on the replacement dates, each containing approximately 53 ± 28 images on average. The first subset, which was used for training in step 1, contains 41 images with 252 HHs.

5.3.3 Experimental setup and configuration

5.3.3.1 Hardware details

Edge Nodes: These are low-power nodes (e.g., OpenMV board) that are distributed in an orchard and are responsible for the steps in block B of Figure 5.7.

Edge Server Node: This node is used to handle block C of Figure 5.7, including steps 6 to 10. For this aim, Raspberry Pi 5 was selected since it is more powerful than the edge nodes but still energy-efficient to occasionally be used at the edge as a server to execute the mentioned tasks.

5.3.3.2 DL models & configuration & hyperparameters

As mentioned, the classifier was trained using triplet loss. For the SemiY-Net model, MSE and BCE-Dice were used for the counting and segmentation outputs, respectively, as described in the previous chapter. As the edge server (e.g., Raspberry Pi) has limited resources, a small batch size and fewer epochs during the retraining step were selected. Additionally, only some of the final layers of the models are unfrozen and updated to minimize memory and processing overhead. Therefore, during retraining on the edge server, the Adam optimizer was used with a learning rate of 0.0002 and 5 epochs for SemiY-Net. For the classifier, these values were set to 0.00005 and 2, respectively. As mentioned earlier, for the classifier, only the weights of the embedding layer were updated, while the remaining layers were frozen. For SemiY-Net, the output branches (segmentation and counting), along with the last three layers of the encoder, were unfrozen and updated during training. Moreover, for K-means, the number of clusters (K) was set to 2, corresponding to the two insect classes. In addition, data augmentation techniques, such as rotation, vertical/horizontal flip, and random brightness/contrast, were used for both models to improve generalization.

5.3.4 Results

5.3.4.1 Accuracy performance

In this subsection, the performance of the proposed framework in terms of accuracy is investigated.

First, the performance of the proposed classifier and the pseudo-labeling mechanism, which use the suggested DL model retraining and the modified K-Means algorithm, was evaluated.

To better illustrate how each part, including incremental retraining and M-K-Mean, contributes to performance, sample distributions related to subset 8 from the dataset are plotted in Figure 5.10 (top). In this figure, each point is colored based on its class label in blue and red. The first column displays the ground-truth labels of each sample point, while the second and third columns show the assigned classes obtained using standard K-Means and the proposed M-K-Means, respectively. The first row corresponds to the base model without incremental retraining, while the second row shows the results from the model retrained incrementally. By comparing the first column, it is obvious that retraining improved class separation and recognition, which led to higher accuracy in both standard K-Means and M-K-Means. In addition, it is clear that the proposed M-K-Means achieved higher accuracy. However, this was not across all data as some samples were classified as “uncertain”, and thus, were not used for the model retraining. This exclusion of data was intentional, as more accurate data was preferred for retraining, even if it resulted in fewer samples being used. Figure 5.10 (bottom) shows the average accuracy of the classifier across all subsets. By incrementally retraining the classifier model and using the proposed M-K-Means across all subsets, the system was able to select 83% of the data with an accuracy of 95%. In contrast, without incremental retraining, the classification accuracy was 96% on only 66% of the data using M-K-Means or 88% on the entire data using standard K-Means. This highlights the improvements made by both the proposed M-K-Means and incremental retraining of the classifier.

Next, the performance of the proposed framework was evaluated in terms of accuracy improvement over time. To compare the proposed framework performance, the following setups were considered:

- **Baseline (Base)**: This is a SemiY-Net model trained on step 1 without any further retraining, a classical DL model deployment approach.
- **Manual-IL (M-IL)**: This is a classical incremental learning approach in which the model is retrained using images labeled by experts. This needs to transfer the entire captured images and manually create masks and HH numbers.
- **IL**: This is the proposed framework in which the system automatically generates masks and HH numbers for each new image and incrementally retrains the model using them. This needs to only transfer important parts of the images instead of the entire images and also eliminates humans from the process. Several JPEG quality levels (IL90, IL50, IL25, IL10, IL1)

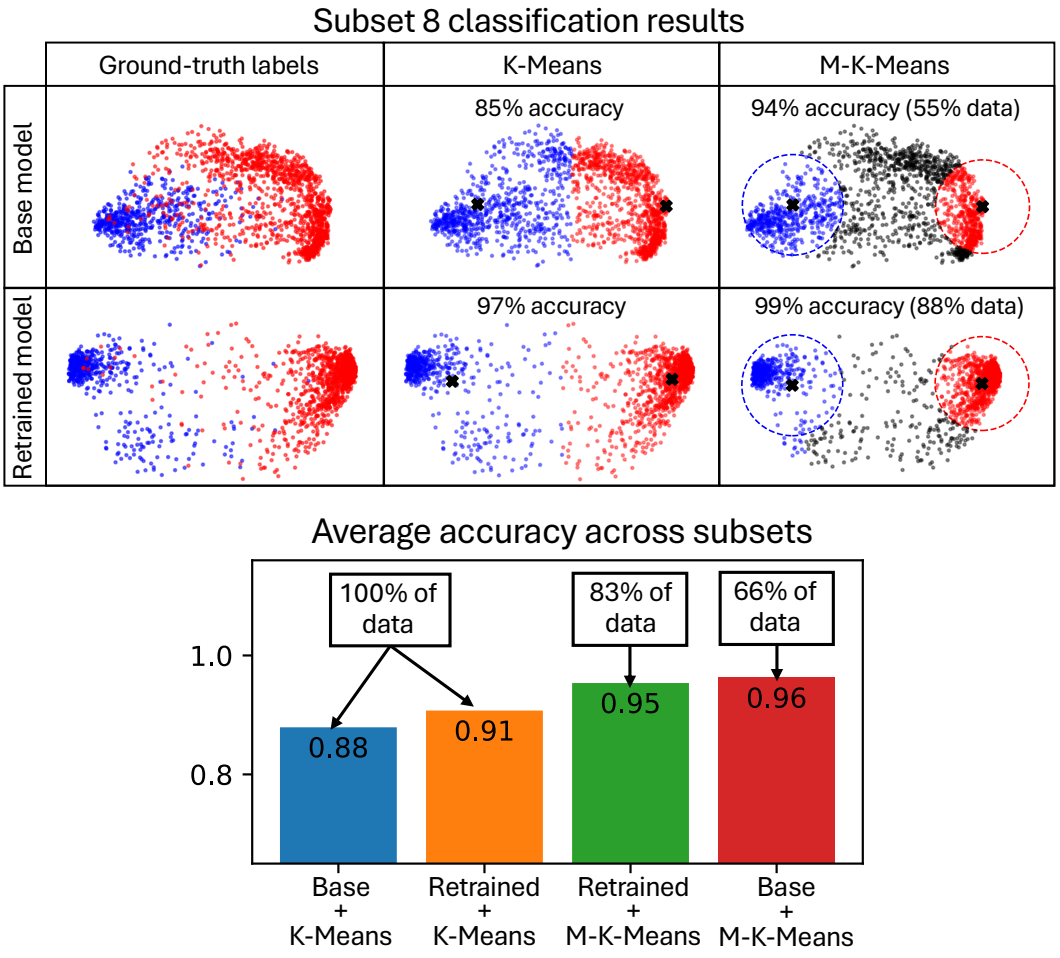


Figure 5.10. Classifier performance. Top: Scatter plots of samples from subset 8 colored by true labels, K-Means predictions, and M-K-Means predictions. The top row shows the base model without retraining, and the bottom row shows the model retrained on previous subsets. Bottom: Average classification accuracy across all subsets with the percentage of data retained after M-K-Means filtering.

were also evaluated to investigate the trade-off between compression and performance.

- **YOLO and Softmax**: These two were considered as external baselines. The first is YOLOv8n-seg retrained using confidence-threshold pseudo-labels to generate masks. The second is a Softmax-based version of the used backbone (without triplet loss and M-K-Means), retrained using confidence-threshold pseudo-labels. Note that the YOLOv8n, the smallest version of the model developed by Ultralytics [14], cannot be deployed on MCUs. In experiments, the YOLO-Base is a model without retraining, and YOLO-IL is incrementally retrained on subsets.

Note that, as mentioned earlier, the dataset was split into nine subsets chronologically. Therefore, the timeline was divided into nine steps corresponding to each subset. The base model was trained on subset 0 (the first subset), and the remaining subsets (1 to 8) were used to evaluate the proposed system performance over time. Figure 5.11 demonstrates the SemiY-Net accuracy using the proposed framework (IL) for JPEG compression levels of 90, the Manual-IL (M-IL), the Baseline (Base), Softmax and YOLO-Base and YOLO-ret over time. For the segmentation output, the accuracy was evaluated by DSC, and the counting output was assessed using MSE. According to the figure, it is clear that the proposed framework outperformed the baseline in both segmentation and counting. In addition, its accuracy was close to that of the Manual-IL, which was a desirable outcome and showed that the generated labels and masks were similar to the manual ones created by humans. Moreover, in some cases, the proposed framework even achieved higher accuracy than the Manual-IL. This can be attributed to the fact that, in the proposed framework, alongside the newly captured images, additional synthesized samples were used which helped the model to better understand and detect the target insect. Regarding external baselines, YOLO-IL achieves a lower average MSE compared to the proposed method (5.3 vs. 9.3), but its average DSC is lower (53% vs. 62%). For the Softmax-based

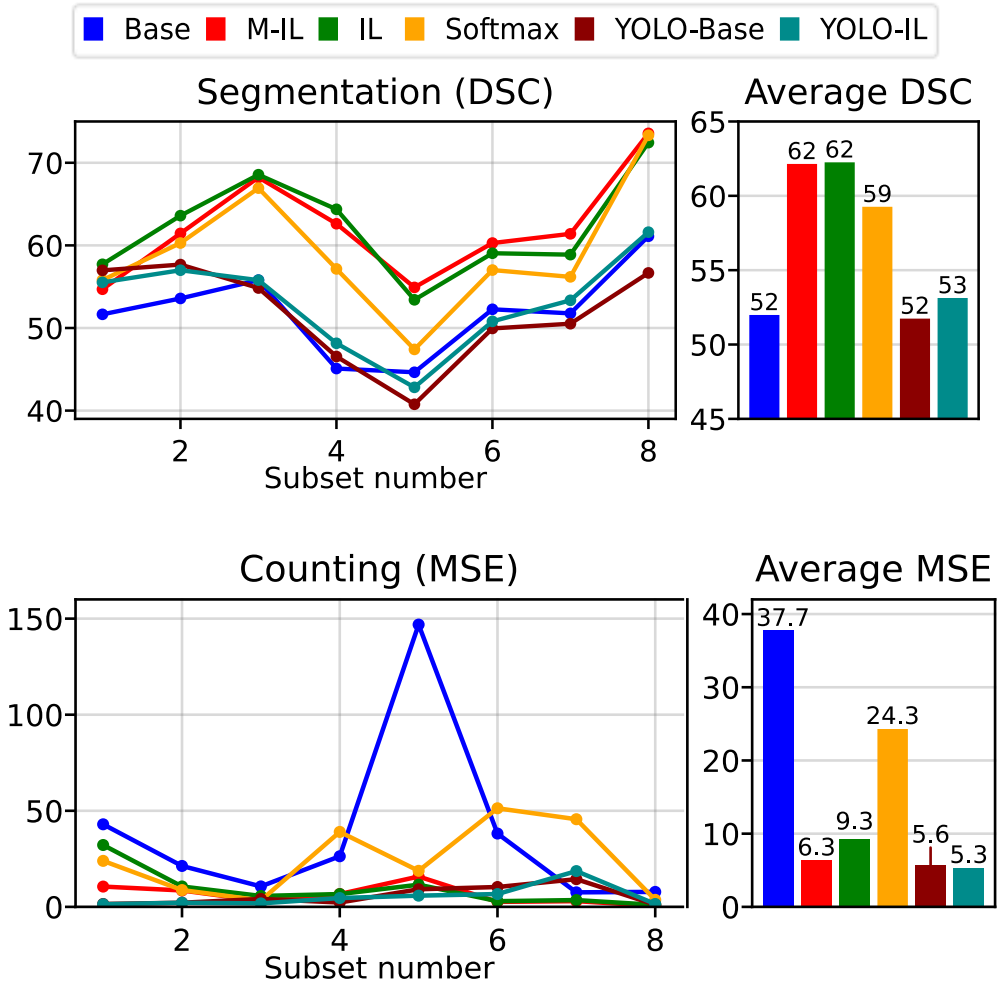


Figure 5.11. Segmentation and counting metrics of the proposed framework (IL) compared to Manual-IL (M-IL), the Baseline (Base), Softmax, YOLO-Base, and YOLO-IL. Left: Metrics on each subset over time. Right: Average metrics across all subsets.

model, the proposed method outperforms it in both average DSC (62% vs. 59%) and MSE (9.3 vs. 24.3).

Moreover, Figure 5.12 shows the performance of the proposed framework (IL) under different JPEG compression levels, including 90, 50, 25, 10, and 1. Surprisingly, there was only a minor difference in the accuracy of the proposed framework when using different levels of JPEG quality, from 90 (IL90) to 1 (IL1), which motivates the use of higher compression (i.e., lower JPEG quality) to reduce the transfer data volume

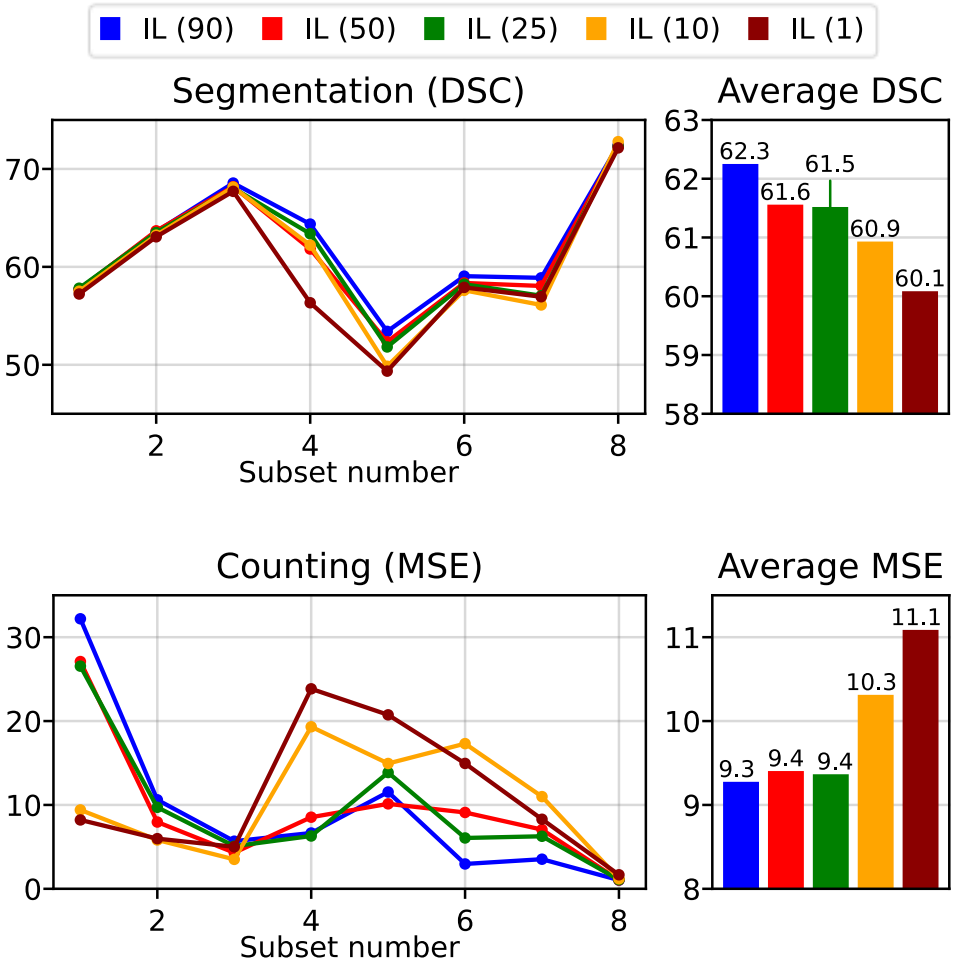


Figure 5.12. Segmentation and counting metrics of the proposed framework (IL) under different JPEG quality levels. Left: Metrics on each subset over time. Right: Average metrics across all subsets.

and consequently lower communication power consumption, as discussed in detail in the next section.

Apart from the above comparison, and to fairly compare the performance of the proposed framework with standard full training, the framework was applied by initially training the SemiY-Net model on subset 0 (step 1) and then incrementally updating it on subsets 1 to 7 in order, following the defined procedure (steps 4 to 10). Then, the final evaluation was performed on subset 8, which was not used in any training step. For the standard full training, the SemiY-Net model was trained on the combined data from subsets 0 to 7 and similarly evaluated it on subset 8. Since both models were trained on the same data and tested on the same subset, this ensures a direct and fair comparison. The proposed framework achieved an MSE of 1.9 and a DSC of 69.2%, while for the standard full training, these values were 1.5 and 71.3%,

respectively. The first case only used labels from subset 0 (i.e., 41 images), while the second case relied on labels from all the images in subsets 0 to 7 (i.e., 361 images). Thus, the proposed approach achieved comparable performance using 8.8× fewer labels. The results clearly demonstrate the effectiveness of the proposed framework and, more importantly, show that such models can be deployed in real-world scenarios where large labeled datasets are unavailable. The framework starts with a small amount of labeled data and improves itself over time using newly incoming data without human intervention, achieving results comparable to full training on a much larger dataset.

5.3.4.2 Incremental learning performance analysis

The performance of the proposed IL framework was evaluated using standard metrics defined in Appendix A to assess the overall accuracy, stability, and plasticity. The proposed approach is designed to balance the stability-plasticity trade-off. Most layers are kept frozen during retraining to preserve previously learned knowledge (stability), while only the last layers are unfrozen with a small learning rate and limited epochs to allow adaptation to new data (plasticity). This acts as a simple regularization-like strategy that constrains weight updates and reduces forgetting. Additionally, a lightweight replay mechanism was used by generating artificial images where earlier cropped insect instances are randomly pasted onto clean backgrounds (step 9 in Figure 5.7-Figure 5.8), which reuses earlier data without a large memory overhead (each cropped image is only about 4.25 KB at 90% JPEG quality).

To further evaluate the impact of these adopted strategies, an ablation study was carried out. For this purpose, a baseline was also tested with no artificial image generation, more unfrozen layers (e.g., 8 more layers), and a higher learning rate (e.g., 0.001). The comparison between the proposed method and the baseline is reported in Table 5.3.

Table 5.3. Ablation study comparing the proposed method with baseline to show impact of adopted strategies using AIA, BWT and FWT. As counting is evaluated as an error, AIA is reported as AIE (lower is better).

	Segmentation (%)			**Counting**		
	AIA	**BWT**	**FWT**	**AIE**	**BWT**	**FWT**
Proposed Method	60.6	-0.6	-6.2	11.7	1.5	-0.6
Baseline	58.6	-6.4	-6.2	20.2	-11.7	-8.9

Compared to the baseline, the proposed approach performs better in both segmentation and counting. For segmentation, it achieved better accuracy (AIA of 60.6 vs. 58.6), about six percent less forgetting (BWT -0.6 vs. -6.4), and similar FWT (-6.2 for both). For counting, the proposed method achieved about two times lower error (AIE of 11.7 vs. 20.2), with much improved stability (BWT 1.5 vs. -11.7) and less negative FWT (-0.6 vs. -8.9). This highlights that the baseline was highly affected by forgetting, while the proposed method was more stable and had better performance. Therefore, the strategies used helped the model to keep performance consistent across subsets and effectively address forgetting.

Moreover, it is also important to note that the dataset was collected from real-world orchards and split chronologically, which naturally makes it imbalanced. This imbalance impacts the performance, since some subsets have higher insect density and more complex appearances than others. The peak, especially for the Base in the MSE chart, is a result of this natural domain shift. The Base model is more affected by these shifts while the retrained models adapt better to the new subsets.

The counting and segmentation losses for the proposed framework is shown in Figure 5.13. It is obvious that both segmentation and counting losses generally decrease across the update steps, which confirms the effectiveness of the proposed framework over time. Some fluctuations are observed which are associated with variations in data distribution, but the overall trend remains downward and converges to lower values. To highlight this behavior, trendlines were also included, which clearly show the overall downward direction. These results confirm that the fine-tuning process remains stable even under realistic imbalanced real-world data. This highlights the suitability of this approach for real-world deployments.

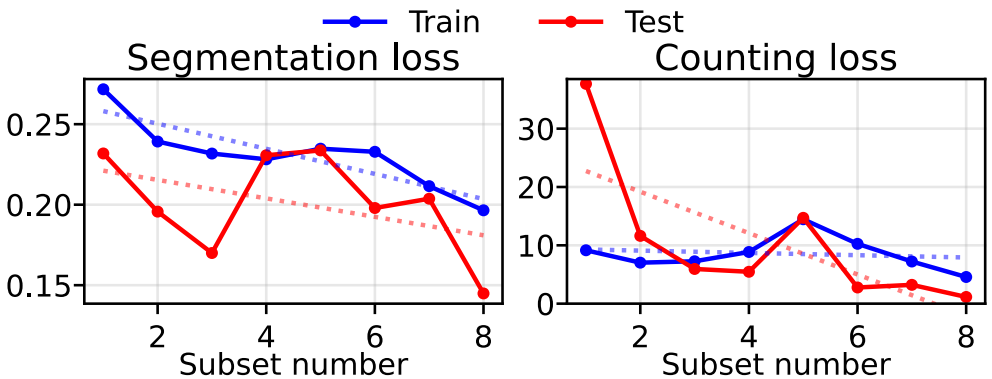


Figure 5.13. Counting and segmentation losses across incremental retraining, showing an overall downward trend.

5.3.4.3 Data transmission evaluation

Data transfer is one of the most power-intensive operations in IoT nodes [15]. Therefore, to optimize this process, it was proposed to transfer only the important regions of images instead of the entire image. In this section, the performance of this proposed strategy was investigated. The proposed strategy was applied to the dataset, and the results indicated that an average of 24 small files needed to be transferred per original image. Figure 5.14 (top) shows the file size distribution resulting from the proposed transfer strategy using different JPEG compression levels in comparison with sending the entire original images. One of these files indicated the locations of the cropped regions, while the remaining files were 150×150 images cropped from the original image. Based on the figure, the original images had a median size of approximately 212.3 KB, while the cropped images had median sizes of about 4.25, 2.08, 1.64, 1.31, and 1.08 KB for JPEG compression qualities of 90, 50, 25, 10, and 1, respectively. This led to at least a 50-fold reduction in the file size that needed to be transferred in each transmission, which is crucial for low-data-rate networks such as LoRa. As a result, as shown in Figure 5.14 (bottom), transferring the entire original

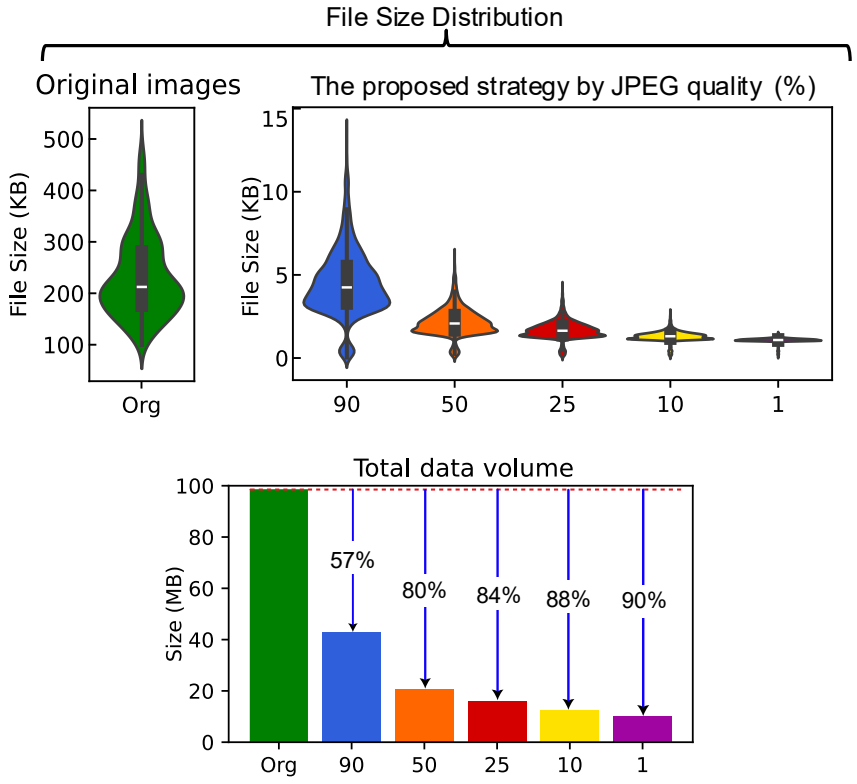


Figure 5.14. Data transmission results. Left: File size distribution of the original images vs. the proposed strategy under different JPEG compression levels. Right: Total data volume comparison between the original images and the proposed strategy for different JPEG compression levels.

images imposed on the communication network to transfer approximately 100 MB of data, whereas the proposed strategy led to a reduction of about 57%, 80%, 84%, 88%, and 90% in total data volume transferring, depending on the JPEG quality level. Figure 5.15 illustrates the impact of different JPEG quality levels on two sample images. It is clear that visible artifacts appear at quality levels of 10 and are even more obvious at level 1, while no significant artifacts are observed at higher quality levels. As a result, by selecting a JPEG quality of 25, instead of transferring a single 212 KB image, the proposed system was required to transfer only 24 cropped images of approximately 1.64 KB each. This makes it feasible to transmit data over low-data-rate networks. In total, only 24 × 1.64 = 39.36 KB needed to be transferred on average, allowing the edge server to efficiently reconstruct the captured image.

5.3.4.4 Execution time and power consumption

In this section, the performance of the proposed framework on the edge nodes and the edge server was evaluated in terms of execution time and energy consumption. In this regard, the edge nodes performance was comprehensively investigated in previous chapter by implementing the SemiY-Net on an MCU-based OpenMV board.

Here, the proposed framework was implemented on a Raspberry Pi 5 board and applied it to the insect dataset to measure execution time and energy consumption. The board was running Raspbian OS in command-line mode to minimize background processes and reduce idle consumption. The average execution time of the proposed framework was 363.6 seconds. Figure 5.16 shows the average execution time of each step of the proposed framework on Raspberry Pi calculated across all subsets of the dataset. Based on the figure, Steps 6 & 7, which handled the classification and retraining of the classifier, required the most time to complete their tasks, approximately 190 seconds per subset, which accounts for about half of the total

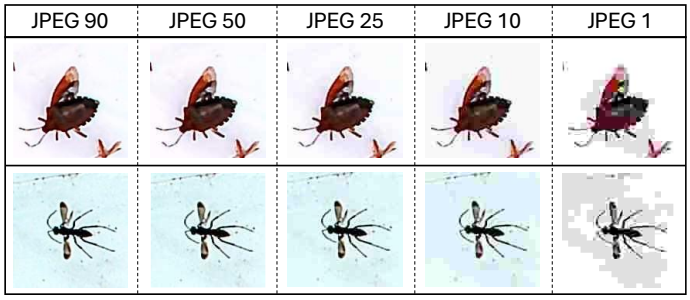


Figure 5.15. Example images illustrating the visual impact of different JPEG compression levels on image quality.

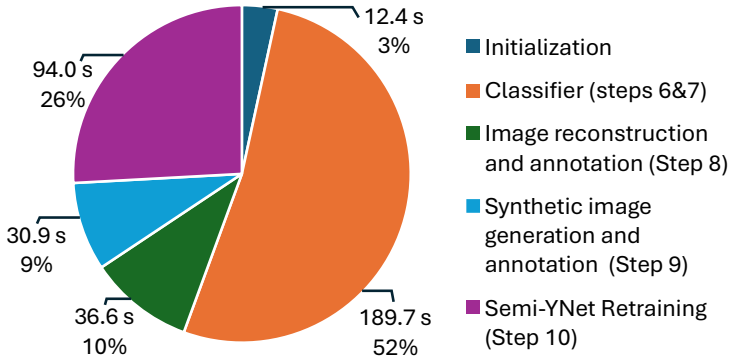


Figure 5.16. Average execution time of each step in the proposed framework when executed on a Raspberry Pi.

execution time. The second most time-consuming task was step 10, which performed SemiY-Net retraining, taking about 94 seconds, which was 26% of the total time. Therefore, it is evident that the retraining was the most time-consuming part of the framework. Moreover, the classifier retraining took longer than the SemiY-Net retraining because it used a triplet loss function. Triplet loss, as a distance metric learning method, is known to be more computationally intensive compared to standard loss functions, since it involves selecting and comparing three samples (anchor, positive, and negative) in the embedding space [14]. Figure 5.17 shows the current consumption of one full operation cycle of the proposed framework on the Raspberry Pi applied to a subset with an execution time close to the average reported above. The current consumption was measured using a Keysight N6705B DC Power Analyzer set to 5.1 V and connected through USB-C input. According to the chart, the highest current consumption, at approximately 1.4 A, was attributed to classification and classifier retraining as well as SemiY-Net finetuning, steps that performed the retraining where the full computational resources of the Raspberry Pi were used and subsequently the most energy was consumed.

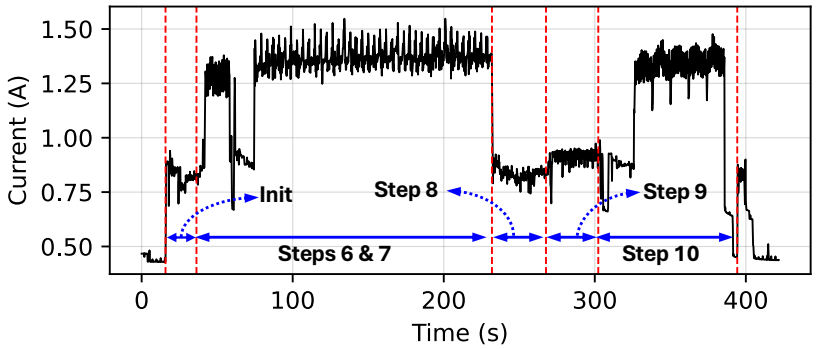


Figure 5.17. Current consumption during one full operation cycle of the proposed framework when executed on a Raspberry Pi.

5.3.5 Discussion

In this study, a self-supervised IL framework was proposed for insect counting and segmentation on resource-constrained IoT devices. This framework autonomously updates the AI model to adapt to possible changes in new data and improve its performance over time after system deployment. This also addresses the fundamental issue of the lack of sufficiently large and representative labeled datasets (i.e., which most AI-powered systems suffer from) by automatically labeling data without human involvement for AI model fine-tuning.

According to the results section, different levels of JPEG quality had only a minor impact on the accuracy of this application. This is attributed to two factors. First, deep networks are remarkably resilient to JPEG compression [16], especially when the model benefits from transfer learning. In such cases, the base model is typically pre-trained on large and diverse datasets like ImageNet, which contain images of varying qualities. Secondly, in this study, the images were downscaled from 1920×1080 px to 320×320 px, which significantly eliminated the noise and artifacts generated during JPEG compression, even at high compression levels such as quality 1. In this regard, Figure 5.18 illustrates examples of cropped images at different qualities, which are then downscaled to simulate the behavior of the proposed system. As shown in the figure, despite visible artifacts, particularly at the lowest quality level of 1, the downscaled images still preserve the structure and shape of the objects (e.g., insects) and appear very similar to those at higher quality levels, such as 90, resulting in nearly the same generated masks. Moreover, unlike large DL models that can extract complex features, tiny and lightweight models, such as SemiY-Net, generally extract

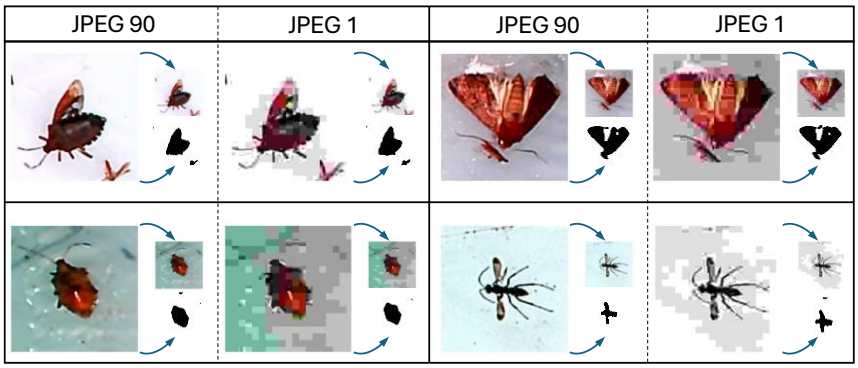


Figure 5.18. Impact of different levels of JPEG compression and image downscaling on the quality of images and the corresponding generated masks.

structural features, such as edges and shape information of the objects, which are less affected by compression artifacts, as explained above.

Furthermore, MobileNetV2 (α=0.35) was selected as the backbone as it provided a balanced trade-off between accuracy and runtime/complexity. Moreover, the proposed classifier itself, which performs classification using the M-K-Means, showed promising performance based on the results and may be applied to other classification tasks (including multi-class cases) that suffer from data shift or a small labeled dataset. However, as mentioned, the triplet loss used, a distance metric learning (DML) method, is computationally intensive and increases execution time and consequently power consumption. By using alternative DML methods or more efficient variants of triplet loss designed to reduce computational complexity [14], it might be possible to further lower execution time and power consumption. This could be a direction for future work.

In the proposed framework, it is suggested to start the retraining based on the number of images received. To make the system more efficient, retraining could instead be started based on the level of shift occurring in newly incoming data. For instance, the shift level could be detected or measured by monitoring the movement of M-K-Means centroids. Moreover, the entire framework could be adapted for other applications and tasks such as segmentation, detection, or classification on IoT-based edge devices. In this study, based on the application, the SemiY-Net model was used and retrained for segmentation and counting. However, the model could be replaced with other architectures for different tasks and depending on the task, some parts of the framework, especially the label generation part, should be modified accordingly.

Regarding data transfer, the proposed framework includes two phases: 1) transferring data from edge nodes to the edge server (step 5), and 2) sending the retrained model from the edge server to the edge nodes (step 10). For the first phase, a strategy was proposed in which only the important areas of an image are transferred and the image is reconstructed later on the edge server. In this way, the edge nodes only need to transfer multiple small images (each 150×150 pixels and approximately 1.6 KB at 25% JPEG quality), making it feasible even via low data-rate networks such as LoRa, as evaluated in [17]. For the second phase, to efficiently transfer the retrained model from the edge server to the nodes, it is proposed to only transfer the updated

parameters rather than the whole model. This was investigated in [18] where the updated parameters are further compressed by the combination of the int8 quantization and Huffman coding. In the case of the SemiY-Net, which has around 750K parameters, only about 380K parameters are retrained (correspond to about 380 KB in INT8 format), leading to a significant reduction (nearly 50%) in the data that needs to be sent. However, even with this reduction, the size is still large to be sent over low data-rate networks such as LoRa. Since model retraining is only needed occasionally (e.g., once every two weeks in this application), it is suggested to use a high data-rate link like Wi-Fi or 4/5G for this part.

The measured average current consumption was approximately 1.2 A at 5.1 V for a duration of about 400 seconds, corresponding to an energy consumption of about 0.68 Wh per cycle. As the cycle occurs only occasionally (e.g., once every two weeks), the overall energy demand is low. For example, a 10,000 mAh Li-ion power bank (37 Wh) can provide around 45 cycles, equal to over 1.5 years of operation. Furthermore, the device can be programmed to be turned on at specific time intervals to reduce standby energy consumption between cycles. With larger capacity batteries or energy-harvesting support (e.g., solar panels), even longer deployments are feasible.

A potential limitation could be in the case of insect overlap. When insects touch or overlap, the blob detection algorithm merges them into a single object. In the current framework, such merged blobs are divided into 150×150 patches and sent to the edge server. Cropping can split insects or create composite objects, and since these cropped patches are classified individually on the edge server, the classifier may misclassify them. As a result, overlapping can lead to classification errors and consequently reduce the quality of the generated masks. This issue could be ameliorated by replacing sticky traps more frequently, especially during peak insect activity. Future work will explore advanced mask generation on the edge server, which has more resources than edge-nodes.

5.4 Conclusion

This chapter completes the third objective by introducing a comprehensive self-improving framework for edge AI in resource-constrained IoT-based systems. The framework is designed to adapt and update the AI model over time using newly captured data. Moreover, it enables efficient data transfer between edge nodes and the

edge server that effectively reduces communication and energy usage overhead. It also eliminates the need for human involvement in labeling and annotation and autonomously updates the model to adapt to changes in data distribution that typically occur in real-world post-deployment scenarios.

With the edge-based device (Chapter 3), the TinyDL model (Chapter 4), and the self-improving framework (this chapter), this work presents a practical solution for real-world insect monitoring on the edge. The final chapter concludes the thesis by summarizing the main findings, discussing the limitations, and providing possible future directions.

5.5 References

[1] J. Liu, X. Yu, and T. Rosing, "Self-Train: Self-Supervised On-Device Training for Post-Deployment Adaptation," *IEEE International Conference on Smart Internet of Things, SmartIoT*, pp. 161–168, 2022.

[2] Z. Yang, S. Al-Dahidi, P. Baraldi, E. Zio, and L. Montelatici, "A Novel Concept Drift Detection Method for Incremental Learning in Nonstationary Environments," IEEE Trans Neural Netw Learn Syst, vol. 31, no. 1, pp. 309–320, Jan. 2020.

[3] S. Zhu, T. Voigt, J. Ko, and F. Rahimian, "On-device Training: A First Overview on Existing Systems," ACM Trans Sens Netw, vol. 0, no. 6, p. 39, Dec. 2022.

[4] J. Gui et al., "A Survey on Self-supervised Learning: Algorithms, Applications, and Future Trends," IEEE Trans Pattern Anal Mach Intell, 2024.

[5] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 4510–4520, Dec. 2018, doi: 10.1109/CVPR.2018.00474.

[6] "Kaggle Cats and Dogs Dataset from Official Microsoft Download Center." Accessed: Feb. 25, 2025. [Online]. Available: https://www.microsoft.com/en-us/download/details.aspx?id=54765

[7] A. Kargar, D. Zorbas, S. Tedesco, M. Gaffney, and B. O'Flynn, "Image Dataset of Brown Marmorated Stink Bug (BMSB) or Halyomorpha Halys (HH)," 2024, Zenodo. doi: 10.5281/zenodo.7887045.

[8] F. Pedregosa FABIANPEDREGOSA et al., "Scikit-learn: Machine Learning in Python," The Journal of Machine Learning Research, vol. 12, pp. 2825–2830, Nov. 2011, doi: 10.5555/1953048.2078195.

[9] I. T. Jollife and J. Cadima, "Principal component analysis: a review and recent developments," Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, vol. 374, no. 2065, Apr. 2016.

[10] "OpenMV Cam H7 Plus." Accessed: Jun. 23, 2025. [Online]. Available: https://openmv.io/products/openmv-cam-h7-plus

[11] T. Chen, D. Eager, and D. Makaroff, "Efficient Image Transmission Using LoRa Technology In Agricultural Monitoring IoT Systems," in International Conference on Internet of Things and IEEE Green Computing and Communications and IEEE Cyber, Physical and Social Computing and IEEE Smart Data, 2019, pp. 937–944.

[12] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), vol. 07-12-June-2015, pp. 815–823, Jun. 2015.

[13] T.-T. Do, T. Tran, I. Reid, V. Kumar, T. Hoang, and G. Carneiro, "A Theoretically Sound Upper Bound on the Triplet Loss for Improving the Efficiency of Deep Distance Metric Learning," in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 10404–10413.

[14] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLOv8," 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[15] J. Azar, A. Makhoul, M. Barhamgi, and R. Couturier, “An energy efficient IoT data compression approach for edge machine learning,” Future Generation Computer Systems, vol. 96, pp. 168–175, Jul. 2019.

[16] S. Dodge and L. Karam, “Understanding how image quality affects deep neural networks,” 8th International Conference on Quality of Multimedia Experience, QoMEX, Jun. 2016.

[17] L. J. Zurek, A. Kargar, B. O’Flynn, D. Niederprüm, L. Wolf, and D. Zorbas, “Evaluation of Wireless Technologies for an Embedded Camera-based Pest Monitoring System,” IEEE International Workshop on Metrology for Agriculture and Forestry, MetroAgriFor, pp. 449–454, 2023.

[18] R. Khamitov, A. Kargar, M. Azamat, M. Ristin, B. O’Flynn, and D. Zorbas, “Incremental Learning of Image Classification Models over Low Data-Rate Networks,” in 21st Annual International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT 2025), Lucca, Italy, 2025.

6 CHAPTER VI. DISCUSSION

6.1 Introduction

This chapter concludes the work presented in this thesis by summarizing the key contributions, acknowledging the limitations, and discussing directions for future work. The goal of this research was to address identified gaps in current scientific knowledge, and to develop a complete, low-power insect monitoring system that works reliably in real-world conditions, especially in remote orchards where internet connectivity and power are limited. Through the design of an edge prototype device, a lightweight optimized DL model, and a self-adaptive learning framework, this thesis makes AI edge-based monitoring systems more practical and efficient for real-world deployment in remote areas. In the following, the key contributions are first summarized, followed by a discussion of the current work limitations and potential areas for improvement.

6.2 Summary of thesis contributions

6.2.1 Chapter 2: Gaps, objectives, and hypotheses identification

This chapter established insect monitoring as a realistic and demanding application for deploying DL-based IoT systems in remote agricultural locations. It first identified the key limitations and considerations for the application, reviewed current approaches and highlighted important gaps in literature, and finally defined the objectives and hypotheses accordingly.

Based on the findings, several interconnected challenges were identified that this research aimed to address, as follows:

- *Hardware and infrastructure limitations:* Existing AI-powered solutions mostly rely on cloud or server-based infrastructure. While these solutions achieve high accuracy, they require stable internet connectivity and high-power consumption, neither of which is available in remote orchards. Edge-based systems eliminate the need for internet access but have strict computational, memory, and power

constraints which make the deployment of conventional DL models impractical. The limited number of studies in the literature that focus on such systems primarily aimed to achieve high accuracy on these devices, which often overlook critical real-world issues.

- *Data scarcity and annotation challenges:* DL models perform well when trained on large, diverse and annotated datasets. Insects are the most diverse animal group on Earth, which makes dataset creation particularly difficult. A seasonal pattern of insect presence in orchards further limits data collection. Besides, manual labeling is time-consuming, difficult and expensive.
- *Data drift:* Orchard environments are dynamic, meaning that conditions change over time due to several factors, such as seasonality, environmental factors or sensor aging. These changes lead to data drift, which causes significant performance reduction of DL models over time.

Based on the above findings, the main objectives of this thesis were defined as follows:

- To develop a low-power edge-based prototype device for insect detection and identification that can capture and analyze insect trap images and transmit required data using a resource-constrained microcontroller unit (MCU), enabling reliable operation in remote agricultural environments with limited power and connectivity.
- To develop and validate a novel TinyDL model designed for deployment on a resource-constrained MCU for automated insect monitoring, capable of performing insect segmentation and counting under strict memory and computational constraints.
- To design and validate a practical and comprehensive edge-based insect monitoring framework that operates reliably under real-world agricultural constraints to address fundamental deep-learning (DL) challenges, including the lack of annotated datasets and post-deployment data drift.

Finally, based on these objectives, a set of research hypotheses was formulated to evaluate the expected outcomes as follows:

- It is hypothesised that a functional low-power insect monitoring device based on MCU hardware can be developed that operates within strict energy and memory constraints, while reliably performing image acquisition and lightweight inference locally in remote orchards without cloud or internet connectivity.
- It is hypothesised that a TinyDL model specifically optimized for insect identification and counting can achieve efficient execution on resource-constrained MCUs (in terms of model size, memory usage, inference time, and energy consumption) while maintaining acceptable performance in terms of accuracy and robustness for insect segmentation and counting tasks.
- It is hypothesised that a self-learning edge-based framework based on pseudo-labeling and incremental learning can effectively mitigate the effects of limited training data and post-deployment data drift, without requiring manual annotation or cloud-based retraining while maintaining stable performance in terms of accuracy and robustness.

To meet these objectives, this research was carried out in three main phases that form a framework for designing, optimizing, and deploying an AI-based insect monitoring system on low-power edge devices. Each technical chapter (i.e., chapters 3, 4, and 5) corresponds to one phase and is related to a separate research hypothesis.

6.2.2 Chapter 3: Feasibility of edge AI on MCU

The first challenge undertaken was to demonstrate whether MCU-based devices, with their extremely limited computational and memory resources, could support on-device insect monitoring in real-world conditions. Unlike cloud- or server-based approaches, MCUs typically provide only a few megabytes of RAM and storage, but they offer low power consumption and cost advantages, making them suitable for IoT deployments in remote agricultural environments.

To address this, a low-power monitoring device based on an STM32H7 (OpenMV board) was developed. The novelty in this approach lay in the co-design of both hardware and AI algorithms to make real-time insect detection feasible under extreme constraints. On-device pre-processing steps were first implemented, including grayscale conversion, adaptive thresholding and RoIs extraction. Then, a small DL-based classifier (≤200 KB model size) was deployed to detect the presence of insects

on these cropped RoIs. By cropping 150×150 RoIs from the original 1600×1200 images, the data volume fed to the DL model was reduced by ~99% before inference, making the model much faster and lighter to run.

This prototype was deployed in an infested orchard in Italy, where it successfully operated fully on-device without cloud connectivity and was able to process two images in ~30 seconds while consuming less than 300 mA of current in active mode. The device operates in active mode twice a day and remains in deep sleep mode for the rest of the time. To the best of the author's knowledge, this was among the earliest MCU-based insect detection systems that combined hardware-software co-design with lightweight design and reliable performance and was one of the few validated through real orchard deployment under uncontrolled outdoor conditions beyond prior lab-only studies. Importantly, the deployment also generated a dataset of Brown Marmorated Stink Bug (BMSB) images in real-world conditions that provided the foundation for Phase 2 model development.

6.2.3 Chapter 4: Lightweight DL model for segmentation and counting

The next task was to design a TinyDL model that provides detailed results, including segmentation and counting output, while remaining deployable on MCUs.

To address this, a novel architecture (Y-Net) was first proposed combining an encoder-decoder backbone for segmentation with an additional regression path for insect counting. This enables the model to generate both pixel-level masks and global insect counts providing more information than classification-based approaches. The novelty of Y-Net lies in its lightweight dual-task design, which has not been previously applied for segmentation and counting in this application. This model achieved a Dice Similarity of ~91.5% and a counting MAE of ~1, with an inference time of nearly 0.4 seconds on an Android smartphone.

However, Y-Net in its original form was not deployable on MCUs. Therefore, SemiY-Net was proposed, which is a memory-optimized model for ≤1 MB RAM MCU-based devices. The optimizations included:

- Layer-level memory profiling to identify and modify high memory usage ones.
- Structured pruning of convolutional filters.
- Selective removal of early skip connections.

- 8-bit quantization via TensorFlow Lite Micro.

In contrast to conventional segmentation models, SemiY-Net deliberately reduced output spatial resolution (down to 1/16 of the input) and simplified both the encoder and decoder paths to reduce and optimize memory usage. This design represents a deliberate trade-off by sacrificing some accuracy to enable deployment on MCUs.

Quantitatively, SemiY-Net reduced parameters by >75% compared to YOLO-v8, U-Net, and DeepLabv3+, required <1 MB of storage and RAM, and achieved ~5 J per inference on an OpenMV board. On the field dataset collected in Phase 1, it achieved Dice Similarity ~85% and counting MAE ~1.32. This was one of the first segmentation and counting models designed specifically for MCU, going beyond the state-of-the-art where existing approaches depended on cloud or server.

6.2.4 Chapter 5: Self-improving edge AI architecture

Despite the success of SemiY-Net, a major practical challenge remained: collecting and labelling a large and diverse dataset to train the model effectively, especially under varying environmental conditions that cause data shift. To overcome this, Phase 3 introduced a novel self-improving edge AI architecture that enables autonomous learning and adaptation after deployment. The system is composed of multiple MCU-based edge nodes connected to an edge server (e.g., Raspberry Pi) over low data-rate link, such as LoRa.

The MCUs perform real-time inference using the proposed SemiY-Net and transmit only compressed RoIs to the edge server, reducing communication overhead by ~84%. The edge server then reconstructs the raw captured images from the received RoIs, then generates the corresponding masks, and finally fine-tunes SemiY-Net to adapt to new field conditions and data distributions. For this purpose, contrastive feature extraction using triplet loss was proposed and implemented for embedding learning, followed by a modified K-Means clustering approach to assign unlabeled RoI images into groups for pseudo-labeling. To improve pseudo-label reliability, only high-confidence cluster assignments were selected and used for generating pseudo-labels. In addition to the reconstructions, artificial images were also synthesized from the RoIs to increase the training sample diversity. These reconstructed and augmented

samples are then used in periodic incremental learning cycles to fine-tune SemiY-Net with the new data. Finally, updated model weights are sent to the edge nodes.

This system enables continuous adaptation without human intervention. Quantitatively, models fine-tuned with pseudo-labels achieved accuracy comparable to those trained with expert annotations, while inference energy consumption on MCUs remained ~5 J and communication energy consumption decreased >80% compared to raw image transfer.

This is one of the first complete lifelong learning architectures designed for resource-constrained IoT networks. Unlike prior work that relies on offline retraining or manual annotation, this solution demonstrates fully autonomous adaptation to data drift at the edge, making it suitable for large-scale agricultural deployments.

Together, these three phases form a practical, scalable and efficient AI-based IoT system for insect monitoring. The system covers the entire pipeline, from hardware design to model development and deployment. By eliminating reliance on cloud and human annotation, it delivers intelligent automation directly to the orchard, enabling reliable and long-term insect monitoring even in resource-constrained environments.

6.2.5 Publications

The works in this thesis were adapted from the first author peer-reviewed articles as follows. In chapter 3, Section 3.2 was primarily derived from [1], Section 3.3 from [2], and Section 3.4 from [3] (only Section V). In chapter 4, Section 4.2 was primarily derived from [4], and Section 4.3 from article [5]. Finally, in chapter 5, Section 5.2 was primarily derived from article [6], and Section 5.3 from article [7].

The remaining publications [3][8][9][10][11][12] cover other aspects of the project not covered in this thesis.

6.2.6 Open data

Dataset collection and creation are time-consuming, expensive and tedious, especially the annotation stage [13]. There are a lot of insect species in the world [14], which makes dataset creation particularly challenging. It becomes even more difficult for insects, as they are small and appear only seasonally, which makes data collection

impossible throughout the year. Thus, sharing data with other researchers and the scientific community is an important aspect of open data principles. Therefore, all the datasets collected and used in this thesis (chapters 3, 4, and 5) are available online on Zenodo as an open-access resource, meeting the FAIR data principles of findability, accessibility, interoperability and, reusability.

> **Kargar, A.**, Zorbas, D., Gaffney, M., O'Flynn, B., & Tedesco, S. (2024). Image Dataset of Brown Marmorated Stink Bug (BMSB) or Halyomorpha Halys (HH) [Data set]. Zenodo.
>
> - Version 1: https://zenodo.org/records/7887046
> - Version 2: https://zenodo.org/records/14051319
> - Version 3: https://zenodo.org/records/16088064

6.3 Limitations, recommendations and future work

As mentioned in the discussion sections of each chapter, there are several areas that can be further explored. Therefore, this section provides recommendations for future work based on the insights obtained from this research. In the following, each phase of the thesis is discussed separately and the potential areas for further improvement and exploration are outlined.

6.3.1 Edge device

In terms of the prototype device, future work can focus on improving the device from different aspects to enhance its performance and efficiency in real-world deployments.

As mentioned in chapter 3, the device power consumption in sleep-mode has a higher impact on the battery lifetime compared to the operational mode. The device is in active mode for only a couple of minutes with higher current consumption (255 mA), while the current consumption in sleep-mode is 7 mA (~36x lower) which is still critical and strongly affects the battery lifetime. Therefore, reducing the sleep-mode current consumption is important; as an example, a reduction of 1 mA can significantly extend the lifetime from 29 to 196 days. This can be achieved at hardware level by disabling all peripherals and components (e.g., camera), ensuring that the MCU enters its lowest-power deep-sleep mode, or by using an ultra-low-power external MCU to

control a power switch that completely shuts down the device. Hence, designing the device to enter a complete deep-sleep mode is crucial and can be further explored in future work.

Moreover, the prototype device was programmed to wake up at fixed intervals. However, a dynamic wake-up approach based on insect activity or density trends could improve energy efficiency by reducing unnecessary operation during periods of low activity (e.g., wintertime) [15]. This would also provide more frequent operation during the high season, which provides better and more real-time information about the orchard conditions for growers for decision-making.

Another important area of improvement is the trap mechanical design, which can impact the insect capture efficiency [16] maintenance needs and the device performance. The suggested device in chapter 3 has a rotating part that mechanically rotates the trap. Despite its advantages mentioned in that chapter, this mechanical part also increases power consumption and can malfunction in harsh orchard environments over time. In addition, the deployment revealed that the camera in such setups is exposed to rain and wind and can easily be affected by dust and dirt over time, which reduces image quality. By comparison, the data used in chapter 5 was collected using a box-shaped design. In this design, the camera is placed inside a box at the top and the sticky trap at the bottom, with openings or windows on the sides to attract insects using pheromones. This design significantly helps shield the camera from environmental factors and simplifies the system by removing the rotating part. Therefore, the design of the trap is important as it impacts both insect capture efficiency and camera performance and can be further investigated.

The communication aspect is another important component of the device that can be improved. The current consumption results reported in Chapter 4 were for a device which used Wi-Fi for data transmission. Based on the results, data transfer is the second-highest source of energy consumption in the system, accounting for about a third of one operational cycle. In another study, it was suggested to use low-data-rate communication protocols, such as LoRa, for data transfer. Despite its promising results for transferring small cropped images (150×150 pixels), this is not suitable for transmitting the retrained model from the edge server to the edge node. Therefore, using a high-data-rate link was suggested for this part, as this update occurs only

occasionally (e.g., once every two weeks in this case). Overall, the communication part is a critical component that can be further explored to better align with the application requirements and system limitations.

6.3.2 TinyDL model

Regarding the proposed DL model, future work can focus on improving its accuracy and expanding it to other scenarios.

In terms of the target insect, this work focused on *Halyomorpha halys*, which made the task a binary problem. In this regard, future extensions could include different life stages (e.g., nymphs and adults) and other insect species, which turns it into a multiclass task to increase the generalization of the model and the system.

In the model optimization, the peak memory usage is limited by the available RAM of typical MCUs (around 1 MB) while some layers use significantly less memory. Future work could investigate using more of the available RAM by such layers (e.g., by increasing the number of feature map channels) to achieve higher accuracy. However, this could also increase computation time and power consumption, which should be considered. In addition to this, the model input size is 320×320 pixels which might be small for detecting smaller insects, since downsampling operations can remove or alter important features. As suggested in chapter 4, a potential solution would be image tiling instead of downsampling. However, this leads to applying the model on several tiles which increase the processing time and consequently power consumption, again highlighting the accuracy vs power trade-off in such systems. Besides, using larger input images could increase the model accuracy, but it would also increase memory usage, computation time, and consequently power consumption. To address this, a Neural Architecture Search (NAS) [17] technique can be used to find a proper model structure that balances accuracy, RAM usage, processing time, and energy consumption, which could be considered a potential area for future work.

In this work, a large Y-Net model was first proposed and then optimized using several modifications to fit within resource-constraints MCU. Therefore, as a direction for future work, the idea of using a knowledge distillation method [18] can be explored

where the larger Y-Net acts as the teacher and SemiY-Net acts as the student during training to improve accuracy.

Moreover, model parameters were converted to int8 based on the post-training quantization provided by TensorFlow. This area can be further investigated by using different quantization techniques or by applying quantization-aware training [19] to achieve higher accuracy and compression rates.

Finally, to achieve more reliable results, the model can provide confidence scores with the output results, which give additional information to help the user make more informed decisions.

6.3.3 Edge framework

Regarding the final part, the edge framework, future work can focus on improving the framework performance from different aspects as follows.

RoI detection is an important part of this framework, which is also used in the image analysis part of Chapter 3. The current RoI detection relies on conventional and static image processing algorithms, such as image thresholding and blob detection. This is a simple and efficient method to run on resource-constrained MCUs but can be inaccurate in varying lighting and with complex backgrounds. This was the main reason to control the environment by capturing images at night using LED lighting and a uniform white background to have reliable RoIs. In this regard, future work could explore more complex and reliable lightweight DL-based segmentation/ detection models to extract these RoIs. However, this could increase complexity and power consumption and would need a high level of compression and optimization to fit on MCUs. It could also be an important step to increase the scalability of the whole approach for other applications beyond insect monitoring.

In the proposed framework, JPEG compression was suggested to significantly reduce the data transmission volume and enhance communication performance, especially by reducing power consumption, while maintaining approximately the same level of accuracy. Therefore, other image compression methods [20] can be considered and investigated in future research to further reduce the data transmission volume. Moreover, to further improve the communication part, a mobile device, such as a drone

that flies over the orchard, can be integrated into the framework to enhance both data collection and model transfer between edge nodes and the edge server.

In the pseudo-labeling phase, the upper whisker, calculated using labeled data, was selected as a threshold to filter the samples with lower uncertainty. To further improve this part, future work can explore an adaptive upper whisker that updates based on the newly incoming data to improve the quality of the selected samples used for model retraining in the next stages.

Moreover, in the synthetic image generation step, the algorithm randomly selects from all cropped images to generate new trap images. To improve model retraining, the algorithm can choose more challenging samples to help the model learn better from them. In addition, in this step, a fixed number of synthetic images is created to be used during the retraining phase. A dynamic number of synthetic images, instead of a fixed number, can help create a balance between real and synthetic images and improve retraining performance. This approach can be considered for future work.

Results in Chapter 5 revealed that the used triplet-loss function, a DML-based loss, is computationally expensive, resulting in an increase in training time and consequently power consumption. As also mentioned in chapter 5, a lighter DML loss function [21] could be used instead of the current one to significantly reduce the retraining time and the associated power consumption.

Finally, one of the important parts is the starting time of the retraining steps on the edge server. In the current implementation, a fixed periodic strategy was suggested. A potential improvement could be a drift detection mechanism [22] to evaluate the level of drift and based on that, decides whether to start the retraining or not. This adaptive retraining strategy could help the system to update the DL model more efficiently, only when needed. This could be another potential area for further research.

6.3.4 Additional recommendations

Another area for further work is the insect overlapping issue, which can lead to errors in both the proposed TinyDL model and reduce the quality of the generated masks in the retraining steps of the framework. To mitigate this issue, practical measures such as replacing sticky traps more frequently, especially during peak insect activity, were suggested to reduce the likelihood of overlaps. From a technical

perspective, future work could explore more advanced segmentation models and mask generation approaches, such as visual attention mechanisms or instance-level segmentation methods, to improve robustness when overlaps occur.

For the entire system, future work could also focus on enhancing the system capability for forecasting and recommendation. In this regard, in addition to the results from visual analysis, other environmental parameters, such as temperature, humidity, rainfall, and wind speed, can be used together to create a model for insect population prediction [23]. This can provide future forecasts that can be used for early warnings and to develop a recommendation system to help growers to make timely decisions.

6.4 Research Impact

According to a report by FAO, up to 40% of global crop production is lost to plant pests and diseases annually, causing over $220 billion in economic losses, along with invasive insects causing at least $70 billion [24]. The proposed edge AI-based insect monitoring system provides a practical solution to improve crop production and strengthen food security while significantly reducing the above-mentioned losses.

Automated insect monitoring is an important part of Precision Agriculture (PA) that optimizes insect population estimation and minimizes manual inspections. Growers currently spend considerable time, money, and resources on field monitoring. By reducing in-person inspections, the system lowers operational costs, fuel consumption and emissions, leading to more efficient farm management. Reports show smart traps can reduce insecticide usage by over 35% [25], save >36,000 km travel and 1,170 labor hours and improve spray efficiency by over 30% [26].

The system also directly supports the European Union ambition, which targets a 50% reduction in pesticide use by 2030 [27]. Early detection helps growers to reduce chemical pesticide usage, which supports both environmental protection and safer food production.

Another major issue is the spread of invasive insects, which has increased recently due to climate change and international trade. The system can be deployed not only in orchards but also in forestry, rural landscapes and at borders to provide automated early warnings. Such detection allows authorities to take timely action

before invasive species establish themselves and avoid large-scale ecological and economic damage.

The project also addresses the rural-AI gap. Many remote orchards lack stable internet or proper computing infrastructure; however, this edge-based solution provides AI directly in the field. Using such affordable edge-based monitoring systems ensures technological equity between urban and rural growers, which improves local food production and supports rural economies.

In summary, this project secures food supplies, reduces economic losses, empowers rural communities and protects the environment. Most importantly, it demonstrates how AI, optimized for the edge, can provide practical and scalable solutions that transform agriculture and beyond to benefit both society and the environment.

6.5 Achievements and Recognition

The research was supported through a prestigious Walsh Scholarship, awarded only to candidates conducting nationally significant agri-tech research, thereby recognizing its potential impact in smart agriculture and AI-driven pest monitoring.

Furthermore, one of the resulting papers was named Best Paper in the session Image Processing and Biological Applications at the 2022 IEEE 5th International Conference on Image Processing Applications and Systems (IPAS), Genova, Italy.

Moreover, the developed device and the project were also featured in the television program "10 Things to Know About[12]" on RTE and will be broadcast in the next episode of this series. Moreover, the work received national recognition through coverage on RTÉ News[13,14].

Finally, and most notably, the work was awarded the "Best Application of AI in a Student Project" at the 2025 AI Awards Ireland[15].

Overall, this thesis brings together the key elements of an AI-powered, edge-based insect monitoring system designed for real-world environments. The work

[12] https://10thingstoknowabout.ie/
[13] https://www.tyndall.ie/news/rte-news-highlights-tyndalls-ai-research-to-protect-crops/
[14] https://www.rte.ie/news/ireland/2025/1117/1544305-irish-ai-climate/
[15] https://aiireland.ie/2025/11/18/ai-awards-celebrates-irish-innovation-in-2025/

demonstrates that low-power hardware can support practical vision-based insect detection and monitoring under field conditions. A lightweight segmentation and counting model, SemiY-Net, was developed to run on MCU-based devices while maintaining acceptable performance. Building on this, an edge-edge collaborative framework was introduced, where MCU-based nodes capture images and perform inference, and an edge server (e.g., Raspberry Pi) performs pseudo-labeling and incremental learning locally to adapt and update the model over time without cloud connectivity or manual annotation. Together, these components demonstrate that reliable, scalable, and low-cost insect monitoring is feasible directly in orchards, despite real-world challenges such as limited resources, environmental variability, and data scarcity.

6.6 References

[1] A. Kargar, M. P. Wilk, D. Zorbas, M. T. Gaffney, and B. Q'Flynn, "A Novel Resource-Constrained Insect Monitoring System based on Machine Vision with Edge AI," 5th IEEE International Image Processing, Applications and Systems Conference, IPAS 2022, 2022, doi: 10.1109/IPAS55744.2022.10052895.

[2] A. Kargar, D. Zorbas, S. Tedesco, M. Gaffney, and B. O'Flynn, "Detecting Halyomorpha halys using a low-power edge-based monitoring system," Comput Electron Agric, vol. 221, p. 108935, Jun. 2024.

[3] L. Almstedt et al., "Beyond the Naked Eye: Computer Vision for Detecting Brown Marmorated Stink Bug and Its Punctures," IEEE Transactions on AgriFood Electronics, pp. 1–12, Aug. 2024.

[4] A. Kargar, D. Zorbas, M. Gaffney, B. O'Flynn, and S. Tedesco, "Y-Net: Insect Counting and Segmentation using Deep Learning on Embedded Devices," Conference Record - IEEE Instrumentation and Measurement Technology Conference, 2024, doi: 10.1109/I2MTC60896.2024.10561007.

[5] A. Kargar, D. Zorbas, M. Gaffney, B. O'Flynn, and S. Tedesco, "Tiny deep learning model for insect segmentation and counting on resource-constrained devices," Comput Electron Agric, vol. 236, p. 110378, Sep. 2025.

[6] A. Kargar, D. Zorbas, M. Gaffney, B. O'Flynn, and S. Tedesco, "Concept Drift Mitigation on Resource-Constrained IoT Devices via Self-Learning," 2025 IEEE Sensors Applications Symposium (SAS), pp. 1–6, Jul. 2025, doi: 10.1109/SAS65169.2025.11105109.

[7] A. Kargar, D. Zorbas, M. Gaffney, B. O'Flynn, and S. Tedesco, "Self-Supervised Incremental Learning for Insect Segmentation and Counting on Resource-Constrained Devices," IEEE Internet Things J, pp. 1–1, 2025, doi: 10.1109/JIOT.2025.3630062.

[8] L. Almstedt et al., "A Comprehensive Pest Monitoring System for Brown Marmorated Stink Bug," IEEE Transactions on AgriFood Electronics, vol. 3, no. 1, pp. 110–120, Oct. 2024, doi: 10.1109/TAFE.2024.3469538.

[9] R. Khamitov, A. Kargar, M. Azamat, M. Ristin, B. O'Flynn, and D. Zorbas, "Incremental Learning of Image Classification Models over Low Data-Rate Networks," in 21st Annual International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT 2025), Lucca, Italy, 2025.

[10] T. Zarymkanov, A. Kargar, C. M. Pinotti, B. O'Flynn, and D. Zorbas, "Enhancing Machine Learning Training Performance in Smart Agriculture Datasets Using a Mobile App," 2023 IEEE International Workshop on Metrology for Agriculture and Forestry, MetroAgriFor 2023 - Proceedings, pp. 455–460, 2023, doi: 10.1109/METROAGRIFOR58484.2023.10424407.

[11] L. J. Zurek, A. Kargar, B. O'Flynn, D. Niederprüm, L. Wolf, and D. Zorbas, "Evaluation of Wireless Technologies for an Embedded Camera-based Pest Monitoring System," 2023 IEEE International Workshop on Metrology for Agriculture and Forestry, MetroAgriFor 2023 - Proceedings, pp. 449–454, 2023, doi: 10.1109/METROAGRIFOR58484.2023.10424396.

[12] L. Almstedt et al., "Technological Innovations in Agriculture for Scouting Halyomorpha Halys in Orchards," Proceedings - 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things, DCOSS-IoT 2023, pp. 702–709, 2023, doi: 10.1109/DCOSS-IOT58021.2023.00110.

[13] Y. Ouali, C. Hudelot, and M. Tami, “An Overview of Deep Semi-Supervised Learning,” Jun. 2020, Accessed: Jul. 30, 2025. [Online]. Available: https://arxiv.org/pdf/2006.05278

[14] W. Li, T. Zheng, Z. Yang, M. Li, C. Sun, and X. Yang, “Classification and detection of insects from field images using deep learning for smart pest management: A systematic review,” Ecol Inform, vol. 66, p. 101460, Dec. 2021, doi: 10.1016/J.ECOINF.2021.101460.

[15] L. Palazzetti, F. Betti Sorbelli, C. M. Pinotti, and others, “Let Me Sleep! A Machine Learning Approach for IoT Wake-Up Mechanism for Insect Detection,” in IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor), 2025.

[16] M. Preti, F. Verheggen, and S. Angeli, “Insect pest monitoring with camera-equipped traps: strengths and limitations,” J Pest Sci (2004), vol. 94, no. 2, pp. 203–217, Mar. 2021, doi: 10.1007/S10340-020-01309-4/TABLES/2.

[17] T. Elsken, J. H. Metzen, and F. Hutter, “Neural Architecture Search: A Survey,” Journal of Machine Learning Research, vol. 20, no. 55, pp. 1–21, 2019, Accessed: Oct. 17, 2025. [Online]. Available: http://jmlr.org/papers/v20/18-598.html

[18] L. Wang and K. J. Yoon, “Knowledge Distillation and Student-Teacher Learning for Visual Intelligence: A Review and New Outlooks,” IEEE Trans Pattern Anal Mach Intell, vol. 44, no. 6, pp. 3048–3068, Jun. 2022, doi: 10.1109/TPAMI.2021.3055564.

[19] B. Rokh, A. Azarpeyvand, and A. Khanteymoori, “A Comprehensive Survey on Model Quantization for Deep Neural Networks in Image Classification,” ACM Trans Intell Syst Technol, vol. 14, no. 6, p. 97, Nov. 2023, doi: 10.1145/3623402.

[20] A. J. Hussain, A. Al-Fayadh, and N. Radi, “Image compression techniques: A survey in lossless and lossy algorithms,” Neurocomputing, vol. 300, pp. 44–69, Jul. 2018, doi: 10.1016/J.NEUCOM.2018.02.094.

[21] T.-T. Do, T. Tran, I. Reid, V. Kumar, T. Hoang, and G. Carneiro, “A Theoretically Sound Upper Bound on the Triplet Loss for Improving the Efficiency of Deep Distance Metric Learning,” in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 10404–10413.

[22] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, “Learning under Concept Drift: A Review,” IEEE Trans Knowl Data Eng, vol. 31, no. 12, pp. 2346–2363, Dec. 2019, doi: 10.1109/TKDE.2018.2876857.

[23] G. R. Palma et al., “Forecasting insect abundance using time series embedding and machine learning,” Ecol Inform, vol. 85, p. 102934, Mar. 2025, doi: 10.1016/J.ECOINF.2024.102934.

[24] “About | Plant Production and Protection | Food and Agriculture Organization of the United Nations.” Accessed: Oct. 15, 2025. [Online]. Available: https://www.fao.org/plant-production-protection/about/en

[25] M. Á. Miranda et al., “Developing and Implementation of Decision Support System (DSS) for the Control of Olive Fruit Fly, Bactrocera Oleae, in Mediterranean Olive Orchards,” Agronomy 2019, Vol. 9, Page 620, vol. 9, no. 10, p. 620, Oct. 2019, doi: 10.3390/AGRONOMY9100620.

[26] “Trapview - Automated pest-monitoring system for sustainable growing with optimal insecticide use,” Sep. 2016, doi: 10.3030/733979.

[27] “Farm to Fork Strategy - Food Safety - European Commission.” Accessed: Oct. 15, 2025. [Online]. Available: https://food.ec.europa.eu/horizontal-topics/farm-fork-strategy_en

APPENDIX A: EVALUATION METRICS

To ensure consistency across the experimental evaluations presented in this thesis, the performance metrics used in the following technical chapters are defined in this section.

A.1 Classification metrics

For the classification task, four different metrics, accuracy, precision (P), recall (R), and F1-score, were used. The accuracy represents the correct predictions over the total number of predictions. Precision indicates the proportion of correct positive predictions and the total positive predictions, while recall indicates the proportion of correct positive predictions and actual positives. The F1-score shows the balance between precision and recall metrics for a model. These metrics were chosen as they are commonly used in binary classification models and are calculated as shown in Eqs. (A.1) - (A.4):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{A.1}$$

$$P = \frac{TP}{TP + FP} \tag{A.2}$$

$$R = \frac{TP}{TP + FN} \tag{A.3}$$

$$F1 = \frac{2 \times P \times R}{P + R} = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{A.4}$$

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives.

A.2 Segmentation metrics

The segmentation output, which creates a black-and-white image highlighting the target insect on the image, was assessed using four different metrics: Intersection over Union (IoU), Dice Similarity Coefficient (DSC), precision, and recall. IoU and DSC values measure the ratio of the overlap and similarity of the predicted mask and the actual mask; IoU penalizes partial overlaps more than DSC, meaning that it is more sensitive to small mismatches.

Precision (P) and Recall (R) measure the accuracy of the model in classifying the pixels as black or white, considering their respective classes. Precision focuses on the ability of a model to minimize false positives by measuring the ratio of correct predicted positives (white pixels) to all predicted positives. Recall focuses on the ability of the model to minimize false negatives by calculating the ratio of correct positives to total actual positives. These metrics are calculated as shown in Eqs. (A.5) - (A.8):

$$P = \frac{TP}{TP + FP} \tag{A.5}$$

$$R = \frac{TP}{TP + FN} \tag{A.6}$$

$$IoU = \frac{P \times R}{P + R - P \times R} = \frac{TP}{TP + FP + FN} \tag{A.7}$$

$$DSC = \frac{2 \times P \times R}{P + R} = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{A.8}$$

where TP is the number of true positives, FP is the number of false positives, and FN is the number of false negatives.

A.3 Counting metrics

The counting task was evaluated by Mean Squared Error (MSE) and Mean Absolute Error (MAE). MSE measures the mean squared difference between the number of predicted insects in the input image and the actual one. MAE evaluates the counting performance by computing the mean absolute difference between the predicted and actual number of insects. These errors are calculated as shown in Eqs. (A.9) - (A.10):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{A.9}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{A.10}$$

where n is the number of input images, y_i is the correct value, $\hat{y}_i$ is the predicted value.

In addition, R^2 is also considered for counting evaluation. R^2 measures the degree of how well the predicted insect number matches the actual insect number. This metric is calculated as shown in Eq. (A.11):

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{A.11}$$

where n is the number of input images, y_i is the correct value, $\hat{y}_i$ is the predicted value and $\bar{y}$ is the mean of the correct value.

A.4 Incremental learning (IL) metrics

Incremental learning performance was evaluated using standard metrics, including Average Incremental Accuracy (AIA), Backward Transfer (BWT), and Forward Transfer (FWT). AIA shows the overall performance trend by averaging accuracy after each subset update, giving an indication of how well the model maintains performance as new data is added. BWT measures stability, showing whether training on later subsets helps or harms earlier ones, negative values indicate catastrophic forgetting. FWT reflects plasticity, the ability to generalize knowledge from earlier subsets to unseen ones. Positive values indicate helpful forward transfer, and negative values show interference. These metrics are calculated as shown in Eqs. (A.12) - (A.15):

$$AA_k = \frac{1}{k}\sum_{j=1}^{k} a_{k,j} \tag{A.12}$$

$$AIA_k = \frac{1}{k}\sum_{i=1}^{k} AA_i \tag{A.13}$$

$$BWT_k = \frac{1}{k-1}\sum_{j=1}^{k-1}(a_{k,j} - a_{j,j}) \tag{A.14}$$

$$FWT_k = \frac{1}{k-1}\sum_{j=2}^{k}(a_{j,j} - \tilde{a}_j) \tag{A.15}$$

Where $a_{k,j}$ is the accuracy evaluated on the test set of the jth after incremental learning of the kth ($j \leq k$) and $\tilde{a}_j$ is the classification accuracy of a randomly initialized reference model trained on task j.

A.5 Computational and resource efficiency metrics

In addition to the above-mentioned accuracy-based metrics, the performance of the models was also evaluated in terms of computational and resource efficiency to assess their feasibility for deployment on resource-constrained embedded systems.

The performance of the proposed DL models in terms of computation metrics, including model complexity, size, peak memory usage, and the number of parameters used in the model, was also assessed. These metrics are particularly important when deploying DL models on resource-constrained MCUs.

In this context, model size refers to the flash storage required to store the model on the MCU. RAM usage represents the runtime memory required during model execution. Therefore, peak memory usage, which indicates the maximum memory required by the model during inference, was used to assess the model in terms of its memory requirements on the MCU. The target deployment platform in this study was MCU-based devices with only a few megabytes of flash and RAM, which impose strict constraints on model size and peak memory usage.

In addition, model complexity was evaluated by the number of MACs (Multiply-Accumulate). MACs represent the number of multiplications and additions performed during one inference, which provides an estimate of the computational cost required for model inference.

The performance of the DL models was also assessed in terms of inference time, which represents the time required to process an input sample and produce a prediction on hardware. Moreover, the run time of one operational cycle was measured and evaluated to provide a comprehensive assessment of the whole system.

Finally, considering that the model should run on a battery-powered device, power consumption, as a critical factor affecting device lifetime, was also evaluated. In this thesis, three related metrics were considered when analyzing the energy behavior of the system: current consumption, power consumption, and energy consumption. Current consumption refers to the electrical current drawn by the system during operation and is typically measured in milliamperes (mA). Power consumption refers to the rate at which energy is consumed by the system (i.e., the amount of energy consumed per unit time). It depends on both the supply voltage and the current drawn

by the device and is typically measured in watts (W). Energy consumption refers to the total amount of energy used over a period of time or during a specific task, such as one complete operational cycle of the system, typically measured in joules (J) or watt-hours (Wh). Therefore, current consumption was primarily used to describe the electrical behavior of the hardware components, while energy consumption was used to evaluate the total energy required for specific operations or processing cycles of the proposed system.