# Automatic Recognition of Bee Larvae for Royal Jelly Production on Embedded Systems

Yanis Blot–EL Mazouzi[1], Umut Gumus[1], Annaelle Delamarche[1], Arthur De Macédo[1], Sebora Memolla[2], and Baptiste Magnier[2]

[1]IMT Mines Ales, Ales, France
[2]EuroMov Digital Health in Motion, Univ Montpellier, IMT Mines Ales, Ales, France
{yanis.blot-el-mazouzi, umut.gumus, annaelle.delamarche, arthur.de-macedo}@etu.mines-ales.fr
{sebora.memolla, baptiste.magnier}@mines-ales.fr

**Abstract.** This study explores two complementary approaches for the automatic detection of bee larvae in hive frame images, in the context of royal jelly production. The first approach is an end-to-end YOLOv8s detector that jointly performs localisation and classification. The second is a hybrid pipeline that decouples geometric circle extraction (via the Generalized Hough Transform) from learned classification (MobileNetV2). The two approaches are evaluated on a small, severely imbalanced dataset (142 images, 334 instances of the target class out of 9,221 annotated cells, 1:27 class ratio). We report per-class precision, recall, F1 score and inference time, and provide a fine-grained diagnostic of the residual errors. The end-to-end detector achieves F1 = 0.62 with an inference time of 80 ms per image; the hybrid pipeline achieves F1 = 0.66 at the cost of a substantially higher latency. We discuss the trade-offs between data requirements, accuracy, and inference latency, and identify actionable directions for future work.

**Keywords:** queen bee larvae detection · royal jelly · small object detection · YOLO · Hough transform · edge computing

## 1 Introduction

Royal jelly, a substance secreted by nurse bees to feed the queen and young larvae, is a highly valued hive product due to its nutritional properties and stimulating effects [4]. Its production relies on a labour-intensive, high value-added step: the selection of royal larvae and their transfer into artificial queen cups. These larvae are then continuously fed with royal jelly throughout their development to become queens. This process enables a more efficient production of royal jelly, concentrated within a series of artificial cups. In France, this task is costly, and 95% of royal jelly is imported. Automating this operation therefore represents a significant strategic and economic challenge for the sector.

The automation of this operation requires a computer vision system capable of reliably identifying and localising suitable larvae within hive frame images prior

to their collection by a three-axis robotic arm. Each acquired image constitutes a partial view of the frame, captured under potentially variable lighting and orientation conditions. The detection task is formulated as a binary classification problem distinguishing good larvae from all other cell contents, embedded within an object detection pipeline that must additionally provide spatial coordinates for each candidate cell.

## 2 Related Work

### 2.1 Automation of bee larvae picking

To our knowledge, only one published study directly addresses the automation of bee larva picking for royal jelly production. Güneş [1] developed a three-axis robotic system coupled with a Raspberry Pi 3B for image acquisition and inference. For larva detection, the author trained a MobileDetSSD [5] model on a dataset of 1,356 images. The system achieved an inference time of 2.75 seconds per image and a larva transfer success rate of approximately 50%. Although the detection results are promising, two limitations are particularly relevant to our work. First, the inference time represents a bottleneck for industrial throughput, as it contributes to a total machine cycle time of approximately 20 seconds. Second, the dataset used in that study is substantially larger than the one available in the present work, which contains only 142 labelled images. This difference raises the question of whether an end-to-end detection model can be trained reliably under stronger data constraints.

### 2.2 Object detection on embedded systems

Beyond the specific domain of apiculture, the broader challenge of deploying object detection models on resource-constrained hardware has received significant attention. Alqahtani et al. [2] provide a benchmark of state-of-the-art object detection architectures, including YOLOv8 [11] (Nano, Small, Medium), EfficientDet Lite (Lite0, Lite1, Lite2) [6], and SSD variants (SSD MobileNet V1, SSDLite MobileDet) [7], evaluated on Raspberry Pi 3, 4, and 5 [18] (with and without TPU accelerators) as well as the NVIDIA Jetson Orin Nano [17]. Performance is assessed along three axes: inference time, energy consumption per request, and mean Average Precision (mAP). Their results demonstrate a consistent accuracy-efficiency trade-off: lightweight models such as SSD MobileNet V1 achieve lower inference times and energy consumption, while higher-accuracy models such as YOLOv8 Medium bring substantially greater computational cost. No architecture dominates across all criteria simultaneously, the optimal choice depends on the constraints imposed by the deployment context.

### 2.3 Positioning of the present work

The two studies reviewed above converge on a shared limitation that directly motivates our approach. On the one hand, Güneş [1] demonstrated that larva

detection is tractable with a compact end-to-end model, but required a dataset approximately ten times larger than ours, and produced an inference time incompatible with high-cadence operation. On the other hand, Alqahtani et al. [2] confirmed that no existing detection architecture resolves the accuracy-latency trade-off on Raspberry Pi class hardware without compromise. Both constraints, data scarcity and embedded inference budget, argue against a standard end-to-end detection approach. This study therefore investigates a hybrid pipeline that replaces learned spatial localisation with deterministic geometric extraction via the Generalized Hough Transform [3], reducing the learning problem to binary classification on individual cell images. This decomposition is intended to lower the data requirements for the learning component while maintaining compatibility with embedded deployment constraints.

## 3 Methodology

### 3.1 Method Overview

Two complementary approaches are evaluated and compared. The first is an end-to-end object detection model based on YOLOv8s [11], which jointly performs localisation and classification in a single forward pass. The second is a hybrid pipeline that decomposes the task into two deterministic stages: cell candidates are first extracted through the Generalized Hough Transform [3] exploiting the circular geometry of brood cells, then classified individually by a MobileNetV2 network [8]. The first approach serves as a learned baseline; the second isolates the classification problem from the localisation problem in order to reduce the learning burden and evaluate whether deterministic localisation can help in a low-data setting.

### 3.2 Dataset

The dataset consists of 142 high-resolution photographs (2592 $\times$ 1944 pixels) of bee frames, captured under varying lighting conditions during two distinct sessions. Each image contains between 30 to 90 cells, annotated by a domain expert as circles parameterized by their centre $(c_x, c_y)$ and radius $r$. In total, the dataset contains 9,221 annotated cells, which are categorised into four classes: *good_larva* (young healthy larva suitable for transfer), *large_larva* (older or oversized larva), *egg* and *unusable_cell* (non-exploitable cells, including empty cells, debris, or reflections).

An additional set of 11 cells labelled *unlabeled* correspond to candidates produced by an automated detector that were never validated by the human annotator. These are excluded from training and evaluation.

The class distribution is highly imbalanced. The target class *good_larva* comprises 334 instances (3.6% of all annotated objects), compared with 8,887 instances (96.4%) for the three other classes combined. Reduced to a binary classification task (*good_larva* vs. *other*), this corresponds to a class ratio of

approximately 1:27, which is a defining constraint of this study and motivates several design choices throughout the pipeline, as shown in Figure 1.

A systematic audit of the annotations was performed prior to training. Three indicators were monitored: objects whose bounding circle exceeds the image boundary (3,222 cases, statistically expected from the photographic framing of densely packed cells), objects whose centre lies outside the image (11 cases, all in the *other* class and therefore not affecting the target class), and the radius distribution (median around 150 px, with no abnormal outliers among the target class). The audit confirmed that the dataset is geometrically clean for the binary classification objective.

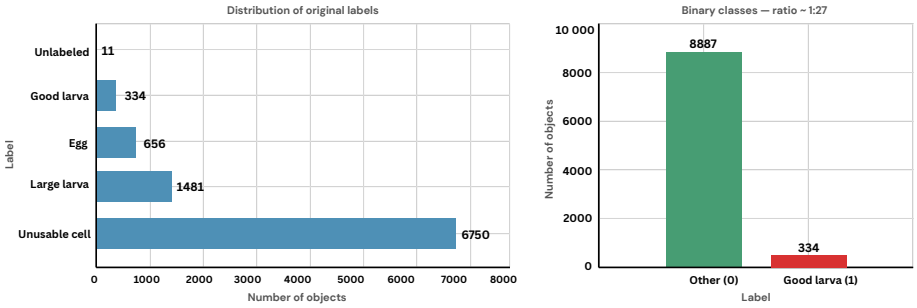


Fig. 1: Class distribution of the dataset. Left: original four-class and the additional *unlabeled* candidates, highlighting the dominance of *unusable_ cell* (6,750 instances). Right: binary reformulation with a class ratio of approximately 1:27 in favour of the *other* class.

### 3.3 Approach 1 – Object Detection Model (YOLOv8)

The first approach treats the problem as standard object detection. We adopt YOLOv8s [11], a single-stage anchor-free detector with approximately 11 million parameters. The model is initialised with COCO pre-trained weights [10], which provide general visual features such as edges, gradients and simple shape patterns. Although COCO is different from our bee-cell images, these features still provide a useful starting point for larvae detection. The complete preprocessing and training workflow is summarised in Figure 2.

**Annotation conversion.** The original annotations are circular. They are converted to YOLO bounding-box format by circumscribing each circle with a square of side $2r$ centred on $(c_x, c_y)$. Coordinates are normalised to $[0, 1]$ and clamped at image boundaries to handle partially out-of-frame cells. This conversion preserves all object centres and accepts a slight over-coverage of background pixels in the corners of the bounding box, which is absorbed by data augmentation during training.

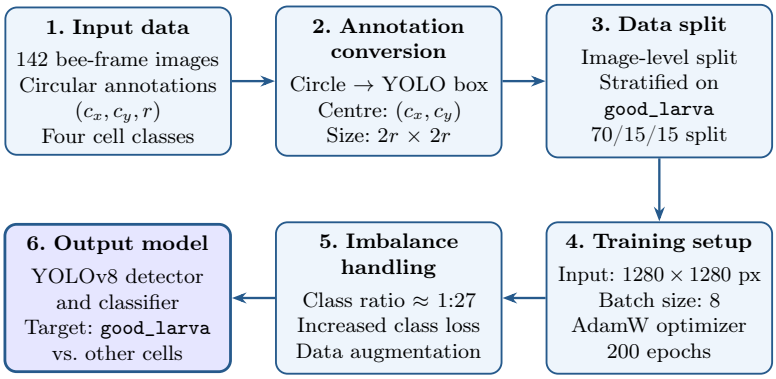


Fig. 2: YOLOv8-based preprocessing and training pipeline used for bee-cell detection and *good_larva* classification.

**Train–validation–test split.** Splitting is performed at the image level, not at the object level, to prevent data leakage between visually similar neighbouring cells of the same image. Since only 75 of the 142 images contain at least one *good_larva*, the split is stratified according to the presence of the target class. The final partition is 70/15/15, with 99 images for training, 21 for validation and 22 for testing. This strategy preserves a comparable proportion of positive images across the three subsets.

**Training configuration.** Images are processed at a resolution of $1280 \times 1280$ pixels rather than the default $640 \times 640$ pixels. This choice is motivated by the median radius of the target objects: at 640, larvae would be reduced to approximately 40 px in diameter, which is insufficient to discriminate visually similar classes such as *good_larva* and *large_larva*. At 1280 pixels, the effective size is about 80 px, which preserves more visual detail. The batch size is set to 8 to fit the 15 GB VRAM of the Tesla T4 GPU.

We use AdamW optimizer [12] with an initial learning rate of $10^{-3}$, a final-to-initial learning rate ratio of $10^{-2}$, and a weight decay of $5 \times 10^{-4}$. The first ten backbone layers are frozen during the first epochs to preserve the COCO features, then fine-tuned progressively. Training is run for up to 200 epochs with early stopping and a patience of 40 epochs.

**Imbalance mitigation.** The dataset has a strong class imbalance, with an approximate 1:27 ratio between *good_larva* and the other classes. To reduce this effect, three mechanisms are used. First, the classification loss weight `cls` is increased to 1.0, so class errors have a stronger impact during training. Second, `copy_paste=0.3` is used to add annotated objects into other training images and increase the diversity of minority-class examples. Third, horizontal and vertical flips are applied with a probability of 0.5 each, since brood cells do not have a fixed orientation.

### 3.4 Approach 2 – Hough Transform and CNN Classification

We propose the following pipeline shown in Figure 3 to take advantage of the geometrical structure of the images.

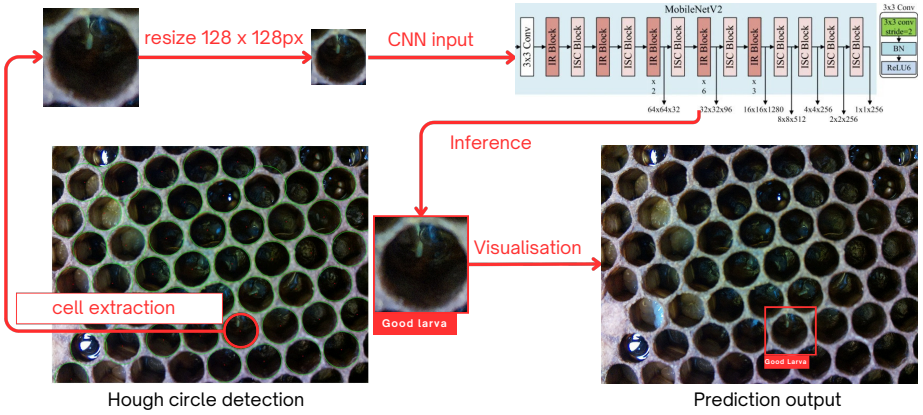


Fig. 3: Overview of the Hough-based extraction and MobileNetV2 [9] classification pipeline.

**Cell Extraction via Hough Transform.** The Hough Circle Transform [3] for circle detection is applied directly to the original RGB image, without prior preprocessing. To reduce computational cost, each image is downscaled by a factor of 0.75 before detection. The resulting circle parameters $(c_x, c_y, r)$ are then remapped to the original image coordinate space for sub-image extraction.

Circle detection is performed with OpenCV `cv2.HoughCircles`. The main parameters are reported in Table 1. For each detected circle with centre $(c_x, c_y)$ and radius $r$, a square crop of size $2r \times 2r$ is extracted and passed to the classifier.

Table 1: Parameters used for circle detection with `cv2.HoughCircles`.

| **Parameter** | **Value** |
|---|---|
| Accumulator resolution ratio, *dp* | 2 |
| Minimum inter-circle distance, `minDist` | 120 px |
| Canny upper threshold, `param1` | 50 |
| Accumulator threshold, `param2` | 30 |
| Minimum radius, `minRadius` | 100 px |
| Maximum radius, `maxRadius` | 150 px |
| Maximum circles per image | 50 |

**Cell-level Dataset.** This pipeline converts the original dataset (142 images) into a larger dataset of 2,471 images of individual cells extracted from the labels. Each sub-image is labelled according to the annotation of the corresponding cell in the source image. The resulting dataset contains 2,137 samples of class *other* (86.5%) and 334 samples of class *good_larva* (13.5%), which indicates that the dataset is unbalanced. The dataset is split into a training set (1,976 images: 1,702 *other*, 274 *good_larva*) and a validation set (495 images: 435 *other*, 60 *good_larva*). Examples of the extracted individual cell crops are shown in Figure 4.

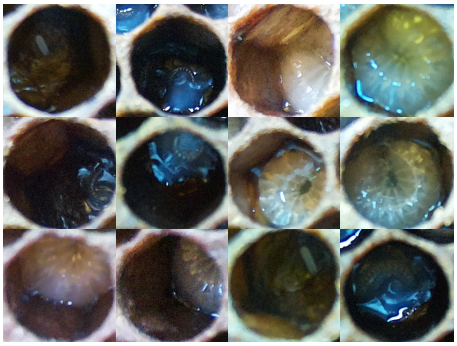

Fig. 4: Samples of individual cell crops obtained after the extraction step.

**Classification Model (MobileNetV2).** Each extracted cell image is resized to $128 \times 128$ pixels and normalised using the ImageNet [13] mean and standard deviation, with $\mu = [0.485, 0.456, 0.406]$ and $\sigma = [0.229, 0.224, 0.225]$. During training, we apply data augmentation (horizontal and vertical flips, random rotations up to 20°, colour jitter on brightness, contrast and saturation).

The classification model is a MobileNetV2 [8] pretrained on ImageNet dataset. At first, all convolutional layers are frozen. Then, the last five feature blocks, `features[-5:]`, are unfrozen to adapt the model to honeycomb cell images. The original classification head is replaced by a dropout layer with rate 0.3, followed by a fully connected layer that maps the 1280-dimensional feature vector to two classes: *other* and *good_larva*.

Training uses the Adam optimizer with differentiated learning rates: $lr = 10^{-3}$ for the classification head and $lr = 10^{-5}$ for the unfrozen backbone blocks. The training loss is standard cross-entropy.The selected model was trained for 35 epochs.

### 3.5 Evaluation Metrics

Due to the strong class imbalance, global accuracy is not used as the main metric. In the original object-level distribution, a model predicting only the majority class

would reach about 96% accuracy while detecting no *good_ larva*. We therefore report per-class precision, recall, F1 score and mAP@50 for the YOLOv8s model. For both pipelines, inference time per image is also reported.

The main evaluation metrics are defined as follows:

$$P = \frac{TP}{TP + FP} \quad (1) \qquad R = \frac{TP}{TP + FN} \quad (2)$$

$$F1 = 2\,\frac{PR}{P + R} \quad (3) \qquad \mathrm{mAP}_{50} = \frac{1}{C}\sum_{c=1}^{C} AP_{c,50} \quad (4)$$

where $P$ and $R$ refer to precision and recall, respectively, while $TP$, $FP$, and $FN$ are the numbers of true positives, false positives, and false negatives. In equation (4), $C$ is the number of classes and $AP_{c,50}$ is the average precision of class $c$ computed at an IoU threshold of 0.5.

For the YOLOv8s detection pipeline, a prediction is considered correct when its intersection-over-union with a ground-truth box satisfies IoU $\geq 0.5$, with IoU $= |B_p \cap B_{gt}|/|B_p \cup B_{gt}|$. Here, $B_p$ and $B_{gt}$ are the predicted and ground-truth bounding boxes, respectively.

The mAP@50 metric gives a complementary view of detection performance across confidence thresholds. For both pipelines, inference time per image is measured as the average processing time over the evaluated images.

In the application context, recall on the *good_ larva* class is the most important metric. A missed suitable larva directly reduces production, while a false positive mainly adds a short manual verification step.

## 4 Results

### 4.1 YOLOv8s end-to-end detection

After 160 effective epochs (early stopping triggered), the YOLOv8s model achieves the per-class metrics summarised in Table 2 on the test set. The learning curves shown in Figure 5 show a stable training behaviour, with training and validation losses decreasing in a similar way. The validation mAP starts to plateau around epoch 100, suggesting that the model reaches the limit of what can be learned from the available data rather than showing clear signs of overfitting.

Table 2: YOLOv8s per-class metrics on the test set (22 images, 1,469 objects).

| **Class** | **Precision** | **Recall** | **F1** | **mAP@50** |
|---|---|---|---|---|
| *other* | 0.894 | 0.963 | 0.927 | 0.957 |
| *good_ larva* | 0.637 | 0.608 | 0.622 | 0.640 |
| **Overall** | 0.765 | 0.785 | – | 0.798 |

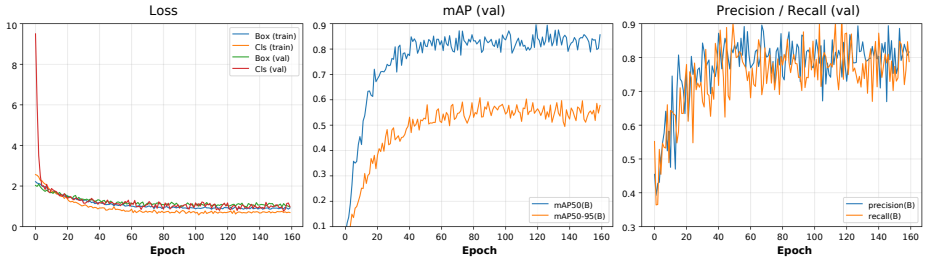


Fig. 5: Learning curves of the YOLOv8s model over 160 epochs. Left: box and classification losses for training and validation, decreasing in parallel without divergence. Centre: mAP@50 and mAP@50-95 on the validation set, plateauing after epoch 100. Right: precision and recall on the validation set, oscillating around 0.75–0.80 once converged. Early stopping is triggered after 40 epochs without improvement.

The majority class is detected with high reliability, reaching an F1 score of 0.93 as expected, given its abundance in the training data. In contrast, the performance on the target class remains lower, with an F1 score of 0.62, due to class imbalance. More specifically, among the 75 ground-truth *good_larva* instances in the test set, 46 are correctly detected, 29 are missed as false negatives, and 26 false positives are produced.

**Confusion Analysis.** The normalised confusion matrix in Figure 6 shows that the main limitation of YOLOv8s is not localization, but class discrimination. For the target class, only 34% of the ground-truth *good_larva* instances are classified correctly, while 66% are assigned to the *other* class. Since the *background* row is empty for the *good_larva* column, the model usually detects the larvae but confuses their developmental stage. The dominant error therefore comes from the visual similarity between *good_larva* and *large_larva*, rather than from missed detections.

We additionally note that the apparent gap between the recall on *good_larva* (0.61) and the diagonal cell of the confusion matrix (0.34) comes from the way the two quantities are computed. The recall counts a ground-truth larva as detected, when at least one correct prediction matches it, while the confusion matrix counts individual predictions. When the model emits multiple overlapping bounding boxes on the same larva with conflicting class labels, the recall remains tolerant while the matrix is penalized. Class-agnostic Non-Maximum Suppression could reduce this effect by removing duplicate boxes regardless of their predicted class.

**Three-class refinement.** Based on this error analysis, we trained a refined version of the model with three explicit classes: *other*, *good_larva*, and *large_larva*. The binary task was recovered by post-inference by merging *large_larva* into *other*. This setting was combined with $\times 3$ oversampling of training images containing the minority class and an increased classification loss weight, with `cls=2.0`.

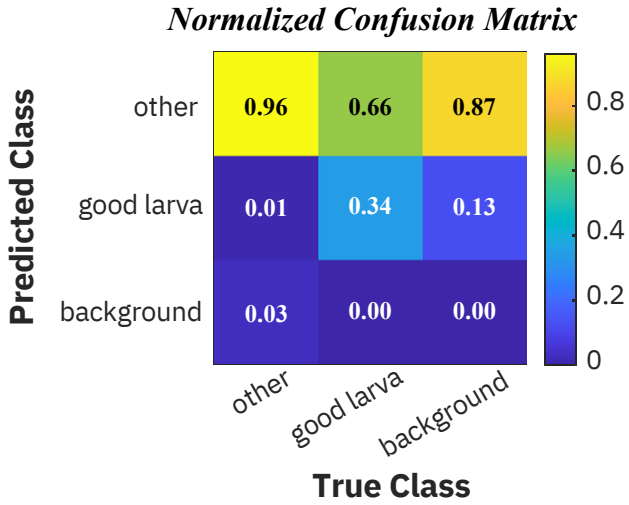


Fig. 6: Normalised confusion matrix of the YOLOv8s model on the test set. Rows correspond to predictions and columns to ground truth. The dominant error is the confusion of *good_larva* with the *other* class, while the empty *background* row for *good_larva* indicates that errors mainly come from misclassification rather than missed detections.

This configuration improves the global mAP@50 from 0.798 to 0.822, suggesting better class separation. However, at a fixed confidence threshold of 0.25, the precision and recall of *good_larva* decrease compared to the binary baseline. This indicates a calibration shift caused by the stronger rebalancing strategy. The refined model therefore requires confidence-threshold tuning, rather than evaluation at a single fixed threshold.

**Inference Time.** At an input size of 1280, YOLOv8s reaches an average inference time of $80.8 \pm 7.8$ ms per image on a Tesla T4 GPU, with a median of 77.5 ms. This corresponds to approximately 12 frames per second. These values cannot be directly transferred to an embedded platform such as a Raspberry Pi 4, where inference would be slower. For deployment, model export to ONNX [14] or TFLite [15] and INT8 quantization [16] would be required to reduce latency.

### 4.2 Hough + MobileNetV2 hybrid pipeline

Because of the class imbalance, global validation accuracy is not considered a reliable primary metric. Instead, Table 3 summarises precision, recall, and F1 score for the *good_larva* class across four confidence thresholds.

As the threshold increases, precision improves at the cost of recall, which is the expected behaviour. The threshold $t = 0.75$ is retained as the operating point, as it yields the highest F1 score (0.66) while maintaining an acceptable recall of 0.70.

Table 3: MobileNetV2 per-threshold metrics on the validation set for the *good_larva* class.

| Threshold | Precision | Recall | F1 |
|---|---|---|---|
| 0.65 | 0.57 | 0.77 | 0.65 |
| 0.70 | 0.59 | 0.73 | 0.65 |
| **0.75** | **0.63** | **0.70** | **0.66** |
| 0.80 | 0.65 | 0.65 | 0.65 |

**Confusion Analysis.** At the selected threshold $t = 0.75$, the MobileNetV2 [8] classifier produces 42 true positives, 18 false negatives, 25 false positives, and 410 true negatives on the validation set (495 images: 435 *other*, 60 *good_larva*). This corresponds to a recall of 70% for the *good_larva* class. However, 18 suitable larvae are still classified as *other*, which represents the main source of error and the most critical failure case for the application.

The model also produces 25 false positives. Although these errors are not negligible, they have a lower operational cost, since they would mainly require manual verification rather than causing a direct production loss. The corresponding confusion matrix is shown in Figure 7.

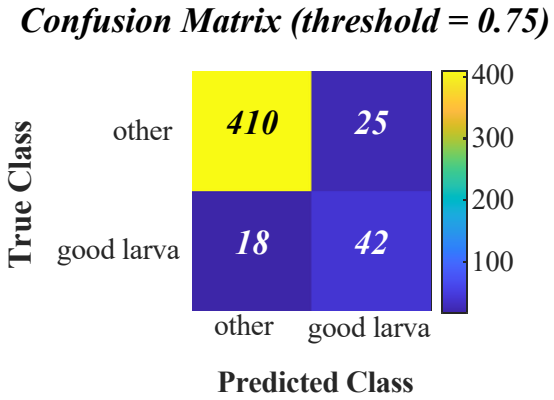


Fig. 7: Confusion matrix of the MobileNetV2 classifier at the selected threshold $t = 0.75$ on the validation set.

**Inference Time.** The average inference time per image is 1,227.71 ms for the Hough circle detection stage and 190.76 ms for the CNN classification stage, yielding a total pipeline time of 1,418.47 ms (approximately 1.4 s), measured on a Tesla T4 GPU. The Hough detection step is the dominant bottleneck, accounting for 86% of the total pipeline latency.

# 5 Discussion

## 5.1 Comparison of the Two Approaches

The two pipelines reflect different design choices and deployment constraints. Table 4 summarizes their main technical characteristics observed in our experiments. The YOLOv8s pipeline performs detection and classification in a single forward pass, while the hybrid approach separates cell localisation from classification by using Hough-based extraction followed by MobileNetV2.

Table 4: Technical comparison of the two approaches.

| **Criterion** | **YOLOv8s** | **Hough + MobileNetV2** |
|---|---|---|
| Type of task | Detection | Classification on crops |
| Trainable parameters | $\sim$ 11 M | $\sim$ 3.5 M |
| Pretraining | COCO | ImageNet |
| Localisation | Learned | Deterministic |
| Multi-object handling | Single forward pass | Sequential per crop |
| mAP@50 | 0.80 | – |
| Inference time (T4) | 80 ms | 1,418 ms |

The main difference between the two approaches is the way localisation is handled. YOLOv8s learns both localisation and classification directly from the annotated images, which makes the pipeline compact and fast at inference time. In contrast, the hybrid method uses the circular geometry of honeycomb cells to extract candidates before classification. This reduces the learning problem to cell-level classification, but introduces an additional preprocessing stage.

The trade-off, however, is dominated by inference time. The hybrid pipeline is approximately 17 times slower than the end-to-end detector (1,418 ms vs 80 ms per image), almost entirely due to the Hough circle detection step, which alone consumes 86% of the total pipeline latency. This observation is qualitatively consistent with the inference time of 2.75 s reported by Güneş [1] on a Raspberry Pi 3B, where similar geometric preprocessing was used. For a real-time grafting system, this latency is incompatible with the typical industrial cycle target of a few seconds per frame, while the YOLOv8s baseline is already operating at 12 FPS on GPU and could plausibly reach 1–2 FPS on a Raspberry Pi 4 after INT8 [16] quantization, marginally compatible with the manual cadence of an apiculturist's grafting workflow.

## 5.2 Diagnostic of the dominant failure mode

The most informative finding of this study is not the absolute metric level but the *nature* of the residual errors observed on the YOLOv8s model. The empty

*background* row of the confusion matrix on the *good_larva* column establishes that the detector successfully localizes larvae across the dataset; the open issue is the discrimination between *good_larva* and the visually similar *large_larva*, which differ only by approximately 12–24 hours of larval development. This distinction is intrinsically subtle even for a human expert, and the inter-annotator agreement ceiling on this specific frontier is empirically known to lie around 93–95% on similar fine-grained biological tasks. Any further model-side improvement is therefore bounded by the quality and consistency of the annotations themselves.

The MobileNetV2 classifier, although trained on a more balanced cell-level dataset (87%/13% rather than 96.4%/3.6%), exhibits a similar dominant error mode: 18 false negatives out of 60 ground-truth positives at the chosen operating threshold. Both approaches converge on the conclusion that the bottleneck is not the model architecture but the limited number of *good_larva* examples available for learning the visually subtle frontier with *large_larva*.

### 5.3 Levers for improvement

Three families of levers may be considered to advance these baselines.

**Data-centric.** Enriching the dataset with additional images containing the minority class is likely to be the most impactful direction. A larger and more diverse set of *good_larva* examples would help the model learn the visual boundary between *good_larva* and *large_larva* more reliably. This hypothesis should be validated in future work by progressively increasing the number of minority-class samples and measuring the effect on recall and F1 score. Targeted collection of frames containing both classes side by side would be particularly useful, as it would force the model to learn the discriminative features rather than relying on contextual cues.

**Architectural.** Higher-capacity variants such as YOLOv8m or YOLOv9 models may improve fine-grained discrimination, but their return on investment is conditioned on the availability of additional data; on the current dataset, they are likely to overfit. A complementary direction is to exchange bounding-box detection for instance segmentation (YOLOv8-seg), which would produce circular masks more faithful to the actual cell geometry than the inscribed square approximation we currently use.

**Operational.** Tuning the confidence threshold at inference allows trading precision against recall in a way that aligns with the asymmetric cost structure of the application, where a missed larva is more costly than a false alarm. For the v3 model, our experiments indicate that a threshold lower than the standard 0.25 recovers the gain in discriminative capacity that is masked by the score calibration shift. Activating class-agnostic Non-Maximum Suppression (`agnostic_nms=True`) would additionally suppress duplicate boxes that arise when the model hesitates between two similar classes on the same object, which we identified as a contributor to the gap between recall and confusion-matrix diagonal.

### 5.4 Limits common to both approaches

A common limitation of both approaches is the absence of cross-validation. With only 142 images, results computed on a single test partition of 22 images carry a non-negligible statistical uncertainty, a single annotation error can shift F1 by several points. A 5-fold stratified cross-validation, although approximately five times more expensive computationally, would yield confidence intervals on the metrics and is a recommended addition for any subsequent iteration intended for industrial deployment.

A second shared limit is the absence of an end-to-end on-device benchmark. All inference times reported in this study are measured on Tesla T4 GPU; the corresponding latencies on the actual target hardware (Raspberry Pi 4 or Jetson Nano) remain to be characterized, ideally after model export and quantization. Without this measurement, the operational viability of either pipeline cannot be conclusively asserted.

## 6 Conclusion

This study has investigated two complementary approaches to the automatic detection of bee larvae for royal jelly production, in a deliberately data-constrained setting (142 annotated images, 334 instances of the target class, class ratio of 1:27). The end-to-end YOLOv8s detector achieves a test F1 score of 0.62 and mAP@50 of 0.80 on the binary task, with an inference time of approximately 80 ms on a Tesla T4 GPU. The hybrid pipeline combining a Generalized Hough Transform for cell extraction with a fine-tuned MobileNetV2 classifier achieves a slightly higher validation F1 of 0.66, at the cost of an inference time of approximately 1.4 seconds, more than seventeen times slower than the end-to-end approach.

Beyond the absolute metrics, the principal contribution of this work is the diagnosis of the dominant failure mode shared by both approaches: the model does not miss larvae through detection failure but through fine-grained classification confusion with visually adjacent developmental stages. This insight directly informs the prioritization of future work, in which targeted dataset enrichment of the minority class emerges as the single most impactful direction, complemented by architectural scaling and threshold-based inference tuning.

From an industrial standpoint, both approaches provide useful baselines for prototyping. The YOLOv8s baseline is the more promising candidate for embedded deployment given its substantially lower latency, with a clear upgrade path via ONNX export and INT8 quantization. The hybrid pipeline, although slower, demonstrates that a deterministic geometric prior can compensate for a small training set and may remain attractive for offline analysis or as a second-stage validator on top of a detector. The two pipelines are not mutually exclusive: a learned end-to-end detector for high-throughput operation, and a deterministic-then-learned hybrid for offline verification, offer a flexible operational framework adaptable to the future scale of the data acquisition effort.

## References


1. Güneş, H.: Development of AI Based Larvae Transfer Machine for Royal Jelly Production. Journal of Agricultural Sciences **29**(1), 209–220 (2023). https://doi.org/10.15832/ankutbd.870464
2. Alqahtani, D.K., Cheema, M.A., Toosi, A.N.: Benchmarking Deep Learning Models for Object Detection on Edge Computing Devices. In: Gaaloul, W., Sheng, M., Yu, Q., Yangui, S. (eds.) ICSOC 2024, LNCS, vol. 15404, pp. 145–160. Springer, Singapore (2025). https://doi.org/10.1007/978-981-96-0805-8_11
3. Duda, R.O., Hart, P.E.: Use of the Hough Transformation to Detect Lines and Curves in Pictures. *Communications of the ACM* **15**(1), 11–15 (1972).
4. Bogdanov, S.: The Royal Jelly Book. Bee Product Science (2011).
5. Xiong, Y., Liu, H., Gupta, S., Akin, B., Bender, G., Wang, Y., Kindermans, P.-J., Tan, M., Singh, V., Chen, B.: MobileDets: Searching for Object Detection Architectures for Mobile Accelerators. In: IEEE/CVF CVPR, pp. 3825–3834 (2021).
6. Tan, M., Pang, R., Le, Q.V.: EfficientDet: Scalable and Efficient Object Detection. In: IEEE/CVF CVPR, pp. 10781–10790 (2020).
7. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., Berg, A.C.: SSD: Single Shot MultiBox Detector. In: ECCV, LNCS, vol. 9905, pp. 21–37. Springer, Cham (2016).
8. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.-C.: MobileNetV2: Inverted Residuals and Linear Bottlenecks. In: CVPR, pp. 4510–4520 (2018).
9. Tsai, C.Y., Su, Y.K.: MobileNet-JDE: A Lightweight Multi-Object Tracking Model for Embedded Systems. Multimedia Tools and Applications **81**(7), 9915–9937 (2022).
10. Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft COCO: Common Objects in Context. In: ECCV, LNCS, vol. 8693, pp. 740–755. Springer, Cham (2014).
11. Jocher, G., Chaurasia, A., Qiu, J.: YOLO by Ultralytics. https://github.com/ultralytics/ultralytics, last accessed 15 May 2026.
12. Loshchilov, I., Hutter, F.: Decoupled Weight Decay Regularization. In: ICLR, (2019).
13. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: IEEE CVPR, pp. 248–255 (2009).
14. ONNX Community: ONNX: Open Neural Network Exchange. https://github.com/onnx/onnx, last accessed 15 May 2026.
15. TensorFlow Authors: TensorFlow Lite. https://www.tensorflow.org/lite, last accessed 15 May 2026.
16. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D.: Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In: IEEE CVPR, pp. 2704–2713 (2018).
17. NVIDIA, Jetson Orin Nano Developer Kit User Guide (Hardware Specification). https://developer.nvidia.com/embedded/learn/jetson-orin-nano-devkit-user-guide/hardware_spec.html, last accessed 15 May 2026.
18. Raspberry Pi Ltd.: Raspberry Pi Computer Hardware. https://www.raspberrypi.com/documentation/computers/raspberry-pi.html, last accessed 15 May 2026.