Real-Time Egg Detection Using Edge Computer Vision

Nicholas Hadjisavvas
Algolysis Ltd
Nicosia, Cyprus
nicholas.hadjisavvas@algolysis.com

Nicolas Nicolaou
Algolysis Ltd
Nicosia, Cyprus
nicolas@algolysis.com

Efstathios Stavrakis
Algolysis Ltd
Nicosia, Cyprus
stathis@algolysis.com

***Abstract*—The adoption of Artificial Intelligence (AI) in agriculture and animal husbandry has accelerated in recent years, driven by the versatility and relatively low costs for development and deployment of smart systems. However, many farms still rely on aging equipment and manual labour rendering these innovations inapplicable. In turn, the inability to harness AI and modernise operations may pose an existential risk. To address this challenge, we advocate for retrofitting existing machinery with AI-based modules as a practical alternative. In this paper, we demonstrate how a poultry egg grading machine can be enhanced with smart capabilities through the integration of deep learning and low-cost commodity edge hardware to enable precise egg counting. We present the methodology and algorithms behind this system that enables real-time processing while maintaining high accuracy. In a limited set of experiments, we demonstrated that the Raspberry Pi 5 (RPi5) running the EfficientDet-lite0 model performed just as well as a desktop with an NVIDIA GPU (graphics processing unit), accurately counting all the eggs it was presented with.**

Keywords-egg counting; smart retrofitting; deep learning.

I. Introduction

Egg production remains to this day as one of the most important farming enterprises providing a steady supply of a highly nutritious and affordable food source. Poultry egg producers are required to follow specific processes for egg handling, processing, labeling, and marketing to ensure the safety and quality of eggs reaching consumers. These are labour-intensive processes that are assisted by purpose-built equipment, such as egg grading and sorting machines, packaging, storage, etc. However, replacing existing equipment and processes can be costly, time consuming and may cause operational disruption. To reduce the cost of modernising existing poultry egg production facilities with little to no interruption the idea of retrofitting can bring about significant gains [1], [2]. Instead of replacing the equipment farmers have learned to rely on, add-on digital devices can be introduced that provide new advanced capabilities.

This work explores the feasibility of ”smart retrofitting” for animal farm equipment. We design, implement, and deploy a low-cost, AI-based edge computing system that integrates with a traditional egg grading and sorting machine. The system automatically counts eggs using computer vision and classifies them based on the configuration of the egg sorting machine. This low-cost solution primarily benefits farms seeking to digitally transform on a limited budget (i.e., the proposed Raspberry Pi 5 system costs approximately one hundred euros).

AI-powered computer vision systems have been widely adopted in animal and food production industries, improving efficiency, product quality, and distribution speed [3], [4]. AI is expected to continue playing a key role in the agri-food industry's transformation [5], [6]. This success is largely due to the availability of pre-trained computer vision models. However, these models usually perform poorly for specialized field applications, such as egg detection, and require to be retrained or fine tuned. Furthermore, most AI models still require substantial computational resources to run in real-time, making them difficult to implement on low-end devices and deploy them in the field.

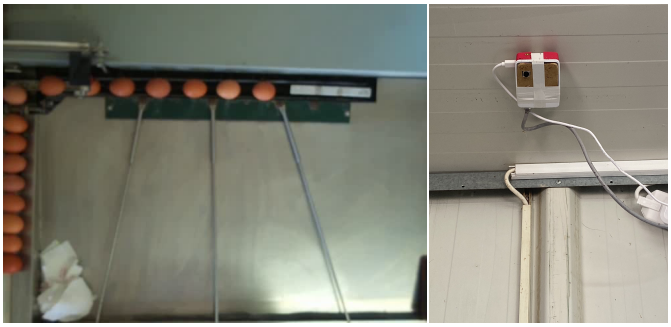
The system presented in this work demonstrates that with very limited computational resources, widely available AI models can be employed to improve operations in animal farms. Our system provides extremely accurate egg counts through a robust object detection algorithm enabling low-end single-board computers (e.g., the Raspberry Pi) to perform object detection and tracking in real time. The system's hardware is inexpensive (i.e., Raspberry Pi and the Pi Camera) and it can be trivially deployed in the field without expert knowledge.

The remainder of this paper is structured as follows: Section II provides information regarding the setup of the system in the environment that it is intended to be used. Section III provides a brief overview of related research that informed our approach. In Section IV, we outline our methodology and present our solution for egg counting at the edge. Section V details the experiments conducted to evaluate the system and discusses the results. Finally, Section VI provides a summary of the paper and highlights key conclusions.

II. Environment

The system developed in this work is based on a Raspberry Pi 5 single-board computer with a Pi Camera V2 module. It was deployed to a chicken farm with a Riva-Selegg Grader, configured to sort eggs into four weight classes (Extra Large, Large, Medium, and Small) with a single feeding lane, shown in Figure 1(a). As eggs move along the horizontal feeding lane, they drop into preconfigured collection areas when their weight exceeds the machine's preset value. These areas are slightly inclined, causing the eggs to roll towards the operator, following random paths, until collected by hand. The machine itself is purely mechanical, without digital features for counting or recording data, so egg counting is done manually by the operators.

The vision-based egg counting edge device was placed



(a) (b)

Figure 1: (a) A snapshot of the images obtained by our system. Egg weighting positions are below the eggs running horizontally and located above the egg grading zones separated with metal rods aligned vertically. (b) Ceiling-mounted egg counting Raspberry Pi 5 deployed above a Riva Selegg egg grading machine.

directly above and almost perpendicular to the egg grading machine at a distance of around 2m, as shown in Figure 1(b), in order to provide a top view of the feeding lane and the weighting positions of the eggs.

The system is composed of two subsystems running as independent services: (a) a back-end developed in Python and providing for image acquisition, object detection and tracking, egg counting, and (b) a front-end implemented in ReactJS providing a web interface for controlling the system and its parameters (e.g., calibration, start/stop egg detection, view/edit/confirm egg counts, etc.)

Once deployed, the system requires a simple calibration procedure (detailed in Section IV-E) and is then ready for use. Re-calibration is needed if either the egg detector or grading machine are adjusted.

III. Related work

In the past decade, the application of computer vision and AI in agriculture and animal husbandry has seen a significant increase. In poultry egg production, deep learning has been used to address egg grading [7], identify egg defects [8], [9] or assess freshness [10]. Egg counting, alongside detection, has also been a popular use case for deep learning based applications in the industry [11]–[14].

Automated egg counting using deep learning entails the training of a convolutional neural network to detect eggs first and then the development of a tracking and a counting algorithm to maintain eggs detected through time. Thus, we review some relevent work in object detection, tracking, alongside their application in edge computing.

A. Object detection Models

Ulaszewski et al. [12] conducted experiments comparing MobileNet-SSD (Single Shot Detector) v2, YOLOv3 (You Only Look Once), and Faster-RCNN (Region-based Convolutional Neural Network) for egg detection and counting on various hardware platforms. Using inference speed (fps) and counting accuracy as the primary performance metrics, their results demonstrated that MobileNet-SSD was the fastest and most reliable model under the specific conditions of their experiments.

Yang et al. [8] employed four different versions of RTMDet (Real-Time Models for object Detection) [15] models to perform egg detection for automated defect observation and sorting. While the focus was on a different egg-related task, all models still conducted pure egg detection. Among these, RTMDet-x demonstrated the highest accuracy, outperforming the other versions.

Subedi et al. [13] tested various YOLO (You Only Look Once) model versions for detecting floor eggs, while Luo et al. [16] enhanced a YOLOv5 model for detecting leaky eggs on a production line, achieving superior performance over YOLOv4 and F-RCNN models. Similarly, Vinod et al. [11] utilized a YOLO model for implementing an egg counting system.

B. Tracking

Tracking is a fundamental prerequisite for effective object counting. Tracking algorithms range from simple geometric approaches to more advanced deep learning-based methods, though the latter often come with increased computational demands.

Ulaszewski et al. [12] and Vinod et al. [11] used simple, yet effective, center-based tracking to pair with their detections. Shen et al. [17] also used a similar center location tracking approach to count people in elevators. Other algorithms like SORT [18] and DeepSort [19] are also used for object tracking (e.g Dinh et al. [20] used it for traffic counting). While algorithms such as SORT and DeepSORT are optimized for more robust tracking, it is crucial to account for their increased computational cost, particularly when designing edge applications where processing resources are limited.

C. Edge Applications

Maximizing efficiency and performance in real-time object detection and counting remains an open challenge. Chen et al. [21] reviewed the use of Deep Learning with edge computing, highlighting key issues such as latency, scalability, and privacy. They also focused on the challenges of deploying deep learning models on resource-constrained devices, such as the Raspberry Pi.

Tsu-Chuan et al. [17] have worked on a similar task of edge-based people counting in elevators using a MobileNet-SSD object detector and a line of interest counting strategy. They deployed their system on NVIDIA Jetson nano boards. Deployment of counting systems on the edge also has relevant applications in traffic management and monitoring. Duc-Liem Dinh et al. [20] have introduced a low-cost edge-based system utilising object detection for vehicle detecting, tracking and counting.

IV. Methodology

Our methodology involves four steps: (a) acquire real-time time images from a camera observing the egg feeding lane of the egg grading machine, (b) perform object detection

inference using a single-shot model, (c) compare detected objects with those of the previous image to track the objects, (d) use a set of predefined zones to count eggs of different grades, that are either defined interactively or automatically via a calibration step.

A. Image Acquisition

A simple camera module is used to obtain images in real-time. A standard resolution of 640 x 480 pixels is chosen to provide sufficient image quality for processing.

B. Object detection

Single-shot detection models [22]–[24] are regarded as state-of-the-art solutions for real-time object detection. However, achieving real-time performance on edge devices is challenging due to processing power limitations, which render many otherwise effective algorithms impractical for such environments.

The YOLO architecture, depending on the model size chosen, contains a number of parameters in the range of 3.2 to 68.2 million and require 8.7 to 257.8 billion floating point operations (GFLOPs) for a single network forward pass to detect objects in a single image. For reference, the Raspberry Pi 5 can reach around 34 GFLOPS [25], while the Raspberry Pi 4 is rated around 10 GFLOPS. An NVIDIA RTX 3070 Ti discreet GPU can reach 21.75 TFLOPS. It is therefore reasonable to expect that while object detection models can be run on all three hardware configurations, the lower-end Raspberry Pi 4 may have difficulties keeping up with real-time processing. On the other hand, a modern discrete GPU can easily handle larger detection models. This allows for verifying the performance of egg counting algorithms without the risk of reaching a processing power limit.

Since we are only interested in egg detection and counting, it is also necessary to consider the ability of these general purpose detectors to reliably detect eggs. In our tests, we noticed that these models either do not recognize eggs, or they need to be specifically trained with egg samples to be able to perform well.

These limitations highlighted the need for an object detection model with a lighter architecture. Such a model should deliver satisfactory results when trained with an appropriate dataset, without being as computationally intensive as the YOLO models. Lightweight object detection models designed for on-mobile or edge device inference are well-supported within the open-source community. Examples of these models include MobileNet-SSD [23], TinyYOLO, and EfficientDet [22]. Google's autoML provides a family of object detection models which include some light, mobile-sized versions. Probably the smallest model is EfficientDet-Lite0, which offers a good balance between performance and computational efficiency.

Although it has the lowest performance among all the EfficientDet models on the COCO dataset [26], EfficientDet-Lite0 is likely the best fit for our needs due to its lightweight and efficient design. Its documented Mean Average Precision (mAP) 26.41% [27] reflects its ability to generalize across a dataset with various object classes, many of which are irrelevant to our goal of recognizing just one class of objects (i.e., eggs). Therefore, the model was further trained and fine-tuned using a curated, custom egg dataset, as described in Section V.

C. Object tracking and counting

Efficient object tracking is essential for accurate counting across frames, requiring an algorithm with minimal computational demand and reliable results. Distance-based centroid trackers meet these needs by matching detected objects between frames using simple Euclidean distance calculations. The accuracy of the counting relies on correct tracking, as each detected object is assigned a unique ID to ensure it is only counted once.

Distance-based object tracking methods, such as the centroid tracker, have notable limitations, with their performance heavily influenced by factors like inference frequency (i.e., the number of frames processed per second by the detection model).

Low inference frequency poses a major challenge when running deep learning models on edge devices in real-time. This limitation can negatively affect centroid tracking algorithms, which rely on comparing an object's position between consecutive frames. Processing only a few frames per second while skipping others can degrade the algorithm's performance, as objects may be too far apart in time, leading to unmatched or mismatched objects.

D. Counting using Region Of Interest (ROI)

Detection and counting is performed at the feeding lane of the machine. This compartment of the egg grader is responsible for weighing each incoming egg (using weight springs placed along a mechanical conveyor belt at predefined different positions).

First, the feeding lane is divided into four zones, corresponding to extra large (XL), large (L), medium (M) and small (S). Zones are defined as polygonal areas by four points on the image plane, as shown in Figure 2. To count eggs in these zones we adopted a binning approach. We utilise the centroid tracker's results (which include the ID and the centre coordinates of each detected object for each frame) to execute the counting logic. A detected egg is added or removed from the count in any of the bins (zones) based on the location of its center. We use a simple point-in-polygon test to determine if an egg's centroid falls within the region of any given zone. As the eggs move along the feeding lane, eggs are reassigned to zones. As eggs drop from the feeding lane into the collection area of the machine they remain assigned to the last bin they have been detected in.

The pseudocode for the ROI egg counting process is provided in the algorithm shown in Figure 3.

E. Calibration: Automatic Zone Computation

The ROI-based counting algorithm relies on the definition of several detection zones. Although this process is a one-off

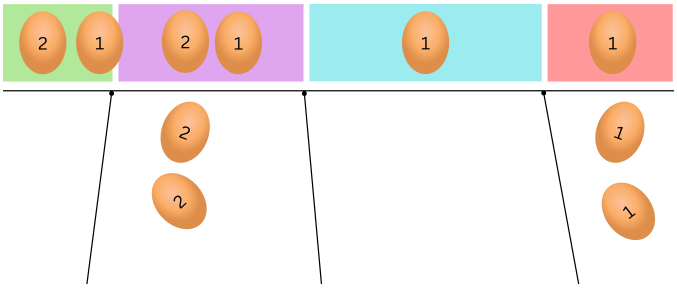


Figure 2: Schematic of the bin counting process using zones. Both eggs with IDs 1 and 2 are assigned to green zone (XL). As they move on the feeding lane (left to right), the eggs are reassigned from the previous bin to the current bin. Therefore, the egg with ID=1 is finally assigned to red bin (S), while the egg with ID=2 is assigned to purple bin (L).

```
// order by which egg grader sorts eggs
Require: zones = [XL,L,M,S]
Require: bins = dict()
  while isStreaming do
    frame = camera.getFrame()
    detections = model.detectEggs(frame)
    ids = assignIds(detections)
    for id, centroid in ids do
      for zone in zones do
        if pointInPolygon(centroid, zone.polygon) then
          bins[zone].append(id)
        end if
      end for
    end for
  end while
```

Figure 3: Pseudocode of the ROI-based egg counting algorithm.

procedure once the equipment has been deployed in the field, it is still necessary to provide the system with at least four points per zone. In our system the operator can connect to the edge device with a smartphone or tablet and configure the zones via a web interface interactively. However, this can be time consuming and tedious to perform in an animal farm and therefore we devised an effective automatic identification of the counting zones.

The calibration procedure shown in Figure 4 is as follows:

- At least 12 small eggs are passed through the egg grading machine. The choice of "small" eggs allows eggs to travel across the entire feeding lane.
- For each image, the centroid of each detected egg is extracted using the trained object detector.
- Locations of all centroids on the image plane are accumulated over the entire duration of the calibration procedure, as shown in Figure 4(a).
- During calibration, the grading machine sequentially moves each egg along the top horizontal feeding lane on the weighing springs. Based on the egg's weight, the springs may release it into the appropriate collection area. Eggs spend more time stationary on the weighing springs than in other positions or while rolling into the gathering area, leading to the formation of dense point clusters at these locations. To identify potential counting zones, we apply DBSCAN [28], [29], a density-based clustering algorithm, to group these closely packed points (see Figure 4(b)). The center of each cluster is then calculated by averaging the positions of the points and stored for further use.
- The computed centers of the clustered points are fitted to straight lines to identify the actual weighing locations of the egg sorting machines. First, a grayscale filter (Figure 4(c)) is applied to the image plotting all raw object centers, followed by a thresholding operation (Figure 4(d)) that converts the image to binary. This process removes areas with few or no detections. Subsequently, the Hough transform is used on the remaining centroids to extract a set of straight lines.
- These straight lines are then used in conjunction with the previously extracted cluster centers. The centers are fitted to each line. Lines with cluster centers matching the number of weighting positions of the machine are preserved, while the rest are discarded (see Figure 4(f)).
- For the specific egg grader used in this work, there are two weight springs for large and medium eggs each, and one spring for extra large and small eggs. A bounding box for each of the weight spring locations is computed and then the ones which are responsible for the same egg size are merged. The result of this is visualised in Figure 5.
- Masking Optimization: Once the counting zones are computed, a region that encompasses all zones is estimated. That region is a convex hull computed using the corners of all counting zones. Any pixel outside that region act as a mask (i.e., set to 0). This masking is a significant optimization for the egg detector, because no masked pixels are used when egg detecting, significantly improving accuracy and reducing processing.

V. System Evaluation & Results

A. Experimental Setup

Experiments were conducted to assess the performance of the counting system on three different hardware configurations: (i) a Desktop PC (OS: Windows 10 Pro, CPU: Intel Core i7-4790K, RAM: 32GB, GPU: NVIDIA GeForce RTX 3070 Ti), (ii) a Raspberry Pi 4 (OS: Raspbian, Model: 4B Rev 1.5, SoC: Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) @ 1.8GHz, RAM: 8GB), and (iii) a Raspberry Pi 5 (OS: Raspbian, Model: 5, SoC: Broadcom BCM2712, Quad core Cortex-A76 @ 2.4GHz, RAM: 8GB).

All devices were tested over two different object detection model architectures: (i) the EfficientDet-Lite0 [22], and (ii) the YOLOv8n [24]. Both models were trained with a custom dataset of 2,226 egg image samples (2065 training and 161 validation images). The dataset contains a variety of egg images covering different lighting conditions and angles, as well as different heights between the camera and the eggs. No augmentation was carried out. While the EfficientDet-Lite0 architecture is lightweight enough to run on all three devices, the YOLOv8n model was converted to the NCNN [30] high-performance neural network inference framework optimized

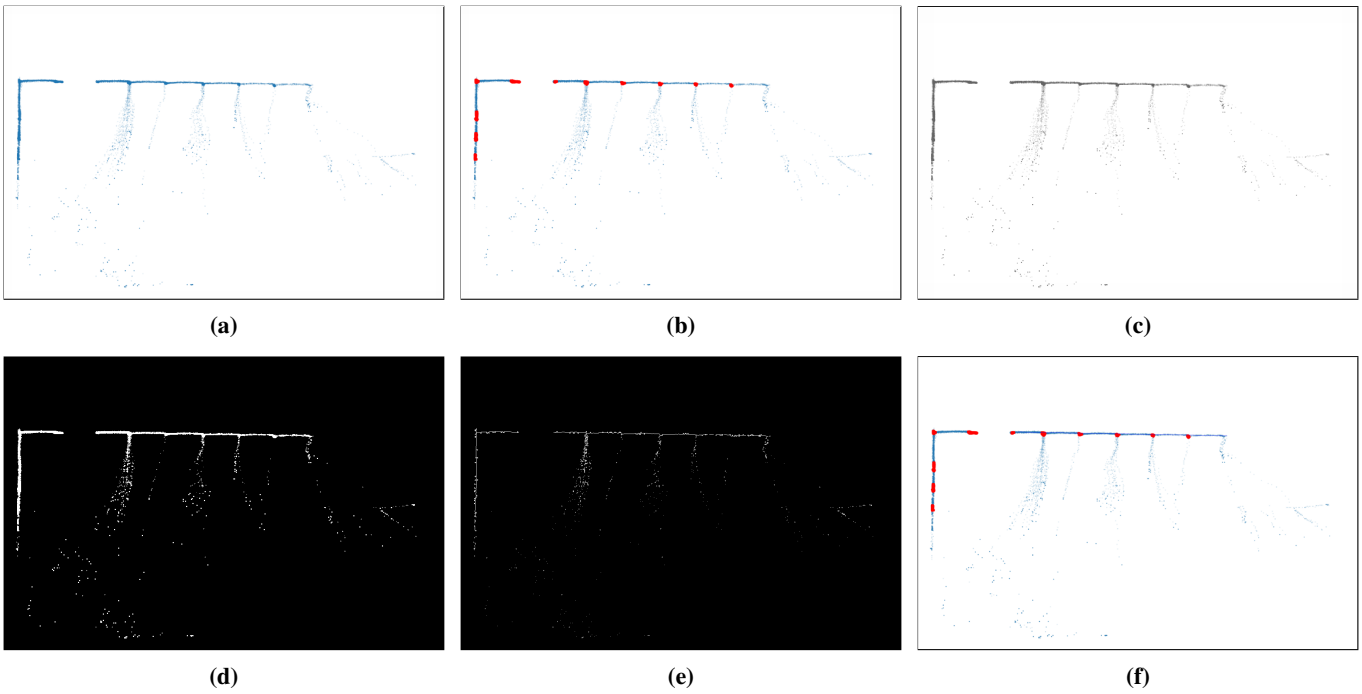


Figure 4: (a) Detection centroids plotted on the XY plane, (b) post application of DBSCAN to the centroid data, with dense locations plotted in red, (c) result after applying grayscale to the plot image, (d, e) result after applying thresholding (add params) and thinning (add params) operations to the image, (f) the final extracted line matching the cluster centers.

Figure 5: (a) Left: After step 7 of the calibration algorithm. The locations (red points) of the weighting springs along the conveyor belt are acquired. (b) Middle: During step 8 of the calibration algorithm where each point is enclosed into a polygon. (c) Right: Towards the end of step 8, where the polygons of weight springs that are responsible for the same egg size are merged and the zones are finally formed.

for mobile and embedded platforms.

The methodology was tested over 3 video recordings of the single-lane egg sorting machine in operation, each capturing a counting session of a mixture of 30 to 110 eggs of 4 different egg grades: small (S), medium (M), large (L), and extra large (XL).

B. Evaluation Results

For each test case, the system was evaluated for its counting accuracy. Figure 6 presents the results from our experiments. Each row in the figure corresponds to a different video feed, while each plot in a column corresponds to the counts produced by a different device. The blue bar corresponds to the ground truth (i.e., a count obtained by manually counting the eggs), while the orange and green bars correspond to the counts achieved using the EfficientDet-Lite0, and YOLO algorithms respectively.

The results justify our initial hypotheses: the Desktop Personal Computer (PC) managed to count flawlessly in all scenarios. In particular, on the workstation, arguably the most powerful device in terms of computation capability among all others, the system manages to produce perfectly accurate counts for all videos with the EfficientDet-Lite0 model. On the same device, inference with the YOLO model manages to achieve similar results, only missing the count of a couple of extra large (XL) eggs in Video 1.

The Raspberry Pi 4 counting accuracy suffered from its low compute power, producing the worst results across all devices. The slow processing and thus long inference times, prevented RPi 4 to catch up with the speed of the counting machine and resulted in under-counts in almost all scenarios. In particular, EfficientDet-Lite0 performed inference in ≈ 0.159 seconds per frame on RPi 4, while YOLO inference took ≈ 0.515 seconds per frame. Worth noting is the fact that our methodology seems to have a positive impact on the egg-counting, since in almost all scenarios the device appears to under- and not over-count the eggs.

Last but not least, the Raspberry Pi 5 appears to draw the processing boundary at least for the performance of EfficientDet-Lite0. Using the EfficientDet-Lite0 model, we managed to achieve perfect counting accuracy for all egg sizes across all three test recordings (see table I) with an inference time of ≈ 0.046 seconds per frame. On the other hand, Raspberry Pi 5 with YOLO produces sub-optimal results by missing the count of a considerable amount of eggs, especially those in large (L) and medium (M) groups. With YOLO on Raspberry Pi 5, inference time per frame reached

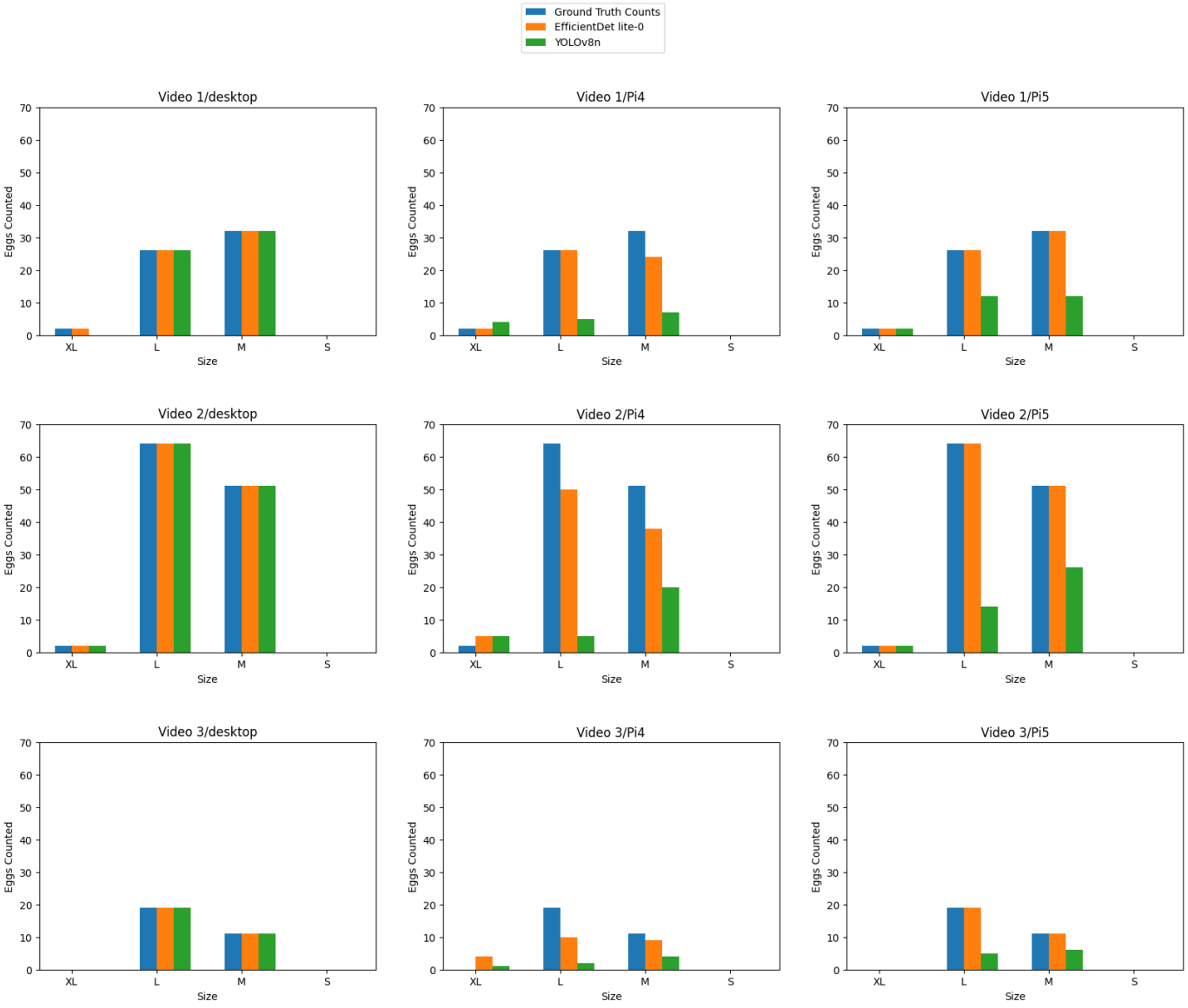


Figure 6: Produced results for all three devices and models over three different test videos. Each row represents processing of a different video. The first column displays the results achieved on the Desktop PC, the second column the counting performance of the Raspberry Pi 4 and the last column that of Raspberry Pi 5.

$\approx$ 0.256 seconds, 5.6 times greater than the time required by EfficientDet-Lite0 for the processing of an individual frame. In light of the preceding observations and analysis, it is apparent that egg-counting accuracy of our system is influenced by variations in inference frames per second. Additionally, the adaptability of the system to diverse egg grading machines poses a challenge, as modifications to certain components would be required for compatibility across other machine layouts. Future work could focus on transforming the system into a more generalized solution capable of operating on a wider range of equipment configurations. Moreover, extending the evaluation to environments with diverse conditions, such as varied lighting, would provide valuable insights into the system's robustness. Expanding testing to a larger, more comprehensive dataset would also help assess the system's accuracy on a broader scale, thus enhancing its reliability in practical applications.

TABLE I: ACCURACY OF THE RASPBERRY PI 5 BASED SYSTEM UTILIZING THE EFFICIENTDET-LITE0 MODEL.

Pi5 / EfficientDet-lite0			
	Counted Eggs	Ground truth	Accuracy(%)
Video 1	60	60	**100**
Video 2	117	117	**100**
Video 3	30	30	**100**

VI. Conclusion

In this work, we have demonstrated the usage of low-cost, resource-strapped, edge device capable of detection, tracking and counting of eggs. In an attempt to apply the idea of smart retrofitting, we have enhanced existing egg sorting equipment with a small footprint device, a Raspberry Pi 5. Following our methodology, we managed to count a set of graded eggs using visual inspection, in *real time*, and with *very high accuracy*. In fact, in the experiments we conducted, the accuracy of our counting was identical of that of a GPU equipped Desktop PC, and matched the ground truth in all cases. This showcases the advantage of our system compared to the rest in the relevant literature, which is the full capability of our system to operate on edge devices without the availability of computing intensive hardware like GPUs, something that other works in the area relied on.

Acknowledgment

This research was funded by the European Union Recovery and Resilience Facility of the NextGenerationEU instrument, through the Research and Innovation Foundation (CODEVELOP-ICT-HEALTH/0322/0061) of the Republic of Cyprus.

References

[1] G. Celenta and M. C. De Simone, "Retrofitting techniques for agricultural machines," in *New Technologies, Development and Application III*, I. Karabegović, Ed. Cham: Springer International Publishing, 2020, pp. 388–396.

[2] D. Jaspert, M. Ebel, A. Eckhardt, and J. Poeppelbuss, "Smart retrofitting in manufacturing: A systematic review," *Journal of Cleaner Production*, vol. 312, pp. 127–555, 2021.

[3] I. Kutyauripo, M. Rushambwa, and L. Chiwazi, "Artificial intelligence applications in the agrifood sectors," *Journal of Agriculture and Food Research*, vol. 11, p. 100502, 2023.

[4] I. Kumar, J. Rawat, N. Mohd, and S. Husain, "Opportunities of artificial intelligence and machine learning in the food industry," *Journal of Food Quality*, vol. 2021, p. 1–10, Jul. 2021.

[5] V.G. Dhanya et al., "Deep learning based computer vision approaches for smart agricultural applications," *Artificial Intelligence in Agriculture*, vol. 6, pp. 211–229, 2022.

[6] H. Tian, T. Wang, Y. Liu, X. Qiao, and Y. Li, "Computer vision technology in agricultural automation —a review," *Information Processing in Agriculture*, vol. 7, no. 1, pp. 1–19, 2020.

[7] M. Omid, M. Soltani, M. H. Dehrouyeh, S. S. Mohtasebi, and H. Ahmadi, "An expert egg grading system based on machine vision and artificial intelligence techniques," *Journal of Food Engineering*, vol. 118, no. 1, pp. 70–77, 2013.

[8] X. Yang, R. B. Bist, S. Subedi, and L. Chai, "A computer vision-based automatic system for egg grading and defect detection," *Animals*, vol. 13, no. 14, 2023.

[9] M. Turkoglu, "Defective egg detection based on deep features and bidirectional long-short-term-memory," *Computers and Electronics in Agriculture*, vol. 185, p. 106152, 2021.

[10] E. Nematinia and S. Mehdizadeh, "Assessment of egg freshness by prediction of haugh unit and albumen ph using an artificial neural network," *Journal of Food Measurement and Characterization*, vol. 12, 09 2018.

[11] A. Vinod, D. Mohanty, A. John, and B. Depuru, "Application of artificial intelligence in poultry farming - advancing efficiency in poultry farming by automating the egg counting using computer vision system," 08 2023.

[12] M. Ulaszewski, R. Janowski, and A. Janowski, "Application of computer vision to egg detection on a production line in real time." *ELCVIA Electronic Letters on Computer Vision and Image Analysis*, vol. 20, no. 2, p. 113–143, 2 2022.

[13] S. Subedi, R. Bist, X. Yang, and L. Chai, "Tracking floor eggs with machine vision in cage-free hen houses," *Poultry Science*, vol. 102, no. 6, 2023.

[14] I. Kanjanasurat, W. Krungseanmuang, V. Chaowalittawin, and B. Purahong, "Egg-counting system using image processing and a website for monitoring," in *2021 7th International Conference on Engineering, Applied Sciences and Technology (ICEAST)*, 4 2021, pp. 101–104.

[15] C. Lyu et al., "Rtmdet: An empirical study of designing real-time object detectors," 2022.

[16] Y. Luo, Y. Huang, Q. Wang, K. Yuan, Z. Zhao, and Y. Li, "An improved yolov5 model: Application to leaky eggs detection," *LWT*, vol. 187, p. 115313, 2023.

[17] T.-C. Shen and E. T.-H. Chu, "Edge-computing-based people-counting system for elevators using mobilenet–single-stage object detection," *Future Internet*, vol. 15, no. 10, 2023.

[18] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 3464–3468.

[19] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in *2017 IEEE International Conference on Image Processing (ICIP)*, 2017, pp. 3645–3649.

[20] D.-L. Dinh, H.-N. Nguyen, H.-T. Thai, and K.-H. Le, "Towards ai-based traffic counting system with edge computing," *Journal of Advanced Transportation*, vol. 2021, p. 1–15, 6 2021.

[21] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[22] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, 6 2020, pp. 10 778–10 787.

[23] W. Liu et al., *SSD: Single Shot MultiBox Detector*. Springer International Publishing, 2016, p. 21–37.

[24] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. [Online]. Available: http://arxiv.org/abs/1506.02640

[25] V. R. Group, "The gflops/w of the various machines in the vmw research group." [Online]. Available: https://web.eece.maine.edu/~vweaver/group/green_machines.html

[26] T.-Y. Lin et al., "Microsoft coco: Common objects in context," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 740–755.

[27] AutoML, "Automl implementation of efficientdet," https://github.com/google/automl/tree/master/efficientdet, accessed: 2024-09-10.

[28] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, p. 226–231.

[29] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited: Why and how you should (still) use dbscan," *ACM Trans. Database Syst.*, vol. 42, no. 3, 7 2017.

[30] "ncnn high-performance neural network inference computing framework," https://github.com/Tencent/ncnn, accessed: 2024-09-10.