Computers and Electronics in Agriculture 236 (2025) 110378

Contents lists available at ScienceDirect

Computers and Electronics in Agriculture

journal homepage: www.elsevier.com/locate/compag

Original papers

Tiny deep learning model for insect segmentation and counting on resource-constrained devices

Amin Kargar [a,c,*], Dimitrios Zorbas [b], Michael Gaffney [c], Brendan O'Flynn [a], Salvatore Tedesco [a]

[a] Tyndall National Institute, University College Cork, Cork, Ireland
[b] Nazarbayev University, School of Engineering & Digital Sciences, Astana, Kazakhstan
[c] Horticulture Development Department, Teagasc Ashtown Food Research Centre, Dublin, Ireland

ARTICLE INFO

Dataset link: https://doi.org/10.5281/zenodo.14051319

Keywords:
Insect monitoring
Image segmentation
Edge computing
Tiny deep learning
On-device deep learning
Microcontrollers
Halyomorpha halys

ABSTRACT

Automated insect monitoring is essential for early detection of insect pest infestations in orchards. It helps growers make an informed decision to control the insect pest population in their fields to avoid crop losses and improve crop quality. This study proposed a tiny deep-learning model for insect segmentation and counting suitable for battery-powered microcontroller (MCU)-based edge devices. For this aim, the critical layers in terms of peak memory usage in a U-Net-inspired model were recognized and then optimized to meet resources constraints on an MCU with 1 MB of RAM and 2 MB of flash storage. We then introduced an image dataset for the insect of interest, *Halyomorpha halys*, and the dataset-splitting strategy for the model training. The proposed model was investigated from different aspects to evaluate its performance and the feasibility of its implementation on battery-powered MCU-based edge devices. The proposed deep learning model only needs approximately 900 KB of RAM and 964 KB of storage to perform its computations and store its parameters, respectively, making it useful on edge, resource constrained systems. Moreover, each inference on an MCU-based board requires 2.6 s and consumes 4.9 J. In terms of segmentation, it achieved a Dice Similarity Coefficient (DSC) of 85% and an Intersection over Union (IoU) of 73% with a precision and recall of 83% and 86%, respectively. In terms of counting, it achieved a Mean Square Error (MSE) of 1.32, Mean Absolute Error (MAE) of 0.78 and R^2 of 0.97.

1. Introduction

Ensuring food security will become a significant challenge as the worlds population continues to grow. The human population is estimated to increase to nearly nine billion people by 2025. Moreover, the Food and Agriculture Organization of the United Nations (FAO) reported an estimate of approximately 40 percent of crop loss annually. These losses cost about $220 billion, a third of which is attributed to invasive insects (Karam et al., 2022). 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.

Using pesticides is the first strategy commonly used by agronomists to control insect populations in their fields, but this approach has harmful side effects on human and animal health, biodiversity and the environment (Teixeira et al., 2023b; She et al., 2022). It is thus important to use pesticides appropriately to effectively control insect pest populations, while also minimizing the pesticides environmental impacts. To achieve this balance, growers should monitor their fields to recognize the presence and population of insect pests in their fields. Traditionally, growers have had to physically monitor their orchards and manually inspect 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. In addition, visual monitoring and identification of insect species is a time-costly, labour-intensive and error-prone task requiring specific skill sets in insect identification (Li et al., 2020; Zhao et al., 2022). Approaches such as the presence-absence model (PAM) (Lee et al., 2022) were 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 (Chen et al., 2018).

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 (Teixeira et al., 2023b; Saranya et al., 2023).

* Corresponding author at: Tyndall National Institute, University College Cork, Cork, Ireland.
E-mail addresses: amin.kargar@tyndall.ie (A. Kargar), dimitrios.zorbas@nu.edu.kz (D. Zorbas), michael.gaffney@teagasc.ie (M. Gaffney), brendan.oflynn@tyndall.ie (B. O'Flynn), salvatore.tedesco@tyndall.ie (S. Tedesco).

https://doi.org/10.1016/j.compag.2025.110378
Received 7 November 2024; Received in revised form 14 March 2025; Accepted 2 April 2025
Available online 18 April 2025

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 (Popescu et al., 2023). Moreover, the use of such systems reduce the need for in-person visits, resulting in a reduction in fuel consumption, emissions, costs, and time (Passias et al., 2024). Automated insect monitoring will also support the implementation of the EU's Green Deal, which aims to reach a 50% reduction in pesticide usage by 2030 (Bremmer et al., 2021).

Halyomorpha halys (HH), also known as Brown Marmorated Stink Bug (BMSB), is a destructive example of an invasive insect originally from East Asia that now infests orchards in North America and Europe. This is a polyphagous insect that can cause significant damage to a wide range of vegetables and fruit trees. 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 (Giannetti et al., 2024).

Therefore, a European-funded research project named HALY.ID was proposed to monitor this insect. As part of this project, this study aims to develop an accurate battery-powered Internet of Things (IoT)-based device for insect monitoring that utilizes a deep learning (DL) model running on resource-constrained devices operating "on the edge". The DL model is developed to carry out insect identification, segmentation and counting, specifically focusing on HH detection. Running the proposed DL model on edge devices allows segmenting and counting the target insect from captured images at the source in real-time, and more importantly, as these are edge based devices, eliminates the need for a reliable internet connection 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 internet connectivity as well as power lines. Microcontrollers (MCU) are well-suited for battery-powered IoT-based edge devices because of their low-cost and low-power consumption features. Thus, the proposed DL model is optimized to run on resource-constrained, MCU-based devices which only have up to 1 MB of RAM and 2 MB of flash storage (e.g. STM32H7).

The main contributions of this study are as follows:

- Semi-YNet, a super lightweight 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 are annotated in two different ways, which makes the dataset suitable for both detection (bounding box) and segmentation task (mask). The dataset is split into training, validation, and test sets based on stratified sampling according to the number of insects on images.
- The model is evaluated on several aspects, including accuracy metrics to evaluate its segmentation and counting performance, and also implementation metrics to investigate its implementation possibility on resource-constrained platforms such as MCUs. Moreover, the proposed Semi-YNet model was implemented and validated on an MCU-based board to evaluate its power consumption and inference time.

The remainder of this paper is organized as follows. Related studies are reviewed in Section 2. In Section 3, the proposed deep learning model suitable for resource constrained MCUs is described. In Section 4, the image dataset developed for the target insect is introduced, then the metrics that were used to evaluate the proposed model, hyperparameters and training details are described. In Section 5, the proposed model is evaluated from different aspects including performance metrics and implementation metrics and the results are reported. Finally, conclusions are drawn in Section 6.

2. Related work

Several studies proposed DL-based insect monitoring systems to automate the insect detection task. Tang et al. (2020) 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 gained better 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. (2023a) suggested segmentation by the U-Net model as a pre-processing step and then using a YOLO (Jiang et al., 2022) (You Only Look Once) 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%. Júnior et al. (2022) proposed an insect detection system to detect aphids and parasitoids from trap images in the lab. The authors utilized Mask R-CNN for insect detection on a server and achieved R^2 score of 0.81 and 0.78 for aphids and parasitoids counting, respectively. In the mentioned studies, the authors used powerful and power-hungry cloud, server, or computer systems and the main focus was on achieving the highest model accuracy. They also depend on a reliable Internet connection or communication infrastructure to transfer raw captured images to a cloud or server, which is generally not available in remote orchards.

More recently, edge-based systems were considered for the insect monitoring task to tackle the above-mentioned issues. Bjerge et al. (2023) evaluated YOLOv5 and YOLOv3 on their collected dataset and achieved 92% of AP, 94% recall, and 93% F1-score. The authors used a Raspberry Pi 3B based system to collect the dataset and applied a DL model after collecting the data but stated that the DL model could be run on a Raspberry Pi or other similar boards for on-site prediction. Brunelli et al. (2019) developed a device to detect codling moths using machine learning algorithms. The proposed device includes a Raspberry Pi 3 providing the pre-processing step and an Intel Movidius neural compute stick to run the DL model (i.e., VGG16) for the classification step. Similarly, Zhong et al. (2018) proposed a YOLO-based system for counting and recognizing six different flying insect species. The proposed method consisted of a YOLO model for the detection and approximate counting of insects and an SVM model for more accurate counting and classification. A Raspberry Pi 2B was used to implement the proposed model and achieved 92.5% accuracy in counting and 90% accuracy in classification. Pham et al. (2021) proposed AlerTrap which is an edge-computing insect monitoring device for remote areas. The authors used a Raspberry Pi 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. They investigated the performance of three different models including 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. (2023) developed an edge-based insect pest monitoring system for mango orchards. The proposed YOLO-based model achieved an F1-score of 96% and an average inference time of about 11 s on a Raspberry Pi Zero W which occupied approximately 151 and 57 MB of GPU memory and storage, respectively.

The insect of interest in this study (HH) has been the subject of some studies recently. Sorbelli et al. (2023) created a dataset of HH using images captured by drones and devices such as smartphones. In order to improve the quality of the dataset samples, a preliminary screening process was conducted. Subsequently, different versions of YOLO-v5 were trained on the dataset and analysed using several metrics for HH detection. Palazzetti et al. (2024) introduced a dataset

for the target insect using autonomously captured aerial telephotos. They trained and evaluated several state-of-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 the drone inspection of the orchard and also to detect the HH on captured images (upon the request of a user) by transferring the image to a server equipped with the trained models. Recently, Kargar et al. (2024a) proposed the Y-Net model, which is inspired by U-net architecture to segment and count the HH target insect from trap images with a simple background. This is a suitable DL-based method computationally suitable for embedded devices such as smartphones and Raspberry Pi devices. The proposed model achieved 84.5% and 91.5% of intersection over union (IoU) and dice similarity coefficient (DSC) for the segmentation task, respectively. The mean squared error (MSE) was 1.9 and required approx. 0.4 s on an Android mobile phone for each inference attempt.

Apart from server/cloud-based systems that use powerful power-hungry hardware, the proposed edge-based systems mostly rely on Raspberry Pi boards which are still expensive and consume more energy compared to MCU (Kargar et al., 2024b). Bompani et al. (2024) developed 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 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 int8 quantization and converted RGB images to greyscale with the size of 320 × 240 to compress the model. Crupi et al. (2024) 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 gained a mean average precision (mAP) of 0.66 and 0.79, respectively. To decrease the memory footprint of the FOMO lower than 1 MB, they reduced the input size to 160 × 160 and 96 × 96. For the SSDLite, the memory footprint was around 1.8 MB with an input size of 320 × 320.

MCUs are a good fit for battery-powered Internet of Things (IoT)-based edge devices because of their low-cost and low-power consumption characteristics, which are critical factors for such devices, especially for this application where long-lasting performance in remote locations is essential. However, typical MCUs only have up to 1 MB of RAM and 2 MB storage, which makes it challenging to implement and run DL models on them. Although, there are several techniques, such as knowledge distillation (Xu et al., 2024) and pruning (Zhang and Lv, 2024), whose main objective is to reduce the model size, 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 during the inference time. However, quantization, which is widely used for model size reduction, is an effective method that also reduces memory usage by representing weights and activations in lower precision (e.g., int8 instead of float32), but is still insufficient to meet the strict resource constraints of MCUs on its own. Therefore, in this study, we propose a DL model for insect monitoring suitable for resource-constrained MCUs. We optimize the RAM and storage usage of the employed DL model to align with the MCU constraints.

3. Proposed model

This paper describes the development of a system with the ability to count and segment HH from captured images on an edge based IoT-based device. As mentioned, the IoT device is designed around a microcontroller, meaning that all the counting and segmentation computation should be performed on a resource-constrained MCU. In this study, the target size in terms of RAM and flash memory storage is 1 MB and 2 MB, respectively, which corresponds to the capability of a typical modern MCU device.

The proposed model is based on the Y-Net model which we recently introduced (Kargar et al., 2024a). Y-Net is a DL model with two outputs and was developed to simultaneously count and segment insects on input images. It can run on embedded edge devices such as smartphones or Raspberry Pi boards with available storage and RAM in the Gigabyte range. Since the goal for this study is to perform the counting and segmentation on MCUs, we propose SemiY-Net, which is described in the following paragraphs.

3.1. Y-Net model

The key points of Y-Net are first briefly described. The Y-Net model, shown in Fig. 1, was inspired by the U-Net model (Ronneberger et al., 2015), which is widely used for image segmentation. Like U-Net, it has an encoder path and a decoder path that extract critical features from the input and generate segmentation output respectively. 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. In Y-Net, the segmentation output generates a black-and-white image with the same dimensions as the input image, in which the objects are marked with white pixels while the other areas are marked with black pixels representing the background. In this application, it is required to estimate the insect population in the orchard to help farmers make a proper decision about pesticide use. Thus, such a device should be able to count the insects. To this end, methods, such as blob or contour detection algorithms, can be used to detect objects on the segmentation output for insect counting. However, this 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. Therefore, in addition to the segmentation output, Y-Net has another output indicating the number of objects (e.g. the target insects) in the input image. This output is taken from the bottleneck where the encoder and decoder paths are connected. In the counting part, as illustrated in Fig. 1, the extracted features are passed through a convolution layer and then a Global Average Pooling (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 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.

In fact, 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. We investigate other parts of the model 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. Our 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. Moreover, the lightweight but powerful MobilenetV2 architecture (Sandler et al., 2018) is utilized in the encoder path for feature extraction. To this aim, the pre-trained MobilenetV2 available in Keras (e.g., trained on the ImageNet dataset) with the alpha value of 0.5 was used.

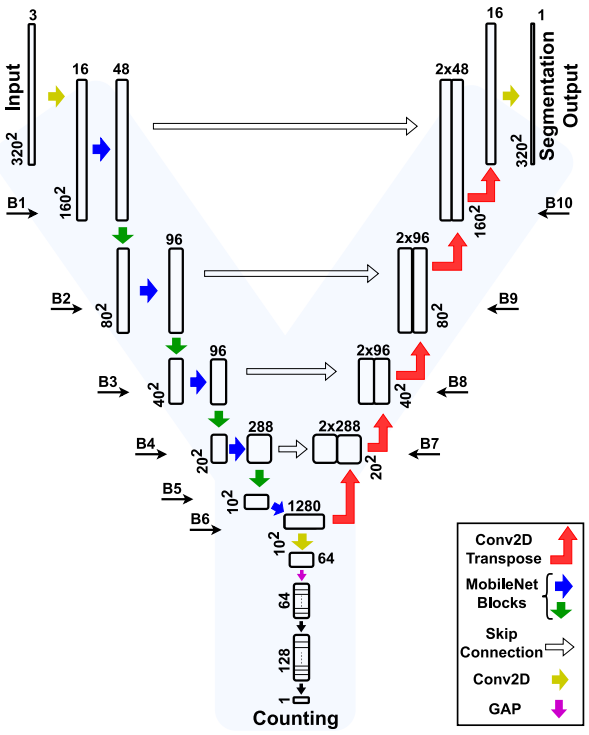

Fig. 1. Y-Net architecture for input size of 320 × 320 (Kargar et al., 2024a); Blocks (B) are defined for the simplicity of the model analysis.

3.2. SemiY-Net

The use of resource-constrained devices impose strict limitations on the design of an accurate DL model specifically in terms of the 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 analysed in terms of:

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

In Kargar et al. (2024a) we examined several configurations of the Y-Net model; in this study, Y-Net with alpha value (i.e., a parameter that controls the complexity of the MobilenetV2) 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 implementation metrics.

3.2.1. Peak memory usage

To analyse the Y-Net memory usage, we divided the model into several Blocks (B) for simplicity and better visualization. Fig. 2 depicts the peak memory usage of Y-Net. The mentioned division was performed based on the feature map shapes; for example, the input has the shape of 320 × 320, Block 1 (B1) has the shape of 160 × 160 and so on. According to Fig. 2, it is obvious that Block 10 allocated the highest memory usage to itself, at nearly 5 MB, which is 5 times the available RAM on a MCU. Moreover, the majority of the blocks need more than 2 MB 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, we investigated the details of the model as follows:

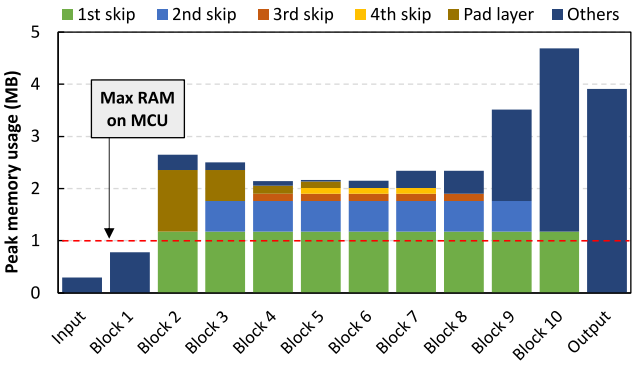

Fig. 2. Peak memory usage of the Y-Net model.

- Based on Fig. 2, 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 Blocks 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 × 160x96 which alone occupies 2.3 MB of RAM.

Above, critical layers and factors that have a significant impact on the model memory usage are described. By considering the mentioned points, we propose SemiY-Net, which is shown in Fig. 3. As such, the following steps are 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 its 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.
- 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 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.

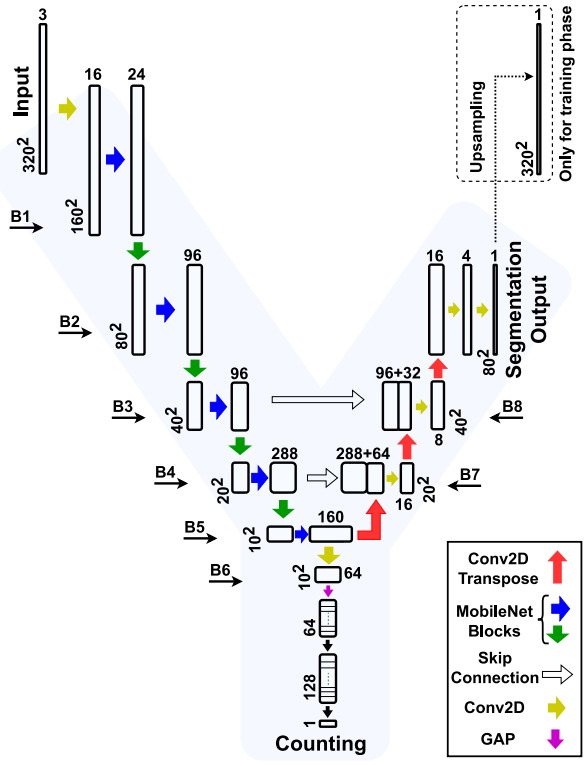

Fig. 3. The proposed Semi-YNet architecture.

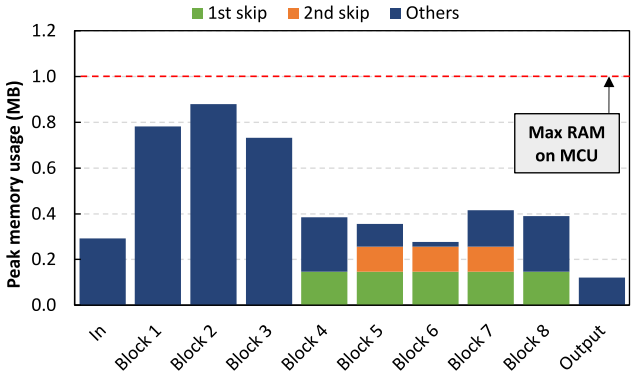

Fig. 4. SemiY-Net's peak memory usage.

Fig. 4 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 has the same size as the input, but the proposed SemiY-Net does not follow this procedure where the input size is 320 × 320 while the output size is 80 × 80. Therefore, it is needed to resize the output from 80 × 80 to 320 × 320 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.

3.2.2. Model size

Regarding the model size, the Y-Net model had 5.5 million parameters. By using quantization techniques and converting the data types from `float32` to `int8`, the model size decreased from about 21 MB to 5.6 MB, which is 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.

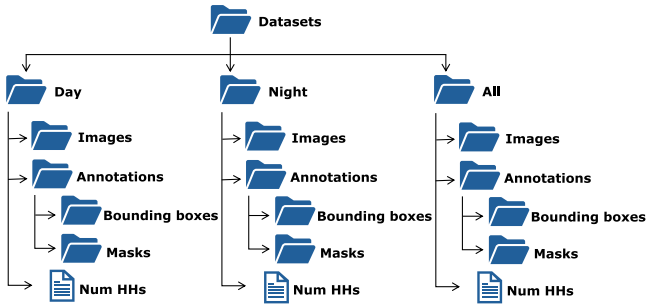

Fig. 5. Dataset structure.

4. Experimental setup & configuration

4.1. Datasets

In this study, we created an image dataset for *Halyomorpha halys* insects captured from two-sided sticky traps. The dataset supports object detection and segmentation tasks by providing two different types of annotation. The dataset structure is shown in Fig. 5. 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 an IoT device developed for the project (Kargar et al., 2024b), 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 2222 HHs, and the 'Night' one has 267 images with 2452 HHs. Thus, the 'All' dataset has 523 images with 4674 HHs in total. Each sub dataset includes three folders as follows:

- Images: This contains the original images in JPEG format from the sticky traps surface that might include the HH as well as other insects.
- Annotations: This folder contains the annotations related to the 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 files indicating the HH locations in images.
- Num HHs: This folder contains a CSV file that indicates the number of HH on images.

Fig. 6 shows two samples from 'Day' and 'Night' datasets.

4.2. 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. Fig. 7 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 our 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.

Fig. 8a shows each side distribution when the dataset was randomly split into training, validation, and test sets. It should be noted that in

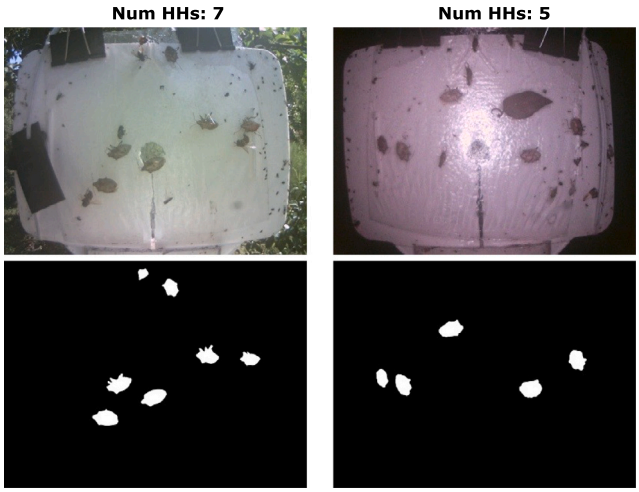

Fig. 6. Two samples from 'Night' and 'Day' datasets; the first row is the captured images and the second row is the corresponding mask.

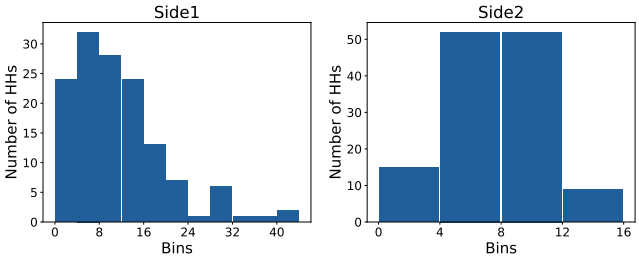

Fig. 7. Distribution of the target insect numbers for each side of the Night dataset; the number of HHs were binned with the bins size of four.

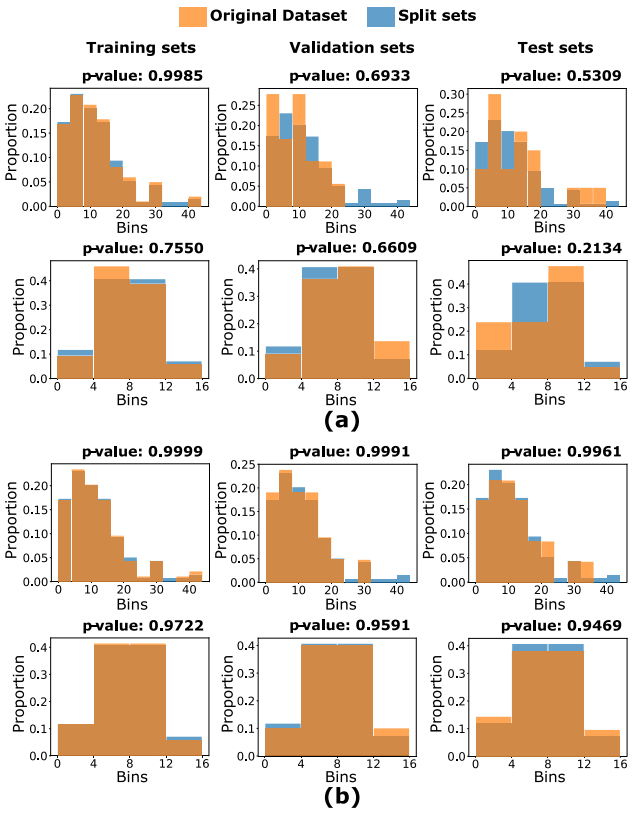

Fig. 8. Distribution of the target insect numbers on each side for training, validation and test sets with the corresponding *p*-value of chi-square goodness of fit test; a: random splitting, b: stratified-based splitting.

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 Fig. 8a, there is no guarantee to preserve the data distribution of each side if the images are randomly split. To overcome this issue, we used stratified sampling based on the HH numbers. 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 done for the validation and test sets. Fig. 8b, shows the strategy used for dataset splitting; it is obvious 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, we statistically analysed the proposed splitting strategy 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 observed data matches the frequency of expected data, 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 Fig. 8. Based on these two figures, it is obvious that the proposed strategy has a higher *p*-value for all sets compared to random splitting, indicating a stronger match.

4.3. Evaluation metrics

Two different types of metrics were used to investigate the proposed model, including performance and implementation metrics.

4.3.1. Performance metrics

These are metrics that evaluate the accuracy of the DL model predictions. Since the proposed model has two different outputs, different metrics were utilized to evaluate their performance.

The counting output that aims to predict the number of HHs was evaluated by Mean Square Error (MSE), Mean Absolute Error (MAE), and R^2. The MSE and MAE metrics measure the error between the actual number of insects on the image with the predicted one by the model, while R^2 measures the degree of how well the predicted insect number matches the actual insect number. These three metrics are calculated as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{1}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{3}$$

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.

The segmentation output that 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 (Müller et al., 2022).

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 follows:

$$P = \frac{TP}{TP + FP} \tag{4}$$

$$R = \frac{TP}{TP + FN} \tag{5}$$

$$IoU = \frac{P \times R}{P + R - P \times R} = \frac{TP}{TP + FP + FN} \tag{6}$$

$$DSC = \frac{2 \times P \times R}{P + R} = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{7}$$

where TP is the number of true positives, FP is the number of false positives, and FN is the number of false negatives.

4.3.2. Implementation metrics

As mentioned, the model was designed to be deployed at the edge on a resource-constrained MCU, therefore it should be evaluated based on some hardware-related factors to investigate its feasibility and efficiency for real-world deployments. In this study, we evaluate the power consumption, the inference time, the model size, the peak memory usage, and the model complexity. Considering that the model should run on a battery-powered IoT device, power consumption is a critical factor affecting the device lifetime. Regarding the model complexity, the total number of multiplication and addition operations required for one prediction (known as MAC, e.g., Multiply-Accumulate) was used to assess the model complexity. This metric directly impacts inference time and subsequently the power consumption of the model.

4.4. Hyperparameters and training details

SemiY-Net was implemented using TensorFlow and Keras. The dataset was split into training, validation, and test sets as discussed in Section 4.2. Additionally, images and masks were resized to 320 × 320. We also benefited from a histogram equalization technique to improve the image contrast, which has a significant impact on the image analysis algorithm (Yoshimi et al., 2024). 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.

As described, a pre-trained MobileNetV2 was used in the Y-Net encoder part for feature extraction, while, we modified some layers of the encoder part to design the 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, we adjusted the learning rate of the Adam optimizer to 0.0002, the epoch number to 500 and the batch size to 8 during the network training by exploring various values of these hyperparameters. Moreover, augmentation techniques were applied on each batch during the training phase to reduce the overfitting and improve the model generalization. This technique creates new variations of the images by using random transformations. In this study, we applied rotation, horizontal and vertical flip, random brightness and contrast transformation for the augmentation method.

5. Evaluation & discussion of the results

In this section, the proposed SemiY-Net model is evaluated and compared to other models from different aspects. For this aim, we trained several other models to compare our model performance with them including Y-Net, DeepLabV3+, YOLOV8n-det, YOLOV8n-seg, and our previous method (Kargar et al., 2024b). It should be noted that all the models perform the segmentation task except YOLOV8n-det and our 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 models 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 our previous method, in this paper, we employed a modified version proposed in Section 5 of Almstedt et al. (2024). This method first utilizes an image processing technique to detect potential areas where HH reside, and then it crops those areas and applies a DL classification model for classification (i.e., HHs or non-HHs) and counting on the cropped images.

5.1. SemiY-Net evaluation

In this subsection, we investigate the impact of the segmentation output size on the model performance to find the best output size offering an optimal trade-off between the mentioned metrics, including model accuracy and implementation metrics. Table 1 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 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 the paper, 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 1. 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 Fig. 4), 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.

Table 1
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 applied to the segmentation output.

Moreover, we trained the proposed semiY-Net from scratch, without loading the pretrained weights to the modified MobilenetV2 as the initial weights in the encoder part. As expected, loading pretrained weights helped the model to better extract the features, leading to around a five percent increase in DSC and a 1.6-fold reduction in MSE. To further investigate the potential impact of removing the pretrained layers of the MobileNetV2, we implemented another version of the Y-Net in which only the MobileNetV2 layers were modified based on the steps described in Section 3.2.1. Results showed that modifying the MobileNetV2 part only in the Y-Net led to a lower than 6 percent reduction in segmentation accuracy, but did not significantly affect the counting performance. Moreover, the peak memory usage of the entire model was decreased to about 3.2 MB, while the original Y-Net was around 4.8 MB. Although this shows a 1.6 MB reduction in peak memory usage, it is still higher than 1 MB of the available memory on the MCU.

5.2. Performance metrics

Table 2 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 our 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 Almstedt et al. (2024). In terms of segmentation, apart from YOLOv8n-det which is a detection model, YOLOv8n-seg has the highest performance followed by the 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. Therefore, we partially trade off accuracy to be able to implement the model on MCU. This highlights that there is a cost to implement DL models at the edge on resource-constrained MCUs (Diab and Rodriguez-Villegas, 2022; Svoboda et al., 2022). Apart from reducing the model parameters in the proposed model, we also utilized post-training quantization to convert parameters data type from original `float32` to `int8`. In this way, parameters with 32-bit precision converted to 8-bit integer numbers led to around 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 error.

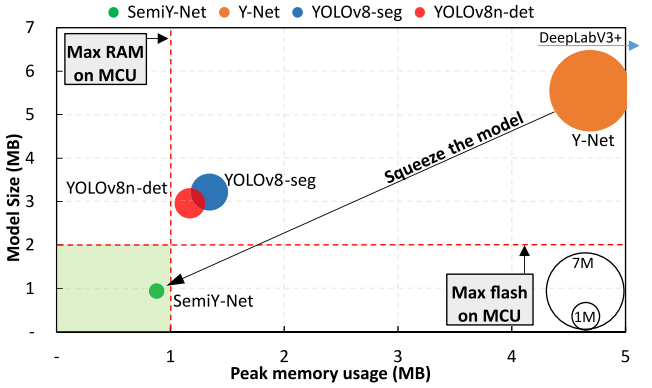

Fig. 9. Comparison of models size (y-axis), peak memory usage (x-axis) and MAC (bubbles size); green region illustrates an area in which a model can be implemented on the MCU.

5.3. Implementation metrics

Fig. 9 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., 1 MB of RAM and 2 MB of flash storage). It is obvious 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 we did not show DeepLabV3+ in the figure for better visualization since its values are 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. This increases the inference time and results in higher power consumption.

5.4. Power consumption and inference time on MCU

Based on the results presented, the only model that can be successfully implemented on MCUs is SemiY-Net. In this subsection, we implement the proposed SemiY-Net model on an MCU-based OpenMV

Table 2
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
Our previous method (Kargar et al., 2024b; Almstedt et al., 2024)	– –	– –	– –	– –	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.

board.[1] This is a board based on the STM32H7 ARM Cortex M7 processor with a built-in camera which is specifically designed for machine vision tasks and can be programmed with MicroPython. As mentioned, the proposed SemiY-Net model was implemented using TensorFlow and then trained on the dataset. Afterwards, the trained model was exported as a TensorFlow Lite (tflite) model, a compatible format for the 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. Since the model input size is 320 × 320, the board was adjusted to capture RGB images with this size. In an operation time, 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 on the 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 we followed to collect the dataset. In a real-life deployment, the transfer of the decision over a long range radio technology would be enough (Almstedt et al., 2024). Finally, the device goes into deep sleep mode until the next operation cycle to reduce the power consumption of the overall system. Fig. 10 shows the current consumption of the board over one cycle of the operation time. The starting point of each stage is shown by a vertical line and also the average elapsed time is reported in this figure. Around 44% of one operation time accounted for system booting, while the proposed SemiY-Net inference time is 2.6 s which is 24% of the operation time. The image capturing and results saving phases only needed 0.2 s. Finally, the data transfer phase needed 3.25 s which is around 30% of the total time; it is worth mentioning that in this phase, a significant amount of time (and power consumption) is associated with the Wi-Fi connection.

Moreover, Fig. 11 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 J respectively. Although the boot and data transfer stages have the highest elapsed time, the DL model consumes more energy during running than the others due to higher current consumption. The average current consumption of the boot and transferring 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 energy consumption in battery-powered edge-based devices.

5.5. 'Night', 'Day', and 'All' datasets

The performance of the proposed SemiY-Net is investigated on each dataset introduced in Section 4.1, including the 'Day', 'Night', and 'All'.

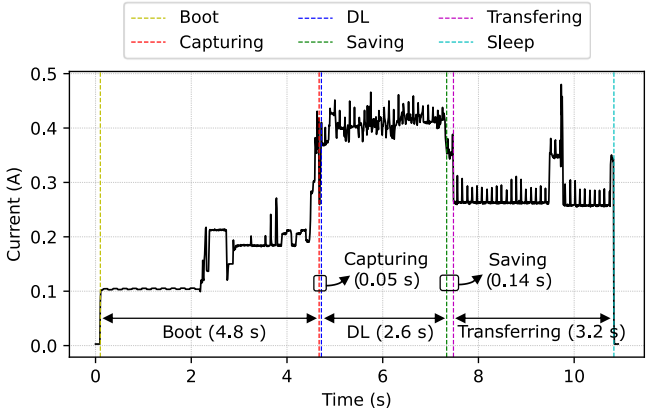

Fig. 10. Current consumption of the proposed SemiY-Net in an operation time.

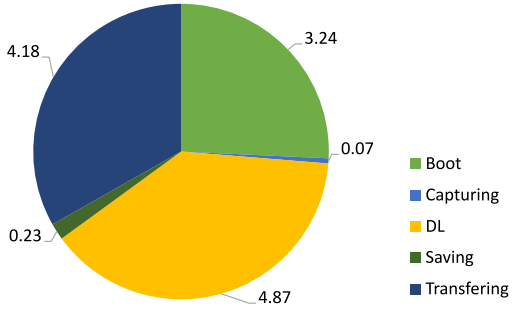

Fig. 11. Energy consumption of the proposed SemiY-Net in joules for an operation time.

The segmentation and counting performances are shown in Fig. 12. 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 (Kargar et al., 2024b). Fig. 6 compares two samples of the 'Day' and 'Night' datasets. It is obvious 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 minimized, resulting in simpler images which facilitate the model training.

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

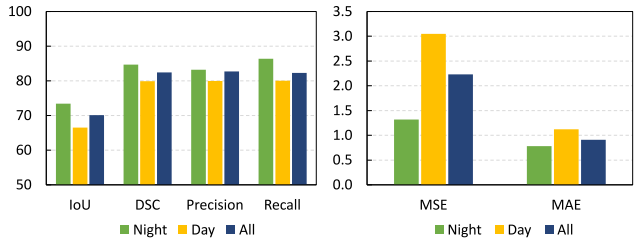

Fig. 12. The proposed SemiY-Net model performance on the 'Night', 'Day', and 'All' datasets.

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 the model trains better when there are more samples in the dataset; 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.

5.6. 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 discussed in Section 4.3. 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, we proposed to decrease the output segmentation size from 320 × 320 to 80 × 80 in SemiY-Net. By doing so, we were able to simplify the model 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 us 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, our 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 s (see Fig. 10), considering that the device operates two times a day, the device would be in sleep mode for almost the entire day except those 22 s of its operation time. The average current consumption in operation time is approximately 255 mA resulting in 1.6 mAh consumption per day. For the sleep time, if the current consumption is 7 mA (for the device in Kargar et al. (2024b)), the consumption would be approximately 168 mAh which is 107 times more than the operation time consumption. For a device with a 5 Ah battery capacity, the battery lifetime would be around 29 days. But if we decrease the sleep mode consumption to 1 mA, the battery lifetime would be 196 days. This analysis clearly shows the impact of such device consumption during sleep mode. Note that this is a theoretical estimation and several factors, such as battery self-discharge, affect the battery lifetime. Moreover, since such devices are deployed outdoors, we can use solar panels to recharge the battery as discussed in Kargar et al. (2024b).

Moreover, Some examples of the proposed model output are shown in Fig. 13. 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

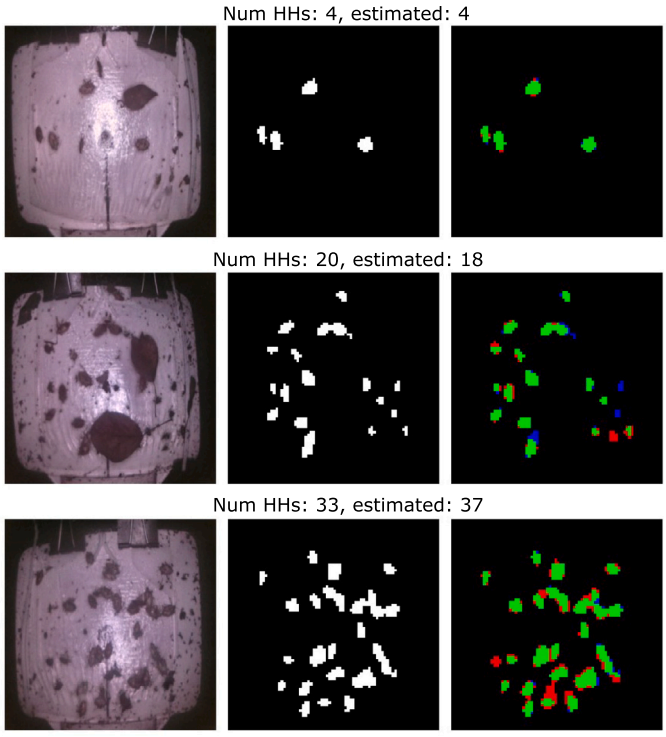

Fig. 13. Samples of SemiY-Net outputs; first column is the images, the second column is the true mask, and the third column is the detailed model output (Green: TP, Red: FP, Blue: FN).

images with three different rates of overlap between insects, from low to high. It is obvious 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.

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 through the use of image tiling. However, it increases the processing time and consequently the energy consumption required since the model should run on each tile separately.

6. Conclusions & future work

This study presents a SemiY-Net which is a super tiny deep-learning 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. The target insect of this study was *Halyomorpha halys* and an image dataset was created from the real world to train and evaluate the proposed model.

A future focus will be on improving the robustness of the model to low-frequency samples in the dataset to enhance its accuracy.

CRediT authorship contribution statement

Amin Kargar: Writing – original draft, Visualization, Validation, Software, Methodology, Conceptualization. **Dimitrios Zorbas:** Writing – review & editing, Supervision, Methodology, Funding acquisition, Conceptualization. **Michael Gaffney:** Writing – review & editing, Supervision, Resources, Funding acquisition. **Brendan O'Flynn:** Writing – review & editing, Supervision, Resources, Project administration, Methodology, Funding acquisition, Conceptualization. **Salvatore Tedesco:** Writing – review & editing, Supervision, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This project is co-funded by the European Regional Development Fund (ERDF) under Ireland's European Structural and Investment Funds Programmes 2014–2020. 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. The first author is 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-VISTAMILK2which are co-funded by the European Regional Development Fund (ER DF) under Ireland's European Structural and Investment Funds Programme 2014–2020. For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission. Moreover, we would like to express our gratitude to postdoctoral researcher, Dr. Mariusz Wilk, for his contribution to the design and development of the IoT device used for dataset collection.

Data availability

The dataset is available at https://doi.org/10.5281/zenodo.14051319.

References

Almstedt, L., Sorbelli, F.B., 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 Trans. AgriFood Electron..

Bjerge, K., Alison, J., Dyrmann, M., Frigaard, C.E., Mann, H.M., Høye, T.T., 2023. Accurate detection and identification of insects from camera trap images with deep learning. PLoS Sustain. Transform. 2 (3), e0000051.

Bompani, L., Crupi, L., Palossi, D., Baldoni, O., Brunelli, D., Conti, F., Rusci, M., Benini, L., 2024. Accelerating image-based pest detection on a heterogeneous multicore microcontroller. IEEE Trans. AgriFood Electron..

Bremmer, J., Gonzalez-Martinez, A., Jongeneel, R., Huiting, H., Stokkers, R., Ruijs, M., 2021. Impact Assessment of EC 2030 Green Deal Targets for Sustainable Crop Production. Wageningen Economic Research, no. 2021–150.

Brunelli, D., Albanese, A., d'Acunto, D., Nardello, M., 2019. Energy neutral machine learning based iot device for pest detection in precision agriculture. IEEE Internet Things Mag. 2 (4), 10–13.

Chen, J., Fan, Y., Wang, T., Zhang, C., Qiu, Z., He, Y., 2018. Automatic segmentation and counting of aphid nymphs on leaves using convolutional neural networks. Agronomy 8 (8), 129.

Crupi, L., Butera, L., Ferrante, A., Palossi, D., 2024. A deep learning-based pest insect monitoring system for ultra-low power pocket-sized drones. In: 2024 20th International Conference on Distributed Computing in Smart Systems and the Internet of Things. DCOSS-IoT, IEEE, pp. 323–330.

Diab, M.S., Rodriguez-Villegas, E., 2022. Embedded machine learning using microcontrollers in wearable and ambulatory systems for health and care applications: A review. IEEE Access 10, 98450–98474.

Giannetti, D., Patelli, N., Palazzetti, L., Betti Sorbelli, F., Pinotti, C.M., Maistrello, L., 2024. First use of unmanned aerial vehicles to monitor Halyomorpha halys and recognize it using artificial intelligence. Pest. Manag. Sci..

Jiang, P., Ergu, D., Liu, F., Cai, Y., Ma, B., 2022. A review of yolo algorithm developments. Procedia Comput. Sci. 199, 1066–1073.

Júnior, T.D.C., Rieder, R., Di Domênico, J.R., Lau, D., 2022. InsectCV: A system for insect detection in the lab from trap images. Ecol. Inform. 67, 101516.

Karam, C., Awad, M., Abou Jawdah, Y., Ezzeddine, N., Fardoun, A., 2022. GAN-based semi-automated augmentation online tool for agricultural pest detection: A case study on whiteflies. Front. Plant Sci. 13, 813050.

Kargar, A., Zorbas, D., Gaffney, M., O'Flynn, B., Tedesco, S., 2024a. 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.

Kargar, A., Zorbas, D., Tedesco, S., Gaffney, M., O'Flynn, B., 2024b. Detecting halyomorpha halys using a low-power edge-based monitoring system. Comput. Electron. Agric. 221, 108935.

Lee, H., Choi, W., Eom, S., Park, J.-J., 2022. Rapid estimation of the density of whiteflies (Hemiptera: Aleyrodidae) on sticky traps in paprika greenhouses using the presence–absence model. J. Asia- Pac. Biodivers. 15 (2), 225–230.

Li, Y., Wang, H., Dang, L.M., Sadeghi-Niaraki, A., Moon, H., 2020. Crop pest recognition in natural scenes using convolutional neural networks. Comput. Electron. Agric. 169, 105174.

Müller, D., Soto-Rey, I., Kramer, F., 2022. Towards a guideline for evaluation metrics in medical image segmentation. BMC Res. Not. 15 (1), 210.

Palazzetti, L., Rangarajan, A.K., Dinca, A., Boom, B., Popescu, D., Offermans, P., Pinotti, C.M., 2024. The hawk eye scan: Halyomorpha halys detection relying on aerial tele photos and neural networks. Comput. Electron. Agric. 226, 109365.

Passias, A., Tsakalos, K.-A., Rigogiannis, N., Voglitsis, D., Papanikolaou, N., Michalopoulou, M., Broufas, G., Sirakoulis, G.C., 2024. Insect pest trap development and DL-based pest detection: A comprehensive review. IEEE Trans. AgriFood Electron..

Pham, D.A., Le, A.D., Pham, D.T., Vo, H.B., 2021. Alerttrap: On designing an edge-computing remote insect monitoring system. In: 2021 8th NAFOSTED Conference on Information and Computer Science. NICS, IEEE, pp. 323–328.

Popescu, D., Dinca, A., Ichim, L., Angelescu, N., 2023. New trends in detection of harmful insects and pests in modern agriculture using artificial neural networks. a review. Front. Plant Sci. 14, 1268167.

Ronneberger, O., Fischer, P., Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation. In: Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18. Springer, pp. 234–241.

Rustia, D.J.A., Lee, W.-C., Lu, C.-Y., Wu, Y.-F., Shih, P.-Y., Chen, S.-K., Chung, J.-Y., Lin, T.-T., 2023. Edge-based wireless imaging system for continuous monitoring of insect pests in a remote outdoor mango orchard. Comput. Electron. Agric. 211, 108019.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.-C., 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4510–4520.

Saranya, T., Deisy, C., Sridevi, S., Anbananthen, K.S.M., 2023. A comparative study of deep learning and internet of things for precision agriculture. Eng. Appl. Artif. Intell. 122, 106034.

She, J., Zhan, W., Hong, S., Min, C., Dong, T., Huang, H., He, Z., 2022. 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. 70, 101690.

Sorbelli, F.B., Palazzetti, L., Pinotti, C.M., 2023. YOLO-based detection of halyomorpha halys in orchards using RGB cameras and drones. Comput. Electron. Agric. 213, 108228.

Svoboda, F., Fernandez-Marques, J., Liberis, E., Lane, N.D., 2022. Deep learning on microcontrollers: A study on deployment costs and challenges. In: Proceedings of the 2nd European Workshop on Machine Learning and Systems. pp. 54–63.

Tang, H., Wang, B., Chen, X., 2020. Deep learning techniques for automatic butterfly segmentation in ecological images. Comput. Electron. Agric. 178, 105739.

Teixeira, A.C., Carneiro, G.A., Morais, R., Sousa, J.J., Cunha, A., 2023a. Segmentation as a pre-processing for automatic grape moths detection. In: EPIA Conference on Artificial Intelligence. Springer, pp. 388–398.

Teixeira, A.C., Ribeiro, J., Morais, R., Sousa, J.J., Cunha, A., 2023b. A systematic review on automatic insect detection using deep learning. Agriculture 13 (3), 713.

Xu, D., Dong, Y., Ma, Z., Zi, J., Xu, N., Xia, Y., Li, Z., Xu, F., Chen, F., 2024. MAFIKD: A real-time pest detection method based on knowledge distillation. IEEE Sens. J..

Yoshimi, Y., Mine, Y., Ito, S., Takeda, S., Okazaki, S., Nakamoto, T., Nagasaki, T., Kakimoto, N., Murayama, T., Tanimoto, K., 2024. 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. 138 (1), 128–141.

Zhang, Y., Lv, C., 2024. TinySegformer: A lightweight visual segmentation model for real-time agricultural pest detection. Comput. Electron. Agric. 218, 108740.

Zhao, N., Zhou, L., Huang, T., Taha, M.F., He, Y., Qiu, Z., 2022. Development of an automatic pest monitoring system using a deep learning model of dpenet. Measurement 203, 111970.

Zhong, Y., Gao, J., Lei, Q., Zhou, Y., 2018. A Vision-Based Counting and Recognition System for Flying Insects in Intelligent Agriculture. Sensors (Basel, Switzerland) 18 (5).