Summary

Drosophila melanogaster, also commonly known as the fruit fly, has for decades been used in scientific research due to its inexpensiveness and substantial equivalency with the human genome. Large-scale experiments can quickly be materialized, and conclusions can be drawn based on the years of research that has made *Drosophila* one of the most comprehensively studied organisms. However, one significant issue with relying on *Drosophila* laboratory experiments is the often labor-intensive process of conducting the experiment. Many experiments require the manual counting of a large number of containers, each containing as many as 100 *Drosophila* eggs, larvae, or eclosed eggs.

As with many manual work processes, software can be a considerable help in automating the process. The use of software for *Drosophila* research is an already explored field, with stand-alone software systems and modifications to existing systems being employed. However, to our knowledge, utilizing the state-of-the-art within deep learning is hitherto unexplored. Therefore, we propose using the modern deep learning model, YOLOv5m, for a software system that gives biology researchers the ability to easily and automatically perform object detection and counting on their *Drosophila* image data.

Previous knowledge of the deep learning domain should not be required. For that reason, a web application is proposed where the model details are obfuscated from the user through the conventional front-end/back-end architecture. Since the user's contact with the object detection process is controlled through a front-end user interface, unnecessarily complex model details can be hidden. Assuming the web application is hosted, this also assures that a tedious setup process is avoided for the user.

Using data from the *Department of Chemistry and Bioscience at Aalborg University*, we train and test two models used in the object detection process. The central YOLOv5m model and a secondary neural network used to handle the domain-specific problem of object clumping, commonly seen in *Drosophila* data. The two models are combined in the complete object detection process, which can be employed from the user interface. Beyond presenting the object detection process, we also present the architecture of the proposed software system. The central segments of the user interface are shown, and the workflow for a user using the system to detect and count objects in uploaded data and subsequently view the results, is highlighted. Since the system should function as a fully-featured production system, we also present features explicitly created to fulfill this requirement. User management and granular resource management, both in terms of processing and storage resources, are included.

Since the selected YOLOv5m model is central to the complete software system, we conduct a comparative study of multiple state-of-the-art deep learning models. In this study, we compare five models, namely YOLOv5m, YOLOv5x6, Faster R-CNN, RetinaNet, and EfficientDet, in their ability to handle issues specific to *Drosophila* data. Object detection metrics, regression metrics, and the models' proficiency at handling diverse data are all considered to find the optimal model for the system. Ultimately, both YOLOv5 variants achieved the best results, YOLOv5x6 performing slightly better with a significantly slower inference time. Since the objective is to produce a production-ready software system, the faster but slightly worse performing YOLOv5m model is chosen. The selected YOLOv5m model achieved a mAP@0.50 of 0.724 and mAP@[0.50:0.95] of 0.543 under a confidence threshold of 0.25 and an NMS IOU threshold of 0.45 with an inference time of 0.3 seconds per image on the test set. When ignoring bounding boxes and classes and only accounting for the object count, the model achieved a MAE of 5.305.

To conclude, we have provided a deep learning-based web application that accounts for domain-specific problems and provides the user with the required functionality as well as supplementary features to function in a production environment.

Musca: Deep Learning-Based Web Application for Counting Laboratory Populations of Drosophila melanogaster

Christian S. Godiksen
Aalborg University
cgodik17@student.aau.dk

Kristoffer N. Knudsen
Aalborg University
kknuds19@student.aau.dk

Abstract

The fruit fly *Drosophila melanogaster* has seen extensive use in scientific research due to its application as a versatile model organism. However, conducting large-scale experiments with *Drosophila* is time-consuming, potentially requiring the manual counting of *Drosophila* eggs and larvae in thousands of Petri dishes. In recent years, machine learning, particularly deep learning, has been a groundbreaking technology for alleviating manual labor. We explore the state-of-the-art within deep learning for object detection and conduct a comparative study to select the optimal model for this domain. Leveraging the state-of-the-art deep learning model for object detection, YOLOv5m, we devise an innovative web application[1] that will give researchers the ability to automatically detect and classify objects in *Drosophila* image data. On our 93 image test set, we achieve a mAP@0.50 of 0.724 and mAP@[0.50:0.95] of 0.543 under a confidence threshold of 0.25 and an NMS IOU threshold of 0.45 with an inference time of 0.3 seconds per image. Further, when treating the detector as class-agnostic while ignoring the predicted bounding box and only counting the detections, we achieve a MAE of 5.305. The proposed software solution provides an opportunity to utilize the cutting-edge within deep learning without requiring prior knowledge of the field and tedious setup from the user. Specifically, the software fulfills the base requirements for supporting *Drosophila* research and provides extended functionality to increase usability. Domain-specific issues, such as diverse data setups and clumping of objects, are handled explicitly. Further, the system provides granular control of users and resources to function optimally as a production software system.

1. Introduction

Due to the low cost, rapid generation time, and innovative genetic tools such as balancer chromosomes [1], *Drosohpila melanogaster* have become essential for biology research. Continuous research over the last century has led to *Drosophila* being one of the most well-understood organisms on the phenotype level. Furthermore, the fruit fly has a simple genome which, even with its simplicity, is 60% homologous to that of humans [2]. The significant connection to the human genome, coupled with the fact that about 65% of genes responsible for human diseases have homologs in flies [3], has made research involving *Drosophila* a cornerstone of substantial areas of biology, such as genetic research. Underlining the immense role of the fruit fly are the six Nobel prizes in physiology or medicine that have been awarded to work based on fruit fly research. The first was given to Thomas Hunt Morgan in 1933 for his work in proving the chromosomal theory of inheritance [4] and the most recent was given to Jeffrey C. Hall, Michael Rosbash, and Michael W. Young in 2017 for their discoveries of molecular mechanisms controlling the circadian rhythm [5].

Drosohpila melanogaster research that relies on large-scale experiments, particularly research involving systematic counting of certain *Drosophila* life stages such as eggs, is heavily influenced by the labor-intensive process of manual counting. Software systems, and especially in recent years deep learning, have played an essential role in alleviating manual labor. However, such automated systems are, in certain circumstances, similarly laborious due to the requirement of a tedious setup process. Furthermore, concerning deep learning, many state-of-the-art methods are negatively impacted by the need for significant knowledge within the field to utilize them. Cutting-edge deep learning research is centered on optimizing performance and efficiency, thereby deprioritizing ease of use. Nevertheless, difficulties specific to *Drosophila* object counting necessitate the use of the latest technology. The minuscule size, clumping of objects, and cracked food surfaces obscuring eggs are among the significant issues hindering the use of automated, easy-to-use software systems.

In this work, we propose a software solution that incorporates the state-of-the-art deep learning model for object detection, YOLOv5m, to aid in object counting for *Drosophila* research. The software solution is usable by researchers in biology and other related fields without requiring any prior knowledge of the underlying deep learning model. In addition, minimal setup is required to use the system, which eliminates laborious elements to a large extent from the process of systematic object counting in *Drosophila melanogaster* image data. The solution also accounts for various domain-specific issues. This includes specific support for handling the clumping of objects and support for a multitude of different data setups. The latter is caused by the large variety of research conducted using *Drosophila* data, resulting in differences in the used containers, the storage conditions of these containers, and the color of the growth medium.

To ensure that the deep learning model that provides the foundation for this software solution is optimal, we conduct a comparative study of multiple state-of-the-art deep learning models within object detection and counting. The models in this study are evaluated based both on conventional performance metrics within the field of ma-

[1] `https://musca.bio.aau.dk/`

chine learning and also on their ability to handle domain-specific issues. Specifically, the models' capability to handle multiple data setups and their generalizability to new and unseen data setups are considered. As the conclusion to this study, we select the model used in the final software solution.

The remainder of this paper is organized as follows. Section 2 examines existing methods for counting *Drosophila*, while section 3 explores state-of-the-art deep learning methods for object detection within the biomedical field. In section 4, we present the system of methods used for the proposed web application, including the data, the deep learning model, and the complete object detection process. The proposed software is then described in detail in section 5, with a focus on presenting the architecture, user interface, and significant features. To substantiate the selection of YOLOv5m as the central deep learning model, we conduct a comparative study in section 6. Finally, we conclude the work and discuss potential future work in section 7.

2. Existing methods for counting *Drosophila*

Developing methods for reducing the workload of manual counting is not a new endeavor. As early as 1959, Geoffrey Keighley and E.B. Lewis from the California Institute of Technology devised a method to count *Drosophila* populations using a hydraulic system [6]. In recent years efforts have continued, focusing on using software tools and fully-fledged software systems. In 2015, Waithe et al. developed the software QuantiFly, for counting eggs with a density estimation approach [7]. To our knowledge, QuantiFly is the earliest example of using trainable machine learning methods to solve this specific problem. More fundamental image processing techniques were employed by Nouhaud et al. in a Java plug-in for the open-source image processing software ImageJ[2] [8]. Specifically, they employed the built-in *Analyse Particles* functionality to count *Drosophila* eggs.

Quantifying fruit fly adults is also an area that has seen work in recent years. Using statistical analysis of the body size distribution and a workflow that requires initially anesthetizing the insects, Yati et al. developed a MATLAB software tool for counting live *Drosophila* [9]. Similarly, Karpova et al. developed a tool for counting adult fruit flies using a modified mobile application, initially intended for counting wheat grains [10].

3. Deep learning for object detection

The software methods mentioned in section 2 have varying degrees of convenience in terms of ease of use. Existing methods require setting up stand-alone desktop software, employing extensions to existing software tools, or adapting counting applications developed for other purposes. Furthermore, existing tools only employ conventional machine learning, statistical analysis, or image processing means to accomplish the task of object counting. To the best of the authors' knowledge, deep learning has not yet been employed to solve the problem of counting different *Drosophila* life stages, primarily eggs, in a laboratory experiment setting.

Deep learning has emerged as a groundbreaking technology for object detection and classification in images. It has especially seen success within the biomedical field with diverse use cases. For example, U-Net, a deep learning model designed for cell counting, detection, and morphometry [11], has seen extensive use in the field. U-Net is particularly useful for segmentation tasks, for example, being applied by Dong et al. for brain tumor detection and segmentation [12].

For non-segmentation tasks, the Region-Based Convolutional Neural Network (R-CNN) family of models has seen success. R-CNN, and more advanced versions such as Fast R-CNN and Faster R-CNN, accomplish object detection by using bounding boxes instead of complete image segmentation [13]. Within the field of biomedical image processing, R-CNNs have been employed for a wide variety of applications such as cancer cell detection [14] and DNA damage analysis [15]. Another group of deep learning models used in the field is the You Only Look Once (YOLO) family of models [16]. Five main versions of the YOLO model currently exist, each adding performance and efficiency improvements to the previous. The only outlier is the fifth model version, where the main contribution is a more accessible implementation while maintaining performance and efficiency.

The main difference between R-CNN and YOLO is the detection process. R-CNN employs a two-stage detection process, where the first stage is used to extract regions of objects and the second is used for classification and refining the extracted regions. In contrast, YOLO handles this entire process in a single stage, with the benefit of speed and simplicity but usually at the cost of accuracy. Other one-stage detectors such as RetinaNet have emerged to attempt to solve this problem of decreased accuracy while still maintaining speed [17]. This model has also seen success within the biomedical field, for example, being successfully applied for the detection of pneumonia in chest x-ray images [18] and lesion detection in Computed Tomography (CT) scan images [19].

Achieving state-of-the-art accuracy is not only a complex issue in architecture design but also costly in terms of resources. To counteract this, certain families of models are explicitly designed with efficiency in mind. One such family is EfficientDet, which focuses on scalability and general efficiency over a wide range of resource constraints [20]. The objective of EfficientDet is to minimize the needed number of model parameters and floating-point operations used for annotating a single image while still maintaining performance. This area of focus means the model is less ideal in static and resource-rich environments but makes it favorable for environments where resources are scarce, either due to computational costs or hardware limitations.

A complete object detection model is generally structured using a multi-stage design, where the Convolutional neural network backbone, used for feature extraction, is the initial stage. Usually, the backbone is pre-trained on an existing public image database, such as ImageNet[3], which results in a model that is generalizable to new and unseen data. The backbone feeds into the head of the object detector, supplying features to the stage responsible for predicting classes and bounding boxes of objects.

[2] https://imagej.net/

[3] https://www.image-net.org/

Finally, modern object detectors employ an intermediary stage between the backbone and head, called the neck. This stage is usually comprised of multiple bottom-up and top-down paths, used to collect feature maps throughout the layers to shorten the information path between lower layers and topmost features [16].

4. Methodology

In this section, we present the system of methods used in the final proposed software. First, we introduce the data supplied by the *Department of Chemistry and Bioscience at Aalborg University*. The chosen object detection model, YOLOv5, is thoroughly introduced, including details about the model's backbone, neck, and head. Following that, the solution for the domain-specific problem of object clumping is established. Finally, we present the complete object detection process, focusing on the internal processing required to obtain the final result.

4.1. Data

In collaboration with the *Department of Chemistry and Bioscience at Aalborg University*, we have obtained 541 images of Petri dishes containing *Drosophila* eggs, eclosed eggs, and larvae. About 10% of the Petri dishes contain no objects, while the most populated Petri dish contains 101 objects. The images are taken through a stereomicroscope using two different phones at different image resolutions and orientations. One of the phones captures photos at a resolution of 4000x3000 or 3000x4000, depending on the orientation of the photo during the capturing, while the other phone captures at a resolution of 4032x1908.

The images are taken over an extended period of time and include various data setups, as seen in Appendix A in figure 9. These setups differ in the color of the growth medium for the base, as well as if the Petri dishes are fresh or if they have been frozen down before the image was taken; a standard procedure at the bioscience department, allowing for more time-controlled experiments. An exhaustive list of the data setups and their corresponding number of images is found in table 1. Further, each container has a varying amount of noise, such as oatmeal, dry yeast, cracks, bubbles, and light reflections. A concrete sample picture containing 39 class instances on the green-blue medium is seen in figure 1a. The object classes, seen in figure 1b, are following:

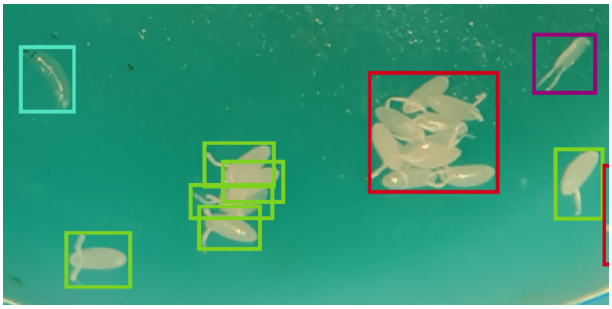
1. Egg. This covers *Drosophila* eggs where the larva has not emerged from the egg. This class constitutes the majority of the objects.

2. Eclosed egg. Empty *Drosophila* eggs where the larva has emerged resulting in an empty egg.

3. Larva. An emerged *Drosophila* larva.

4. Lump. A dense collection of *Drosophila* eggs, where even domain experts have a hard time counting the exact amount of eggs.

(a) An image can contain multiple classes and several objects within the same class. Note also the light reflections in the growth medium.



(b) Zoomed in rectangle from subfigure (a) showing the different classes. Green rectangles indicate *Drosophila* eggs; likewise purple indicate eclosed eggs, teal indicates larvae, while red indicates lumps.

Figure 1. These images contain *Drosophila* eggs, eclosed eggs, larvae, and lumps.

Do note that the 72 Petri dishes used for the fresh grey setups are the same Petri dishes used for the frozen grey setup; likewise, the 84 fresh black Petri dishes are the same as the frozen black Petri dishes. While the Petri dishes are the same, inevitably, there will be differences in the placement of the Petri dish, angle of the camera, light reflections, bubbles, and the frozen setups are generally more blurry. Further, the physical objects can change location between the fresh photo is taken and the Petri dish is frozen. As a result of these dissimilarities, the bounding boxes are not equivalent, and the images can be seen as distinct images; figure 10 in Appendix B exemplifies the differences between fresh and frozen Petri dishes.

The images are labeled manually using the online tool Supervise.ly[4] with assistance and guidance from the *Department of Chemistry and Bioscience at Aalborg University*.

Detecting small objects is notoriously known for being more difficult than larger objects, with e.g YOLOv4 having an average precision (AP) of 20.4% for small objects, 44.4% for medium objects, and 56.0% for large objects on the MS COCO dataset at resolution 416 [16]. The lower AP for small objects is a common pattern also appearing in other models such as SSD, RetinaNet, CornerNet [16].

[4] `https://supervise.ly/`

Table 1. The dataset consists of six different data setups.

Data setup	Image count	Description
Pink spoon	160	Pink spoon with black growth medium
Blue-green	71	Petri dish with blue-green growth medium
Fresh grey	72	Petri dish with grey growth medium
Frozen grey	70	Petri dish with grey growth medium that has been frozen
Fresh black	84	Petri dish with black growth medium
Frozen black	84	Petri dish with black growth medium that has been frozen
Total	**541**	

Consequently, we introduce a dataset similar to our aforementioned *Drosophila* dataset, with the only difference being a tiling preprocessing step, where all images are cut into multiple rows and columns, making the objects relatively larger compared to the image size. We will refer to these as *tiled images*, and refer to the original images as *full-sized images*. The tiling process will be explained in greater detail in section 4.4.

4.2. YOLOv5

The specific YOLO model presented in this section is YOLOv5, chosen due to its more accessible implementation and increased usability in a production-ready software solution. We present the architecture of YOLOv5, including which backbone, neck, and head the model employs to perform object detection and classification. Note that YOLOv5 and YOLOv4 mainly differ in implementation details, which means the more rigorously defined YOLOv4 model will be used to outline the architecture.

The backbone was selected based on experiments and the theoretical justification that more layers result in a higher receptive field to handle increased input size, and more parameters result in a greater capacity to detect objects of different sizes in a single image. Note that the receptive field is defined as the size of the region in the input that produces the feature, which, when increased, also helps in recognizing large objects [21]. Thus, Bochkovskiy et al. selected CSPDarknet53 as the backbone of YOLOv4 [16]. The same backbone is used in YOLOv5 [22], we assume the motivation is similar.

CSPDarknet53 is a C-based open-source neural network implementation of the Cross Stage Partial Network (CSPNet), which allows for more gradient flow through the network, solving problems with redundant gradient information causing inefficient optimization and costly inference. CSPNet works by partitioning feature maps into two parts. One part is sent through a dense block of multiple convolutions while the other is sent directly to the next transition layer, thereby increasing gradient flow [23].

To further increase the receptive field while still maintaining network operation speed, Spatial Pyramid Pooling (SPP) is used in the neck of the model. SPP allows for pooling features at a deeper layer in the network to generate fixed-length outputs that can be fed to fully connected layers, thereby eliminating the need for initial cropping and warping of the input [24]. Furthermore, a Path Aggregation Network (PANet) is used in the neck to achieve parameter aggregation from different backbone levels. PANet aims to boost information flow and employs adaptive feature pooling to propagate useful information from each feature level to the predictor [25].

The head of the YOLOv5 architecture is an anchor-based prediction head that has been similarly used in YOLOv3 and YOLOv4. Anchor-based heads use predetermined anchor boxes to represent the objects' ideal shape, size, and location. A large number of anchor boxes are added to the image, these are then filtered, categorized, and their coordinates refined before finally representing the object detection results [26]. To summarize, YOLOv5 uses a CSPDarknet53 backbone with Spatial Pyramid Pooling and a Path Aggregation Network as the neck to link the features to the anchor-based prediction head.

YOLOv5 provides multiple pretrained weights, trained on different architectures, that can be used to achieve promising results even with few images and few epochs. These pretrained weights, referred to as *checkpoints*, are trained on the COCO dataset[5]. These checkpoints are intermediate weights from which we continue the training to specialize the model to detect *Drosophila* life stages. YOLOv5 provides five different architectures with a varying number of layers: nano (YOLOv5n), small (YOLOv5s), medium (YOLOv5m), large (YOLOv5l), and extra large (YOLOv5x).

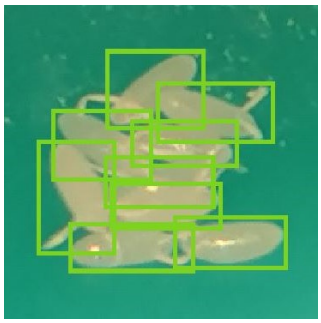
As each checkpoint is trained on a distinct architecture and each has their advantages and disadvantages, there is no definite best model. Choosing a larger architecture limits the batch size as well as the input image size, due to memory constraints. On the contrary, a smaller architecture might not be able to capture enough information resulting in lower performance. For each of the five mentioned checkpoints, there is also a supplementary checkpoint, trained on the same architecture, but using images of size 1280px rather than 640px. The checkpoints trained on 1280px size images are suffixed with a "6", e.g "YOLOv5n6" instead of "YOLOv5n" indicates that the checkpoint is trained on the nano architecture with images of size 1280px.

4.3. Estimating lumps

In this section, we present a method to solve the problem caused by *Drosophila* objects clumping together, resulting in the lump class presented in section 4.1. One technique that could solve this problem is annotating each object in a lump and relying on the model to provide the detection of each object in subsequent lumps. The problem with this method is that the objects are overlapping and so tightly concentrated in a small area that providing bounding-box annotations for each object might decrease the overall quality of the model. This is due to an object in a lump possibly not being representative of a general

[5] `https://cocodataset.org/`

object. To visualize the weakness of this approach, an attempt to annotate individual objects in a lump is presented in figure 2a, showing overlapping bounding boxes in even a relatively simple lump. Reinforcing the need for an alternative method is the presence of even more complex lumps, such as the one seen in figure 2b.



(a) Simple lump with overlapping annotations.

(b) Complex lump with overlapping objects.

Figure 2. Extracts from the data showing the complexity of the lump problem.

Since individual annotations are undesirable, we have employed an alternative method where lumps are first annotated as a single object. This results in the model being able to detect areas where further prediction is necessary. As the final objective of the object detection process is to count each individual *Drosophila* object, the detection of lumps serves only as an intermediary step.

After an image is annotated with the object detection CNN model, the detected eggs, eclosed eggs, and larvae are sent to the final result while the detected lumps are further examined. Lump object detections are used to extract the specific segment of the image containing the lump. Each lump image is then passed through a separate neural network model, explicitly trained to estimate the object count in a single lump.

Due to the high complexity of obtaining an exact object count of a lump, the output of the lump count model should only be seen as a rough approximation. To account for this, the estimated count of each lump n, is converted to an integer interval $[\lfloor n - (n * 0.2) \rfloor .. \lceil n + (n * 0.2) \rceil]$. An expert user can then leverage this range of numbers to decide the exact count of the lump efficiently. Finally, the user-specified count is added to the total object count of the image in question. We assume that all objects in a lump belong to the egg class. Note that the process of the user supplying the exact count is handled in the proposed software solution and is described in section 5.2.

The neural network model employed to estimate the object count of a single lump is trained using data extracted from the data described in 4.1. The annotated image data from the *Department of Chemistry and Bioscience at Aalborg University* was parsed to extract 486 images, each containing a single lump. These images were then individually labeled to provide the object count of each lump image. Using the images as input to a CNN model was explored but resulted in less than ideal performance, most likely due to the complex composition of the lumps. Instead, continuous features are extracted from the images and fed to a more conventional neural network.

Each feature was chosen based on an assumed correlation with the object count of the lump. The final feature set was selected based on an ablation study where features were added and removed systematically to gauge the individual contribution of each feature. The size of the lump image is used as a feature, as well as two features aiming to describe the lump specifically. To avoid issues with spread-out lumps resulting in large lump images that inflate the object count, color thresholding is used to approximate the percentage of the image that consists of the lump itself. Finally, to capture information about individual objects within the lump, Canny edge detection [27] is applied to the image, and the percentage of the image consisting of these edges is calculated. The final feature set is size, lump percentage, and edge percentage. Finally, the features are normalized to ensure that each feature is equally influential.

In terms of model architecture, the neural network is a conventional sequential deep network with six layers. This includes five hidden non-linear layers and one linear output layer. The first layer has 256 neurons, the second and third 128 neurons, and the fourth and fifth 64 neurons. Each layer uses the Rectified Linear Unit (ReLU) activation function. For training, we use mean absolute error for the loss function, and Adam [28] with a learning rate of 0.001 for the optimizer. All hyperparameters were selected using grid search, including the number of layers, number of neurons in each layer, activation function, optimizer, and learning rate.

The data was split into training, validation, and test sets using 64% for training, 16% for validation, and the final 20% for testing. Evaluating the trained model on the test set resulted in a mean squared error of 1.1630 and a mean absolute error of 0.7618. We consider this adequate for estimating the object count of lumps in future data.

4.4. Object detection process

In this section, we present the complete object detection process, from the user supplying a batch of images to the final result. The section aims to illuminate the internal architecture of the actual object detection process used in the proposed tool. The described process assumes the existence of a trained YOLOv5m model and a trained lump estimation neural network.

The full process for detecting objects in a single image is shown in figure 3. For a batch of images, the shown process is run on each image in the batch sequentially. When the user supplies an image to the object detection process, the image is first tiled to ensure the images supplied to the YOLOv5m model are of ideal size. The tiling method used during inference is identical to the method used during training to ensure the controllable aspect of the image data, namely size, is equivalent to the training data. As mentioned in section 4.1, smaller images result in the objects being larger relative to the image size, therefore, being easier to detect and classify. However, smaller tiles also require more processing to handle a single full-sized image. Based on experiments, 1280x960 was chosen as the ideal tile size, maintaining inference speed while optimizing detection results. It should be noted that to avoid zero-padding when border tiles are not large enough to fill the entire 1280 width or 960 height, the excess pixels are concatenated to the tile to the left with excess width and the tile above with excess height.

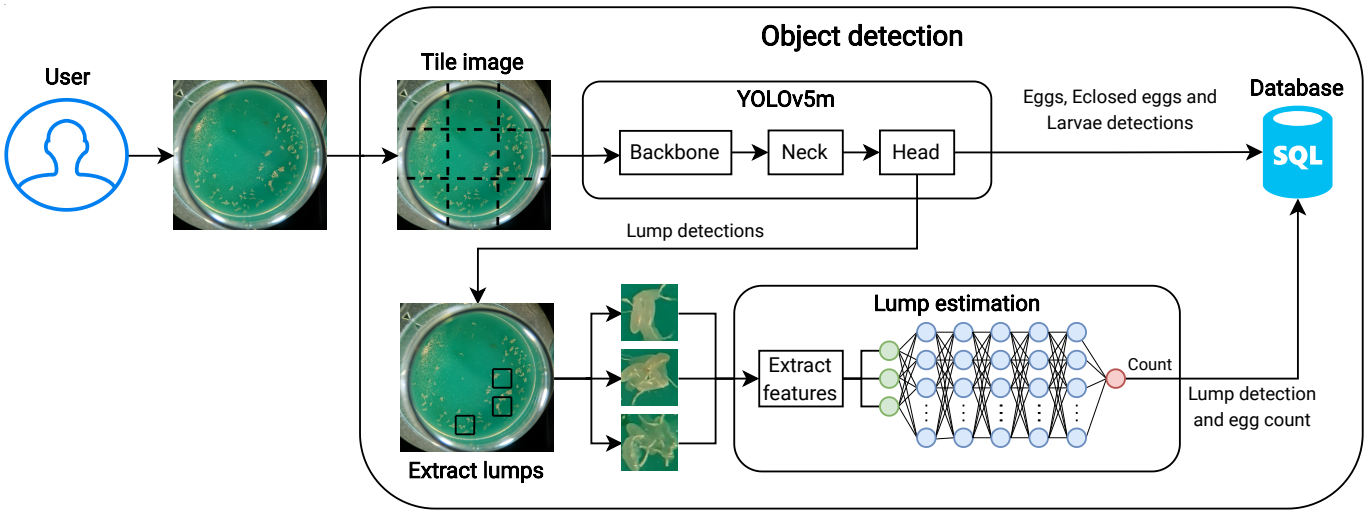


Figure 3. Full object detection process from the user supplying an image to the detections being saved in the database for later retrieval.

When the full-sized image has been fully divided into tiles, each separate tile is fed into the YOLOv5m model. As explained in detail in section 4.2, the model processes the image by passing it through the pre-trained backbone, which passes image features to the head by utilizing the model neck. In the model head, detections, containing both information about the bounding box and class, are generated. Eggs, eclosed egg, and larvae detections are sent directly to the database, where they are persisted for later retrieval. As mentioned in section 4.3, lump detections are further processed to estimate the egg count within each lump.

The bounding box information is used to extract each lump detection into a separate lump image. After extraction, each lump image is processed to retrieve the three aforementioned continuous features, namely size, lump percentage, and edge percentage. The feature data is then passed through the layers of the lump estimation neural network to retrieve an estimated count of the number of eggs within the specific lump. This count is then combined with the lump detection and saved in the database for later retrieval. It should be noted that this count is initially marked in the database as an estimate and will only be marked as the actual count when a user manually supplies the count. The process for supplying the actual lump egg count is described in detail in section 5.2.

5. Proposed software

We now present our proposed software solution to solve the problem of manually counting *Drosophila* eggs in scientific experiments. The purpose of the software is to provide a usable, fast, and flexible platform that researchers can utilize in their work without requiring prior knowledge of deep learning. In relation to that, the platform should not require a lengthy or tedious setup. Fulfilling this requirement would provide researchers within the bioscience field an opportunity to utilize the state-of-the-art within deep learning. We propose a web application solution[6] to obfuscate the implementation details of the above-described YOLOv5 model and remove the need for individual setup. As a minimum requirement, the web application should provide functionality for image uploads and subsequent viewing of the object detection results. The natural front-end/back-end architecture of web applications enables hiding the model in the back-end, and it requires no setup from the individual user, assuming the web server is already hosted.

In this section, we first describe the system architecture for the proposed web application and present the workflow for using the application to detect objects in *Drosophila* image data. Then, we present the supplementary features of the application, added to increase usability. Finally, we present features that support using the proposed software in a production environment.

5.1. System architecture

The complete system architecture can be seen in figure 4. The users of the application interact with the front-end, which is implemented in the statically typed programming language Typescript. The user interface is built using the open-source front-end framework React[7]. Communication between the front-end and back-end is handled using an Application Programming Interface (API), specifically using Django[8], which is a framework for developing web applications in Python. We adhere to RESTful conventions such as a uniform interface, separation between client and server, and statelessness. From the perspective of the front-end, requests are made to the API using the promise-based HTTP client Axios[9].

Due to the main objective of the server being interaction with the YOLOv5m PyTorch model, Python was chosen as the implementation language. This allows direct usage of the model in the server without further message brokering to external clients. To store user information, uploaded images, detections, and other persistent data, we use the small, fast, and self-contained SQL database engine, SQLite[10]. SQLite is especially powerful in a resource-sparse production environment because the entire database is contained within a single file. The internal Django object-relational mapper (ORM) is employed

[6] https://musca.bio.aau.dk/

[7] https://reactjs.org/

[8] https://www.djangoproject.com/

[9] https://axios-http.com/

[10] https://www.sqlite.org/

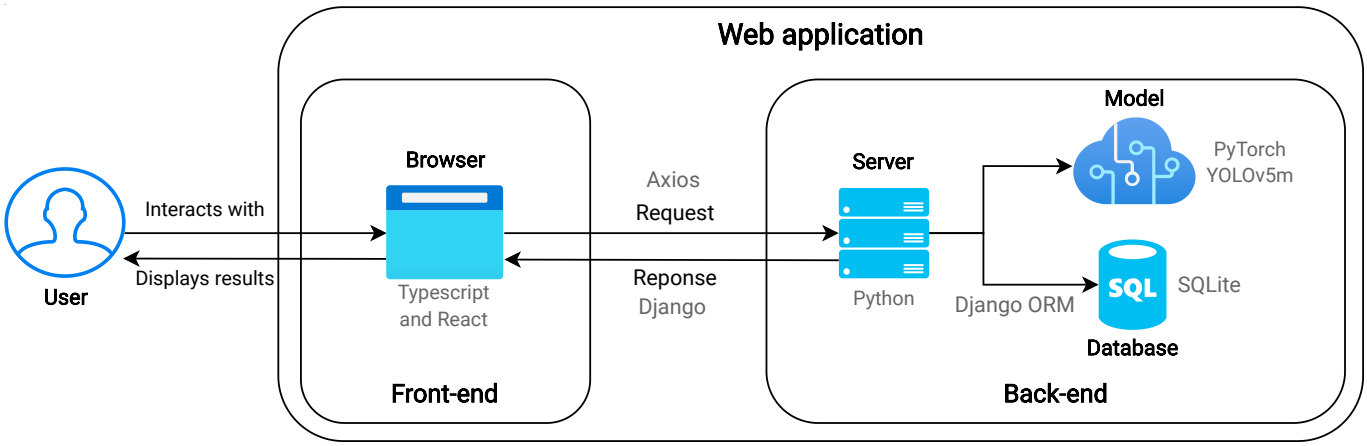


Figure 4. Full system architecture of the proposed web application.

to interact with the SQLite database from the Python server.

5.2. User interface

Run a job

The main requirement of the web application is to allow for object detection and classification in manually uploaded image data. The workflow for accomplishing this task using the proposed software solution is seen in figure 5. To initialize the task, the user interacts with the *Run* page. The user interface of this page can be seen in figure 11 in appendix C. The user can upload a collection of related images and create a new job with a name and description. For example, the user might want to group image data from the same experiment or data within the experiment that conforms to a certain set of criteria. When the job is submitted, the name, description, and image data are sent to the server. From here, the name and description are used to create a new *result* in the database, and the image data is sent to the YOLOv5m model. The images pass through the model's backbone-neck-head architecture and is subsequently passed through the lump estimation model. The resulting object detections are saved in the database. It should be noted that this is a heavily simplified view of the internal processing required to obtain detections from the given image data. The entire process is described in detail in section 4.4.

View the results

When the above-described process is completed, the user can now view the object detection and classification results using the *Results* page. The user interface of this page is shown in figure 12 in appendix C. The page includes a browsable list of all the user's results. When a result is selected, the user is presented with an interactable image canvas, an image list, an object list, and a list of lumps. Selecting a new image in the image list changes the presented image and object lists. From here, the user can add new objects, edit the class of existing objects, or delete objects.

As mentioned in section 4.3, the output of the neural network responsible for counting eggs in lumps can only be seen as a rough estimate. To account for this, *counted* and *uncounted* lumps are presented separately in the user interface with green and red, respectively. Images with

Table 2. Table representation of the per image class distribution CSV file, generated when exporting a result.

id	filename	egg	eclosed_egg	larva	total
89	run2_FreshBlack_27.jpg	31	1	0	32
90	run2_FreshBlack_19.jpg	45	0	0	45
93	run2_FreshBlack_24.jpg	24	1	1	26
96	run2_FreshBlack_31.jpg	1	0	0	1
97	run2_FreshBlack_20.jpg	90	2	0	92

Table 3. Table representation of the class distribution CSV file, generated when exporting a result.

class_name	image_count	object_count	min	max	average
Eclosed Egg	4	5	0	2	0.26
Egg	19	475	1	90	25
Larva	1	1	0	1	0.05
Total	19	481	1	92	25.32

uncounted lumps are also marked with red in the list of images. The user can go through the *uncounted* lumps in the list and provide an exact count, guided by the estimated range. Finally, to provide a more usable version of the result, the user can export and download the annotated images and corresponding result statistics. Note that the intended workflow is that the user counts all lumps manually before exporting the result statistics. This is reinforced by the user being warned if trying to export a result with *uncounted* lumps.

When exporting a result, two CSV files are downloaded. One file shows the class distributions per image, and the other shows total class distributions for the entire result. The first five rows of the image statistics CSV file, generated when exporting the result shown in figure 12, can be seen in table 2. The accompanying total class distribution CSV file can be seen in table 3.

Share the results

Sharing results only using the above-described features would require that the user exports the result and shares it manually through external means. A supplementary page is added to the web application to ease the process of sharing results with other users. The *Teams* page enables the creation of teams with other users of the system and allows users to link their results to a team to share them with all other team members. The user interface to support this is shown in figure 13 in appendix C. The feature was added to increase the usability for the intended user base, hereby

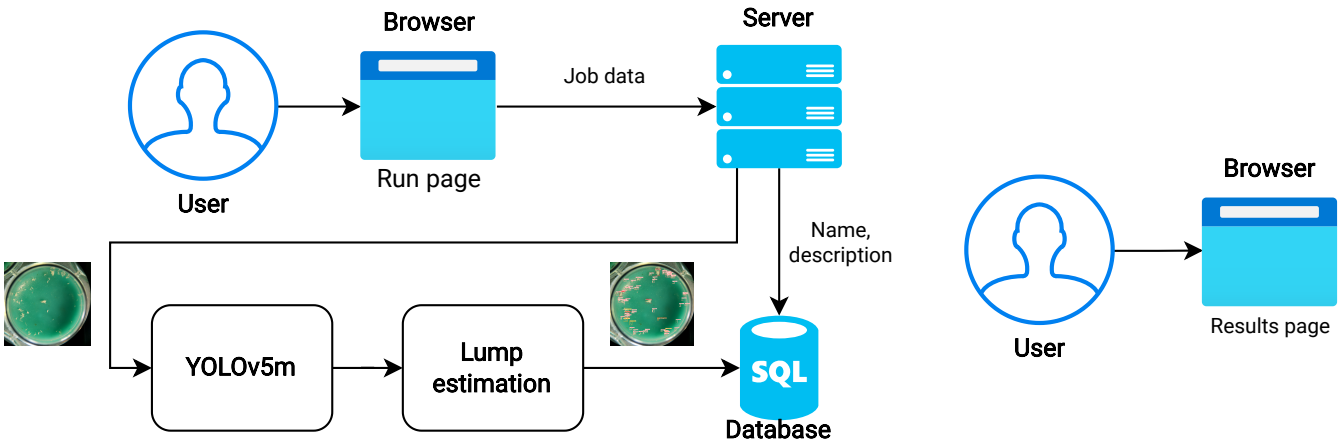


Figure 5. **How to use the web application to detect and count objects in uploaded data and view the results.**

meaning researchers and aims to support the cooperative nature of their work.

5.3. Production features

The above-described features provide a hypothetical exemplary user the ability to use the system to fulfill the main requirement, automatically counting *Drosophila* objects. However, when building a production-ready piece of software, especially software open to the web, non-exemplary, even in some cases malicious, users have to be considered. Furthermore, deploying a web application requires a host environment, most likely with limited resources, necessitating strict resource management in terms of processing power and storage.

User management

A foundational user system with login and register capability is included to support multiple users, each with their own results and teams. To further limit the expensive internal processing of the system from malicious users, user verification by an admin is required, effectively limiting the system to users approved explicitly by the hosting body. A simple ownership-based permission system is employed to limit specific user actions on a per-object level. Results and teams are visible to both owners and users with whom the objects have been shared with, but create, update, and delete actions are only available to the owner of the corresponding result or team object.

Processing resource management

For processing resource management, the admins are given complete control over limiting the resources on a per-user level. Multi-threading is utilized to avoid blocking requests to the server for the duration of model inference and related processing when running a job. The CPU and GPU heavy process is started in a separate thread, leaving the main thread available for processing subsequent requests. Using the same admin panel as for user verification, admins can control how many concurrent job threads a specific user can have simultaneously. If this maximum is exceeded, the creation of new threads is blocked.

Storage resource management

Following the same system design methodology, storage is similarly controlled on a per-user level, with complete control given to the admin users. Storage space is primarily used to store the images for the lifetime of a result, which, even with heavy compression applied, are sizeable compared to the objects stored in the relational database. Admins can therefore control the amount of storage space available to a specific user for image storage down to the megabyte level. If a user attempts to start a job that would exceed their maximum storage limit, the job is blocked, and the user is notified, allowing them to delete or archive previous results before attempting to run again. Archiving a result only removes the images from the server but keeps the data in the database required to view the class statistics of a result. Automatic *Garbage collection* of old results is also employed to continuously archive no longer needed results. Users are notified before results are achieved to allow them to postpone the *garbage collection* process manually.

6. Experiments

In this section, we will first introduce the performance metrics used to evaluate the models. Subsequently, we will introduce the deep learning models used in the comparative study in addition to the already introduced YOLOv5. After having introduced the models, we will examine the models' performance on the dataset using both object detection and regression performance metrics. Finally, we will conduct a failure analysis for each model on a concrete image to further inspect the inference results.

For YOLOv5, we use the medium-sized architecture, YOLOv5m, found to achieve the best results on a subset of our *Drosophila* dataset in a previous YOLOv5 comparative study [29]. Supplementary to YOLOv5m, we include YOLOv5x6 in the comparative study under the hypothesis that the larger architecture and the higher resolution of the images in the pretrained COCO dataset will lead to a more generalizable model that supports detection of objects in multiple, or previously unforeseen, data setups.

The models are trained in Aalborg University's High-Performance Computing environment CLAAUDIA with a NVIDIA Tesla V100 GPU. When conducting the experiments, the models will be evaluated on a regular consumer GPU; namely the NVIDIA GeForce RTX 3060 with 12GB VRAM.

(a) Ground truth bounding box.

(b) Predicted bounding box.

(c) Both bounding boxes.

Figure 6. IOU is calculated by dividing the intersection of the bounding boxes by the union of the bounding boxes.

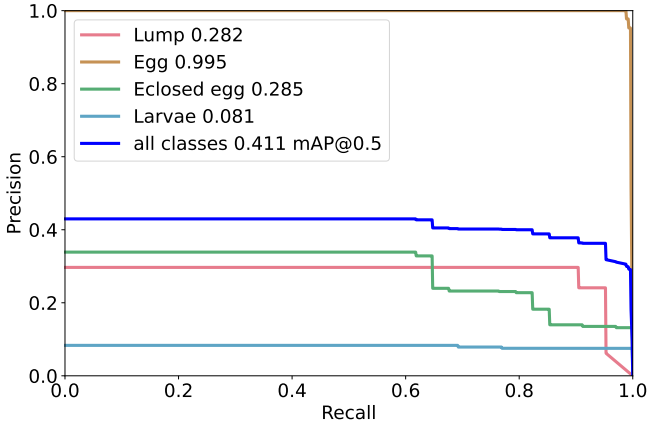


Figure 7. Precision-recall curve. The precision and recall values in this figure are not a part of the experimental results.

6.1. Performance metrics

We validate our models using the Average Precision (AP@k) performance metric. AP@k is a commonly used performance metric in image classification [30–32], and is a measurement of both Precision and Recall under specific Intersection Over Union (IOU) thresholds. Thus, AP@k measures the model's ability to detect the ground truths while minimizing the amount of false positive detections. Inspired by the well-acknowledged computer vision articles "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks" and "YOLOv4: Optimal Speed and Accuracy of Object Detection" [16, 33], we use both AP@0.50 and AP@[0.50:0.95] to evaluate the performance.

IOU corresponds to how well the predicted bounding box covers the ground truth bounding box [34]. These bounding boxes are illustrated in figure 6, while the definition is seen in Eq. (1). It is worth noting that since humans create the ground-truths in Supervise.ly, these ground-truth bounding boxes might be off by a few pixels between the objects; thus, a mean IOU of 100% is unachievable.

$$IOU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})} \tag{1}$$

Average Precision (AP) is defined as the area under the precision-recall curve (AUPRC). The precision-recall curve shows each class's precision at a given recall level. Figure 7 shows an example of such a curve. In this article, we follow the COCO convention of approximating the AUPRC using a 101-point interpolation [35]; these points being 0.00, 0.01, 0.02, .. , 1.00. We define AP_r as the precision at recall r. Let R be the set of interpolation points, such that $R = \{0.00, 0.01, .., 1.00\}$. We then define AP as seen in seen in Eq. (2).

$$AP = \frac{1}{101} \sum_{r \in R} AP_r \tag{2}$$

Finally, AP@k is the average precision, where the precision-recall curve results from the model using an IOU threshold of k, e.g., AP@0.50 uses an IOU threshold of 0.50. AP@[0.50:0.95] is the mean of AP@0.50, AP@0.55, .. AP@0.95, i.e., the average AP at IOU thresholds from 0.50 to 0.95 with a step of 0.05. Thus, AP@k can be used to examine the tightness of the predicted bounding boxes.

In continuation of AP, we define mean Average Precision (mAP) simply as the mean of multiple average precisions, i.e., the mean of each class' average precision.

Supplementary to the aforementioned image classification performance metrics, we also introduce a traditional regression performance metric as adopted by other existing object counting papers [36, 37]. We disregard the predicted bounding boxes for this metric and solely focus on the number of detected objects. Inspired by [36], we use Mean Absolute Error (MAE). More formally, let Y_i be the observed number of objects in image i, $\hat{Y}_i$ be the number of detected objects in image i, and n be the number of images. We then define MAE as seen in Eq. (3).

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |Y_i - \hat{Y}_i| \tag{3}$$

Lower MAE values are preferable, with the range of MAE being $[0, \infty)$. This performance metric and AP are used to decide the model in the comparative study.

Finally, it should be noted that the model can make a seemingly correct detection without getting rewarded with a correct detection, ultimately decreasing the AP and increasing the MAE. Such an occurrence is exemplified in figure 8, where the ground truth annotation is a lump, but the model instead detects two eggs. The detection is not technically wrong in such a situation, yet it does not equate to the ground truth. In this concrete example, the AP for both egg and lump decreases, while the MAE for egg, lump, and the total number of objects increases.

6.2. Training, validation, and test data sets

Prior to introducing the remaining models, we briefly discuss how the data is split into subsets for training, validation, and testing.

First, to enable examining the models' ability to adapt to new unprecedented setups, we exclude the blue-green Petri dish setup from the training, validation, and testing subsets. Instead, we use the 71 images from the setup as a holdout set. This setup is chosen as the growth medium visually distinguishes the most from the medium in the other setups.

Table 4. Five of the setups are split into training, validation and test sets.

	Training	Validation	Test	Median objects
Pink spoon	96	32	32	5.0
Fresh black	50	17	17	16.5
Frozen black	50	17	17	18.0
Fresh grey	46	13	13	8.0
Frozen grey	43	13	14	6.5
Total	**285**	**92**	**93**	

We employ a stratified split to enforce an even distribution of the setups across the subsets. Further, this stratified split also satisfies that images with a dense or sparse number of objects are equally distributed across the subsets. For each of the remaining five setups, we find the median number of objects per image, m. Subsequently, we partition the images in the setup into two subsets. The first subset contains the images with less than m objects, while the second subset contains the images with greater than or equal to m objects. Finally, we split these two subsets into training, validation, and test subsets, where we use 20% of the images for testing, 20% for validation, and the remaining 60% images for training. Finally, the respective subsets are concatenated, resulting in a single training, validation, and test set. This stratified split leaves us with the dataset as seen in table 4.

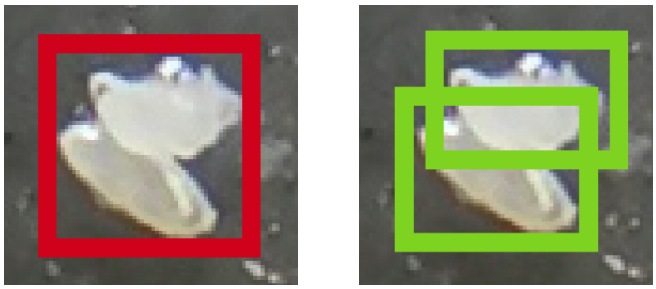
6.3. Comparative study

In this section, we introduce the models that will be used in the comparative study in more detail. Specifically, we present the architecture and distinguishing features of three different families of deep learning models: Faster R-CNN, RetinaNet, and EfficientDet. A detailed examination of YOLOv5, which is the object detection model used in the tool, can be found in section 4.2.

Faster R-CNN

Faster R-CNN [13] is an upgrade to Fast R-CNN with improved inference speed, obtained mainly by improving computations associated with region proposals. The model consists of two modules, where the first module proposes regions of interest, while the subsequent module is the Fast R-CNN detector, which conducts the object classification. While being two modules, both networks share convolution layers to reduce the cost of computing proposals. Specifically, we use *Faster R-CNN X101 32x8d FPN*, a modification of the original Faster R-CNN to use *ResNeXt-101-32x8d* [38] as the backbone, while also adding a Feature Pyramid Network neck to support varying image resolutions.

Data augmentation techniques used in Faster R-CNN include rescaling images while keeping the aspect ratio, rotating, random flip, and random crop. These data augmentations are, as well as the model itself, provided through Facebook AI Research's (FAIR) library *Detectron2* [39]. FAIR achieved a mAP@[0.50:0.95] of 0.430 with this model on the MS COCO dataset.



(a) Ground truth is a lump. **(b) Potentially the model detects the two eggs as individual eggs.**

Figure 8. It is possible for a model to make a seemingly correct detection, without the detection being a ground truth.

RetinaNet

RetinaNet is introduced in the 2017 article "Focal Loss for Dense Object Detection" [17]. Contrary to Faster R-CNN, RetinaNet is a one-stage detector and thus does not have a separate initial module to propose regions. By not having a region proposal network, RetinaNet faces the challenges of both determining which of the hundreds of thousands of anchors includes objects and are not just background, while also having to reposition the anchors to only cover the object. To overcome this, RetinaNet's head consists of two parallel sub convolution networks. The *classification subnet* predicts, for each anchor and each class, the probability of the object class being present within the anchor. Simultaneously, the *box regression subnet* regresses the anchor to fit the detected object.

Concretely, we use the RetinaNet model with *ResNet-101-FPN* for the backbone as FAIR found this achieved the highest AP on the COCO dataset. Data augmentation techniques are identical to the ones used in Faster R-CNN. On the MS COCO dataset, FAIR achieved a mAP@[0.50:0.95] score of 0.404 using this model. While this is slightly lower than that obtained using Faster R-CNN, RetinaNet used only 55% of Faster R-CNN's inference time.

EfficientDet

EfficientDet [20] is a one-stage detector employing *EfficientNet* as the backbone, while using their proposed *bi-directional feature pyramid network* (BiFPN) as the feature network to support input images of varying resolutions. BiFPN is a modification of PANet used by, e.g., YOLOv5. The head of EfficientDet consists of a class and box network. EfficientDet challenges existing detectors and aims to achieve both higher accuracy and more efficiency. As EfficientDet uses predefined anchors, we calculate the anchor ratios using the K-means algorithm on our training set, with IOU as the distance measure, as seen in other work [13, 40]. Inspired by their work, we solely use horizontal flipping and random image rescaling as data augmentation techniques.

EfficientDet provides multiple scaling configurations, denoted by ϕ and referred to as the compound coefficient. Due to memory constraints during training and later inference, we use $\phi = 0$, which uses the smallest EfficientNet backbone and has the lowest number of channels and layers. However, we modify the model to train on input size 1280, as seen with compound coefficients $\phi = 5$ and $\phi = 6$, rather than the default input size 512. The Google Brain

Team achieved 0.346 mAP@[0.50:0.95] using $\phi = 0$, while they achieved 0.515 using $\phi = 5$.

6.4. Comparative study results

Object detection results

Examining the results for the full-sized images, seen in table 5, it is evident that both YOLOv5 models outperform the remaining models on the test set, with YOLOv5x6 achieving the highest mAP@0.50 and mAP@[0.50:0.95] on both the test and holdout set, while YOLOv5m obtained the fastest inference speed. Concretely, YOLOv5x6 on the test set achieved a mAP@0.50 of 0.739 and a mAP@[0.50:0.95] of 0.561 in 40 seconds, while achieving respectively 0.341 and 0.206 on the holdout set. YOLOv5m, being the fastest, had an inference speed of 14 seconds, while achieving close to 90% of YOLOv5x6's performance metrics on the test set, but only nearly 50% of YOLOv5x6's performance metrics on the holdout set.

Faster R-CNN and RetinaNet yielded lower performance metrics than YOLOv5m on the test set. However, they outperformed YOLOv5m on the holdout set, with Faster R-CNN achieving mAP@0.50 and mAP@[0.50:0.95] of respectively 0.296 and 0.140 compared to YOLOv5m's performance metrics of 0.178 and 0.097. EfficientDet achieved significantly worse results than all the other models, with EfficientDet only being able to detect lumps, as the remaining objects were too small to be correctly detected by its anchors.

For the full-sized images, we conclude that YOLOv5x6 is the most suited model, with the remaining models being uncompetitive on either the test or holdout set. YOLOv5x6 achieved the best results on the holdout set, whereas YOLOv5m did not manage to keep up with the larger architecture on the not previously seen growth medium color.

The performance metrics for the models when using tiled images, contrary to full-sized images, are listed in 6. Also here YOLOv5x6 outperformed the remaining models w.r.t mAP except for mAP@[0:50:0.95] on the test set, where YOLOv5m achieved slightly better results. YOLOv5m especially improved on the holdout set, where its mAP@0.50 and mAP@[0.50:0.95] increased from 0.178 and 0.097 to 0.402 and 0.257. On the test set, YOLOv5m increased from 0.670 and 0.479 to 0.724 and 0.543.

Disregarding the inference time, all the models except for YOLOv5x6 benefitted positively from using tiled images on both the test and the holdout set. YOLOv5x6's performance metrics on the test set decreased slightly by using tiled images, possibly due to a reduction in contextual information near the objects. Despite the slight decrease on the test set, YOLOv5x6 did significantly improve on the holdout set, with the mAP@0.50 and mAP@[0.50:0.95] increasing from 0.341 and 0.206 to 0.447 and 0.284.

Particularly EfficientDet improved by using tiled images, making it outperform Faster R-CNN and RetinaNet on the test set, with EfficientDet increasing its performance metrics from 0.101 and 0.050 to 0.583 and 0.383. On the holdout set, EfficientDet outperformed RetinaNet while achieving results similar to Faster R-CNN.

In this experiment, we found that the YOLOv5 models achieved the highest mAP on both the test and the holdout set, with YOLOv5x6 obtaining slightly higher accuracy. Further, we also found YOLOv5m to be the model with the fastest inference speed among all the models.

When comparing the two tables, it is evident that object detection is slower on the tiled images compared to the full-sized images. For the YOLOv5 models, the inference time was slowed down by a factor of two, while for the remaining models, it slowed down by a factor of nine. It is also worth noting that all the models consistently achieve lower mAP scores for the holdout set compared to the test set, indicating that, to some extent, new and unseen data setups challenge all the models. As such, there are no guarantees that the models will be equally successful on non-industry standard setups or growth medium colors.

Regression results

While we in the previous experiment found that the YOLOv5 models with tiled images achieved the highest mAP, we will in this experiment continue exploring the most suited model from a regression perspective. Here, we examine the models' ability to count the various object classes. For this, we use the MAE regression metric previously mentioned in section 6.1. The MAE is determined for each object class per model. Further, for each model, we also determine the single-class MAE. The single-class MAE is calculated by treating the detector as class-agnostic, and thus solely comparing the number of detections to the number of ground truths while disregarding the predicted class.

For easier comparison, we include a *baseline non-detector*. This baseline model predicts each image to have zero objects. We will not discuss the results from this baseline method, as it is only included for comparison.

The results are shown in table 7. The classes will not be intercompared within the same model due to the high variance in class occurrences. E.g., many images do not contain eclosed eggs, so naturally, this is likely to have a low MAE compared to the egg class, which has a high occurrence rate.

Examining the results, we observe that YOLOv5x6 achieved the lowest MAE values, with the tiled images attaining the lowest MAE for eggs, lumps, and single-class, while the YOLOv5x6 with full-sized images obtained slightly lower MAE results for eclosed eggs and larvae. More concretely, YOLOv5x6 with tiled images achieves the following MAE results: 0.707 for eclosed eggs, 4.427 for eggs, 1.067 for larvae, and 0.671 for lumps and 4.591 for single-class. For the same classes, YOLOv5m with tiled images obtains respectively 0.707, 4.634, 1.055, 0.884, and 5.305. This does indicate that on tiled images YOLOv5x6 does outperform YOLOv5m with respect to MAE.

The highest MAE per class when disregarding the baseline detector is: RetinaNet for eclosed eggs with 0.896 MAE; 14.085 for eggs by EfficientDet when using full-sized images; 1.488 for larvae by EfficientDet on full-sized images; 1.366 using tiled-imaged with EfficientDet; and finally 16.573 for single-class again by EfficientDet, but with full-sized images. These large MAE results by especially EfficientDet will be further examined in the next section, where we explore the inference results.

Altogether, we found that YOLOv5x6 with tiled images yields the best MAE results, but at the higher inference time found in the previous experiment. With

Table 5. Performance metrics per model on full-sized images

	Validation set		Test set			Holdout set	
Model	mAP @0.50	mAP @[0.50:0.95]	mAP @0.50	mAP @[0.50:0.95]	Inference speed	mAP @0.50	mAP @[0.50:0.95]
YOLOv5m	0.685	0.479	0.670	0.479	**0m14s**	0.178	0.097
YOLOv5x6	0.726	0.527	**0.739**	**0.561**	0m40s	**0.341**	**0.206**
Faster R-CNN	0.413	0.216	0.412	0.219	0m34s	0.296	0.140
RetinaNet	0.269	0.116	0.311	0.147	0m16s	0.215	0.096
EfficientDet	0.049	0.098	0.101	0.050	0m52s	0	0

Table 6. Performance metrics per model on tiled images

	Validation set		Test set			Holdout set	
Model	mAP @0.50	mAP @[0.50:0.95]	mAP @0.50	mAP @[0.50:0.95]	Inference speed	mAP @0.50	mAP @[0.50:0.95]
YOLOv5m	0.756	0.567	0.724	**0.543**	**0m28s**	0.402	0.257
YOLOv5x6	0.750	0.557	**0.725**	0.537	1m32s	**0.447**	**0.284**
Faster R-CNN	0.532	0.334	0.487	0.297	4m38s	0.317	0.182
RetinaNet	0.463	0.282	0.428	0.268	2m19s	0.234	0.132
EfficientDet	0.392	0.599	0.583	0.383	6m40s	0.291	0.161

YOLOv5m obtaining comparable results on both eclosed eggs, eggs, and larvae, we can almost inconsequentially replace YOLOv5x6 with YOLOv5m for a three times faster inference speed. However, this replacement comes at the cost of a slight increase in MAE for lumps, with the MAE increasing from 0.671 to 0.884. The consequence of missing a lump will naturally depend on the size of the lump.

Failure analysis

To further inspect the inference results, we examine each of the models' detections on the image in the test set with the most objects. The main objective of this further examination is to make the detection differences between the models more perceptible. All the images that are referred to are found in Appendix D. The original image is seen in figure 14. In the following images, detection failures are marked with a blue rounded rectangle.

YOLOv5m: In the full-sized image (figure 15), two mostly-transparent eggs are missed in the top of the growth-medium. A larva that is partially covered by eggs in a lump is also missed. Some of the objects are location-wise correctly detected but are misclassified; this includes two eggs incorrectly classified as larvae, and some instances of two nearby eggs getting classified as one egg. When examining the tiled image (figure 16), we see that the two missed eggs in the top are now correctly detected, and moreover, the partially covered larva is also correctly detected. Further, the nearby eggs that would otherwise get detected as one egg is now detected as a lump of two eggs.

YOLOv5x6: In the full-sized image (figure 17), the model missed the same partially covered larva as YOLOv5m on the full-sized image. The model also detects two relatively close-laying eggs as one lump. It would have been more correct for the model to detect these as two independent eggs; however, the lump estimation component will handle this in the postprocessing phase. We also see some eggs that are counted both as an egg and as a part of a lump; NMS does not remove these egg detections due to NMS not being class-agnostic. It is worth noting that a class-agnostic NMS still would not solve this, as the IOU is low. In the tiled image (figure 18), we see that some blurry eggs are misclassified as eclosed eggs, as well as two eggs in continuation of each other, almost looking like a larva, actually getting detected as a larva. Another mistake, so far unique to this model, is some noise getting misdetected as an egg.

Faster R-CNN: Examining the full-sized image (figure 19), it is seen many of the bigger lumps are missed. While the model does detect some of the objects, the majority are missed. Besides missing lumps, various eggs are also missed. The only clear pattern in which eggs are missed is near the edge of the container, where the growth-medium is slightly lighter or darker than the middle. Some detected objects are misclassified as eclosed eggs rather than eggs. The network greatly improved from tiled images (figure 20), but it tends to detect nearby eggs as one single egg rather than multiple eggs or a lump. As we saw in some earlier models, mostly-transparent eggs and larvae are missed. This model also missed one of the larger lumps in the bottom of the container, as well as eggs where only the top of the egg is visible.

RetinaNet: This model also misses a lot of objects in the full-sized image (figure 21. Much like Faster R-CNN, RetinaNet also misses lumps near the edge of the container. Many of the missed eggs are also comparable to those missed by Faster R-CNN. RetinaNet does, however, also miss quite a few eggs in the middle of the container, which Faster R-CNN did correctly detect. While this model also benefitted from using tiles (figure 22), it still misses some lumps in both the bottom and the top of the container. The model misses some slightly misshaped and blurry eggs, and like Faster R-CNN some close-laying eggs are misdetected as being a single egg instead of a lump.

EfficientDet: Only lumps are detected in the full-sized image (figure 23), due to the other objects being too small for the anchor. Despite only detecting lumps, many smaller lumps are still false negatives. Overall, the model has a very low recall, missing a vast majority of the ground truths. On the other hand, when using tiled images (figure 24), the model can detect objects smaller than lumps and achieves inference results comparable to the YOLOv5 models. Still, some of the lumps close to the container's

Table 7. Mean Absolute Error for detected objects per class per model

Model	Eclosed eggs	Eggs	Larvae	Lumps	Single-class
YOLOv5m, full-sized	0.707	7.415	0.988	0.774	8.555
YOLOv5m, tiled	0.707	4.634	1.055	0.884	5.305
YOLOv5x6, full-sized	**0.659**	5.183	**0.970**	0.890	4.811
YOLOv5x6, tiled	0.707	**4.427**	1.067	**0.671**	**4.591**
Faster R-CNN, full-sized	0.665	7.348	1.048	0.854	8.220
Faster R-CNN, tiled	0.884	8.000	1.134	1.048	8.970
RetinaNet, full-sized	0.756	8.970	1.396	0.915	11.610
RetinaNet, tiled	0.896	9.152	1.140	0.939	10.494
EfficientDet, full-sized	0.829	14.085	1.488	0.707	16.573
EfficientDet, tiled	0.707	6.610	0.994	1.366	6.409
Baseline non-detector	0.774	14.140	1.488	1.384	17.786

border are missed, possibly due to the container having a slightly different growth-medium color in these areas. We also note that some of the eggs are misclassified as larvae, and the aforementioned partially covered larva is not detected as an object. Eggs, where only the top of the egg, and not the whole egg is visible, are most often missed. It is evident that especially EfficientDet greatly improved from using tiled images.

Based on the experiments in the study, we chose YOLOv5m with tiled images for the proposed software, as this model achieved results comparable to YOLOv5x6 while using less than a third of the inference time.

7. Conclusion and future work

In this work, we proposed a software solution to solve the problem of laborious manual counting of different *Drosophila* life stages in laboratory experiments. Through a study of the state-of-the-art within deep learning for object detection, four models, namely Faster R-CNN, RetinaNet, EfficientDet, and YOLOv5, were selected to be part of a comparative study. Examining the models' performance in terms of various object detection metrics, regression metrics, and a failure analysis, it was found that the two YOLOv5 models performed significantly better than the other models. While the larger YOLOv5x6 consistently achieved the best metrics, the faster YOLOv5m model was chosen due to its comparable results while being three times faster. Specifically, the chosen YOLOv5m model has a mAP@0.50 of 0.724 and a mAP@[0.50:0.95] of 0.543 on the test set, achieving these metrics with a per-image inference time of 0.3 seconds. Treating the chosen model as a class-agnostic detector yielded a MAE of 5.305.

One main issue with automating the process of manual counting is the large variety of data setups used to perform experiments. Such data setups can vary in the containers' material, shape, and storage conditions and in the color of the growth medium in the containers. To support this characteristic of *Drosophila* experiments, data from multiple different setups were used to train and test the models in the comparative study. Furthermore, a holdout setup was used to test the models' ability to process new and unseen data conditions. Here, the selected YOLOv5m model achieved a mAP@0.50 of 0.402 and a mAP@[0.50:0.95] of 0.257.

Another property of *Drosophila* data is the significant clumping of eggs. Since an important requirement of object detection is having a clear equivalency between objects of the same class, treating eggs within a lump in the same manner as individual eggs could significantly decrease the model's performance. To solve this problem, lumps were treated as separate single objects, allowing the model to detect areas of the image where further processing is required. Due to the high complexity of obtaining the actual egg count within a lump, it was decided that the user should provide the actual final count. To aid in this decision, we provide a lump estimation neural network, trained on a processed version of the original dataset, that can supply a range covering the options for the final actual count.

To eliminate as many tedious aspects of using the above-described YOLOv5m and lump estimation models, we proposed a web application tailored to hide the deep learning aspects of the system from the user. This system, designed to be used by a user without prior knowledge of the deep learning domain, provides the user with the functionality to upload images, process the images automatically, and view the results. Beyond fulfilling the main requirement of a common user, the system also accommodates users sharing their results with other users. Finally, to support the system running in a production environment, features aimed at providing the hosting body with intricate control over processing and storage resources are included.

7.1. Future work

Even though the proposed software solution is presented as a complete piece of production software, we acknowledge that further work could extend the functionality and usability of the solution. In this case, the software solution provides a strong foundation for future development, both in terms of optimizing the internal object detection process and improving the web application.

Optimizing object detection

One significant advantage of the architecture chosen for the tool is the simplicity of plugging a new model into the system. This allows the underlying model to be optimized or even changed without disturbing the system's users. Further optimization of the chosen YOLOv5m model or extending the comparative study to include more models is, therefore, ideal future work. Beyond further hyperparameter optimization, domain-specific optimization could also be considered. One characteristic of the data that could be exploited is that there is a significantly higher

chance of seeing a larva object when in close proximity to an eclosed egg and vice versa.

Since the internal object detection process depends on both the YOLOv5m model and the lump estimation model, both should be considered when optimizing the internal process as a whole. Using CNNs for the lump estimation model was attempted. However, it was quickly abandoned in favor of a more conventional neural network due to the complexity of CNNs and less than ideal performance. Revisiting CNNs and using the entire image as the input to the model instead of extracting continuous features could be a viable method to optimize the predicted egg count. Furthermore, extending the existing neural network with more complex features could be equally viable.

Optimizing the internal object detection architecture is not only dependent on optimizing each separate part. Making larger changes to the structure of the architecture could be similarly productive. One such change could be changing the models from being fully trained offline to work as online models. Since the web application provides the user with functionality to add, change, and delete detections, this user input could be used to train the models further while the system is running. The lump estimation model is especially ideal for this change since the user is expected to provide the actual count for each lump, which could then easily be converted into new training data.

Modifying the data

As with any tool dependent on a trained machine learning model, extending the data used in the training process is a viable next step. One main contribution of this work is the diverse data setups used in the data, which could be further extended to include even more, possibly domain or institution-specific, setups. The data setups covered here represent the *industry-standard*, but further training might be necessary if non-standard setups are used. This necessity is further supported by the models achieving significantly worse results on the holdout setup, signifying that unseen data setups are problematic.

Beyond extending the data, work could also be focused on attempting to increase the quality of the current data. One significant issue with the data is the clumping of objects and how this is currently handled. The lump class, created through our solution to this problem, is problematic since it is only relevant to the internal object detection process and not to the final user. Since the user only cares about eggs, eclosed eggs, and larvae objects, lumps obfuscate the objects of interest. The lump class also negatively impacts the performance metrics. Changing the final object count by aggregating multiple objects, and causing some objects to be different from the ground truth while not being technically wrong, are both negative effects of the lump class that could be avoided.

Currently, all data from the different setups feature the *Drosophila melanogaster* fly. The fruit fly is an ideal initial focus due to its popularity in academic experiments. However, since the web application is not designed to be specific to *Drosophila*, supporting further species such as conventional house flies or soldier flies would be an ideal next step. Technically, if the lump estimation features are removed, the web application is generalizable to any object detection and classification task. It should be noted that this also requires re-training the YOLOv5m model and removing the lump estimation step.

More features

Finally, we acknowledge that the web application could be extended with more features to increase usability. Currently, the only preprocessing applied to the data before being evaluated by the models is tiling. More advanced preprocessing, such as cropping the image to only include the area of interest, could be provided as an option to the user. Furthermore, usability features aimed at making the internal processing more transparent would be relevant. This could, for example, be achieved by showing the user the live progress of a job. Many tools providing advanced functionality to users also commonly provide features to walk new users through the system's functionality. Due to non-standard characteristics of the system, such as lump handling, this might be relevant to include.

Acknowledgements

We want to give a special thanks to Dalin Zhang and Miao Zhang for their work in supervising the project. Furthermore, we would like to thank Natasja Krog Noer, Torsten Nygård Kristensen, and Stine Frey Laursen from the department of Chemistry and Bioscience at Aalborg University for their domain expertise and contribution of data.

Bibliographical remarks

Parts of the article are based on our article from the previous semester [29]. The abstract and section 1 are rewritten to include new results and article structure. Further, we have updated section 1 to include details regarding the comparative deep learning study conducted in this article. Section 2, *Existing methods for counting Drosophila*, is copied from last semester as we found there had not been significant progress within the field.

Section 3 is significantly extended to include more deep learning models relevant for object detection. Section 4 is heavily modified to include our new dataset, various data setups, our proposed lump estimation neural network, and the process of detecting objects using tiles. However, section 4.2, where we introduce YOLOv5, is mostly copied with only minor changes to make the section shorter without reducing coherence.

The introduction to section 5, *Proposed software*, is heavily inspired by our work in the previous semester, while the remainder of the section is rewritten to reflect the new software architecture and user interface.

While Intersection Over Union and the Average Precision performance metric in section 6.1 is influenced by our work last semester, the section has been significantly shortened while replacing both the regression performance metric and the experiment using the regression performance metric.

References

[1] Hugo Bellen, Chao Tong, and Hiroshi Tsuda. 100 years of drosophila research and its impact on vertebrate neuroscience: A history lesson for the future. *Nature reviews. Neuroscience*, 11:514–22, 04 2010.

[2] Zhasmine Mirzoyan, Manuela Sollazzo, Mariateresa Allocca, Alice Maria Valenza, Daniela Grifoni, and Paola Bellosta. Drosophila melanogaster: A model organism to study cancer. *Frontiers in Genetics*, 10:51, 2019.

[3] Berrak Ugur, Kuchuan Chen, and Hugo J. Bellen. Drosophila tools and assays for the study of human diseases. *Disease Models Mechanisms*, 9(3):235–244, 03 2016.

[4] Thomas Hunt Morgan. Sex limited inheritance in drosophila. *Science*, 32(812):120–122, 1910.

[5] Linda W Van Laake, Thomas F Lüscher, and Martin E Young. The circadian clock in cardiovascular regulation and disease: Lessons from the Nobel Prize in Physiology or Medicine 2017. *European Heart Journal*, 39(24):2326–2329, 12 2017.

[6] GEOFFREY KEIGHLEY and E. B. LEWIS. DROSOPHILA COUNTER. *Journal of Heredity*, 50(2):75–77, 03 1959.

[7] Dominic Waithe, Peter Rennert, Gabriel Brostow, and Matthew D. W. Piper. Quantifly: Robust trainable software for automated drosophila egg counting. *PLOS ONE*, 10(5):1–16, 05 2015.

[8] Pierre Nouhaud, François Mallard, Rodolphe Poupardin, Neda Barghi, and Christian Schlötterer. High-throughput fecundity measurements in drosophila. *Scientific Reports*, 8, 03 2018.

[9] Arpit Yati and Sutirth Dey. Flycounter: a simple software for counting large populations of small clumped objects in the laboratory. *BioTechniques*, 51(5):347–348, 2011. PMID: 22054548.

[10] Evgenia K. Karpova, Evgenii G. Komyshev, Mikhail A. Genaev, Natalya V. Adonyeva, Dmitry A. Afonnikov, Margarita A. Eremina, and Nataly E. Gruntenko. Quantifying drosophila adults with the use of a smartphone. *Biology Open*, 9(10), October 2020. © 2020. Published by The Company of Biologists Ltd.

[11] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. (available on arXiv:1505.04597 [cs.CV]).

[12] Hao Dong, Guang Yang, Fangde Liu, Yuanhan Mo, and Yike Guo. Automatic brain tumor detection and segmentation using u-net based fully convolutional networks. In María Valdés Hernández and Víctor González-Castro, editors, *Medical Image Understanding and Analysis*, pages 506–517, Cham, 2017. Springer International Publishing.

[13] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28:91–99, 2015.

[14] Junkang Zhang, Haigen Hu, Shengyong Chen, Yujiao Huang, and Qiu Guan. Cancer cells detection in phase-contrast microscopy images based on faster r-cnn. In *2016 9th international symposium on computational intelligence and design (ISCID)*, volume 1, pages 363–367. IEEE, 2016.

[15] Riccardo Rosati, Luca Romeo, Sonia Silvestri, Fabio Marcheggiani, Luca Tiano, and Emanuele Frontoni. Faster r-cnn approach for detection and quantification of dna damage in comet assay images. *Computers in Biology and Medicine*, 123:103912, 2020.

[16] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934, 2020.

[17] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2017.

[18] Ilyas Sirazitdinov, Maksym Kholiavchenko, Tamerlan Mustafaev, Yuan Yixuan, Ramil Kuleev, and Bulat Ibragimov. Deep neural network ensemble for pneumonia localization from a large-scale chest x-ray database. *Computers Electrical Engineering*, 78:388–399, 2019.

[19] Martin Zlocha, Qi Dou, and Ben Glocker. Improving retinanet for ct lesion detection with dense masks from weak recist labels. In Dinggang Shen, Tianming Liu, Terry M. Peters, Lawrence H. Staib, Caroline Essert, Sean Zhou, Pew-Thian Yap, and Ali Khan, editors, *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019*, pages 402–410, Cham, 2019. Springer International Publishing.

[20] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection. 2019.

[21] Andre Araujo, Wade Davenport Norris, and Jack Sim. Computing receptive fields of convolutional neural networks. *Distill*, 2019.

[22] Linlin Zhu, Xun Geng, Zheng Li, and Chun Liu. Improving yolov5 with attention mechanism for detecting boulders from planetary images. *Remote Sensing*, 13(18), 2021.

[23] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, and Jun-Wei Hsieh. Cspnet: A new backbone that can enhance learning capability of cnn, 2019.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1904–1916, 2015.

[25] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation, 2018.

[26] Shifeng Zhang, Cheng Chi, Yongqiang Yao, Zhen Lei, and Stan Z Li. Bridging the gap between anchor-based and anchor-free detection via adaptive training sample selection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9759–9768, 2020.

[27] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[28] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[29] Christian Schmidt Godiksen and Kristoffer Nøddebo Knudsen. Counting laboratory populations of drosophila melanogaster eggs using yolov5. `https://projekter.aau.dk/projekter/da/studentthesis/counting-laboratory-populations-of-drosophila-melanogaster-eggs-using-yolov5(d3a83ef7-c746-45db-83f9-82d89329e73a).html`, 2022.

[30] Addie Ira Borja Parico and Tofael Ahamed. Real time pear fruit detection and counting using yolov4 models and deep sort. *Sensors*, 21(14), 2021.

[31] Liang Zheng, Hengheng Zhang, Shaoyan Sun, Manmohan Chandraker, and Qi Tian. Person re-identification in the wild. 04 2016.

[32] Fetulhak Abdurahman, Kinde Anlay Fante, and Mohammed Aliy. Malaria parasite detection in thick blood smear microscopic images using modified yolov3 and yolov4 models. *BMC Bioinformatics*, 22(1):112, Mar 2021.

[33] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[34] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, Jun 2010.

[35] Rafael Padilla, Wesley L. Passos, Thadeu L. B. Dias, Sergio L. Netto, and Eduardo A. B. da Silva. A comparative analysis of object detection metrics with a companion open-source toolkit. *Electronics*, 10(3), 2021.

[36] Ersin Kilic and Serkan Ozturk. An accurate car counting in aerial images based on convolutional neural networks. *Journal of Ambient Intelligence and Humanized Computing*, Jul 2021.

[37] Issam Laradji, Pau Rodriguez, Freddie Kalaitzis, David Vazquez, Ross Young, Ed Davey, and Alexandre Lacoste. Counting cows: Tracking illegal cattle ranching from high-resolution satellite imagery, 2020.

[38] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks, 2016.

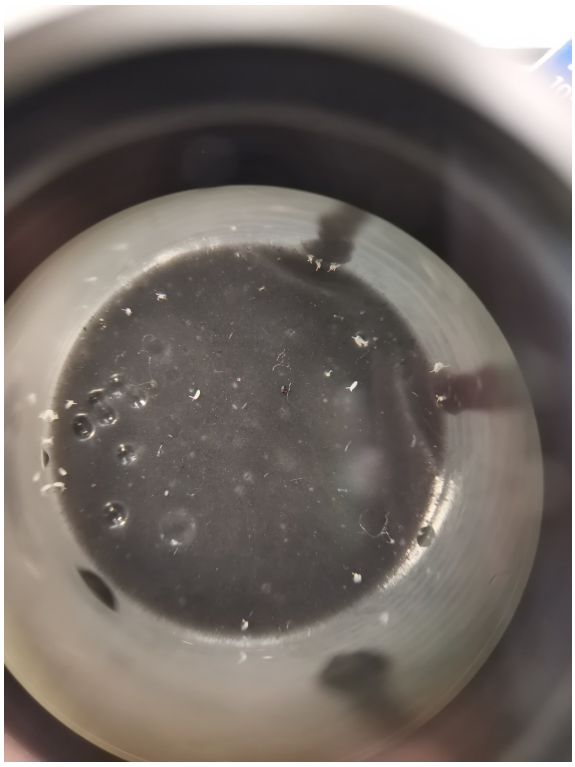
[39] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.

[40] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot MultiBox detector. In *Computer Vision – ECCV 2016*, pages 21–37. Springer International Publishing, 2016.

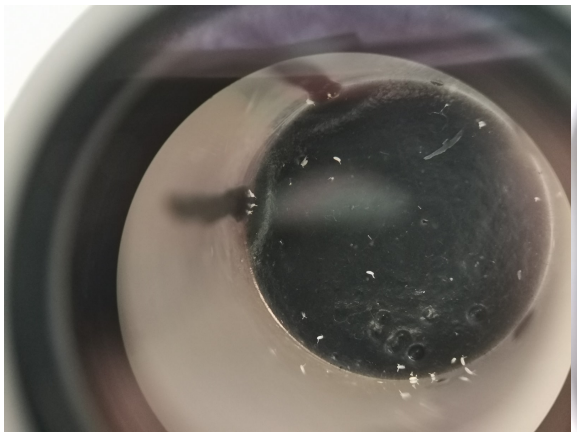
Appendix

A. Six data setups

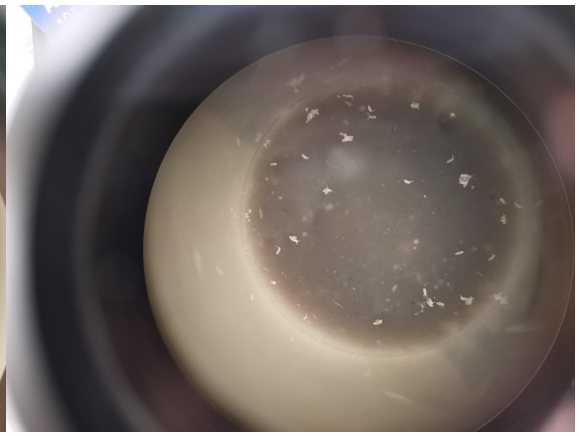
(a) Blue-green (3000x4000).



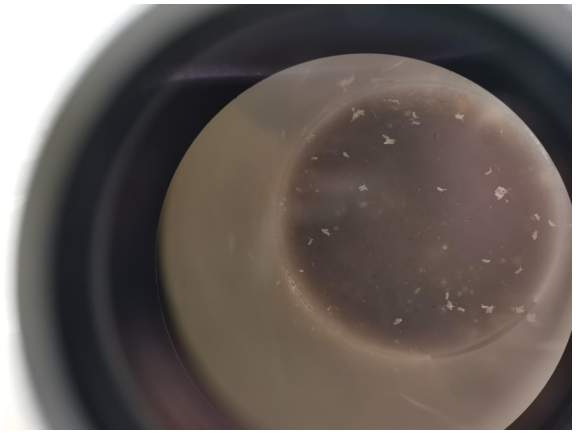
(b) Fresh black (3000x4000).



(c) Frozen black (4000x3000).



(d) Fresh grey (4000x3000).

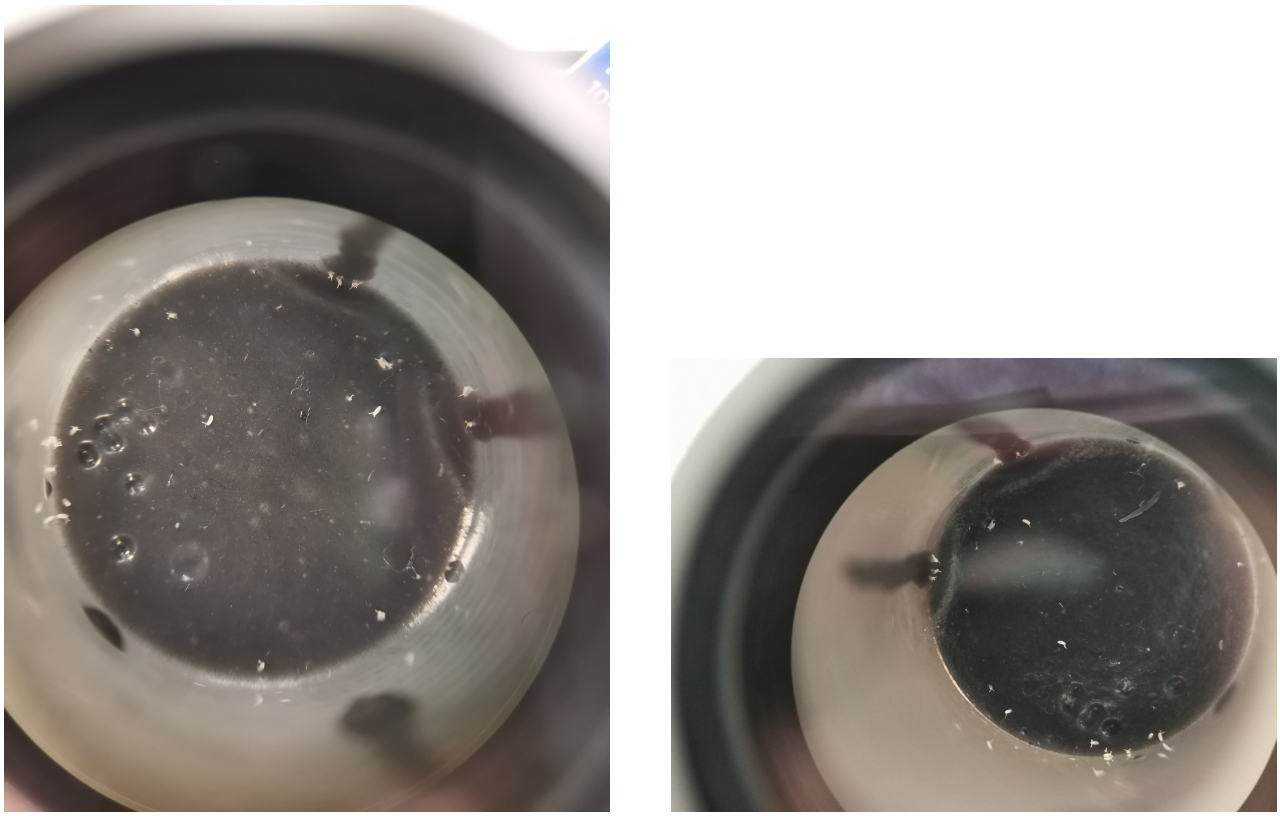


(e) Frozen grey (4000x3000).

(f) Pink spoon (4032x1908).

Figure 9. The dataset consists of six different setups and has three different resolutions.

B. Petri dish storage conditions



(a) Petri dish with black medium before being frozen down.

(b) Same Petri dish as in (a) after being frozen. At first glance, it can be difficult to see it is the same Petri dish.

Figure 10. Illustration of the dissimilarity between the fresh and the frozen Petri dishes.

C. User interface of proposed web application

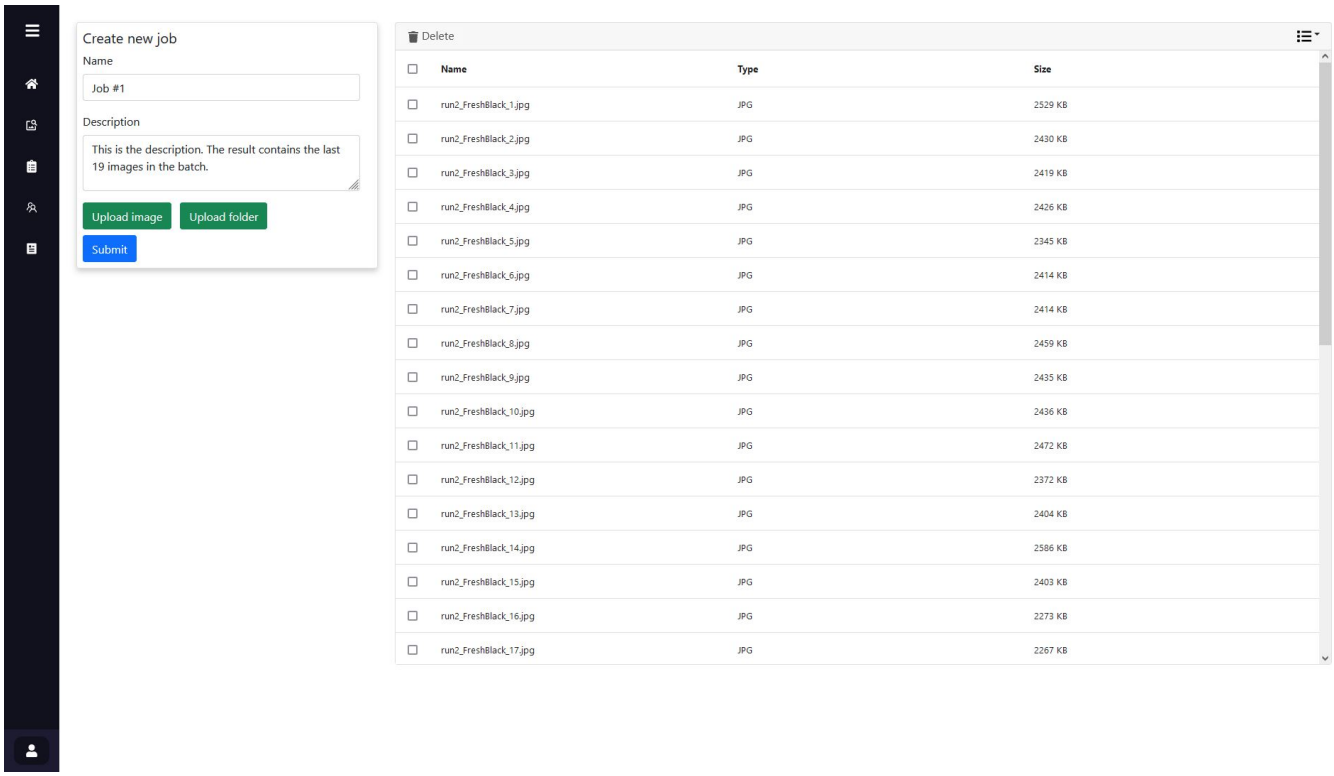


Figure 11. User interface for uploading data to do object detection and classification.

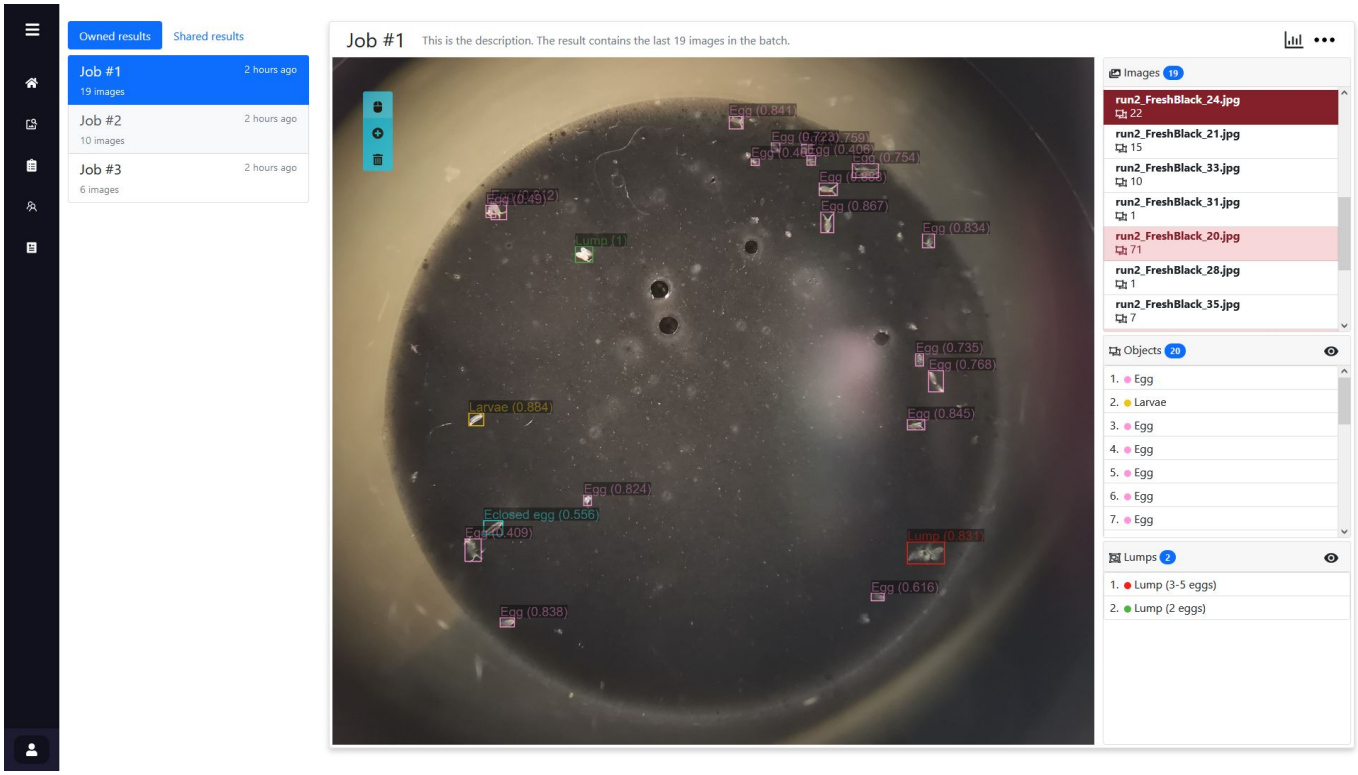


Figure 12. User interface for viewing the object detection and classification results.

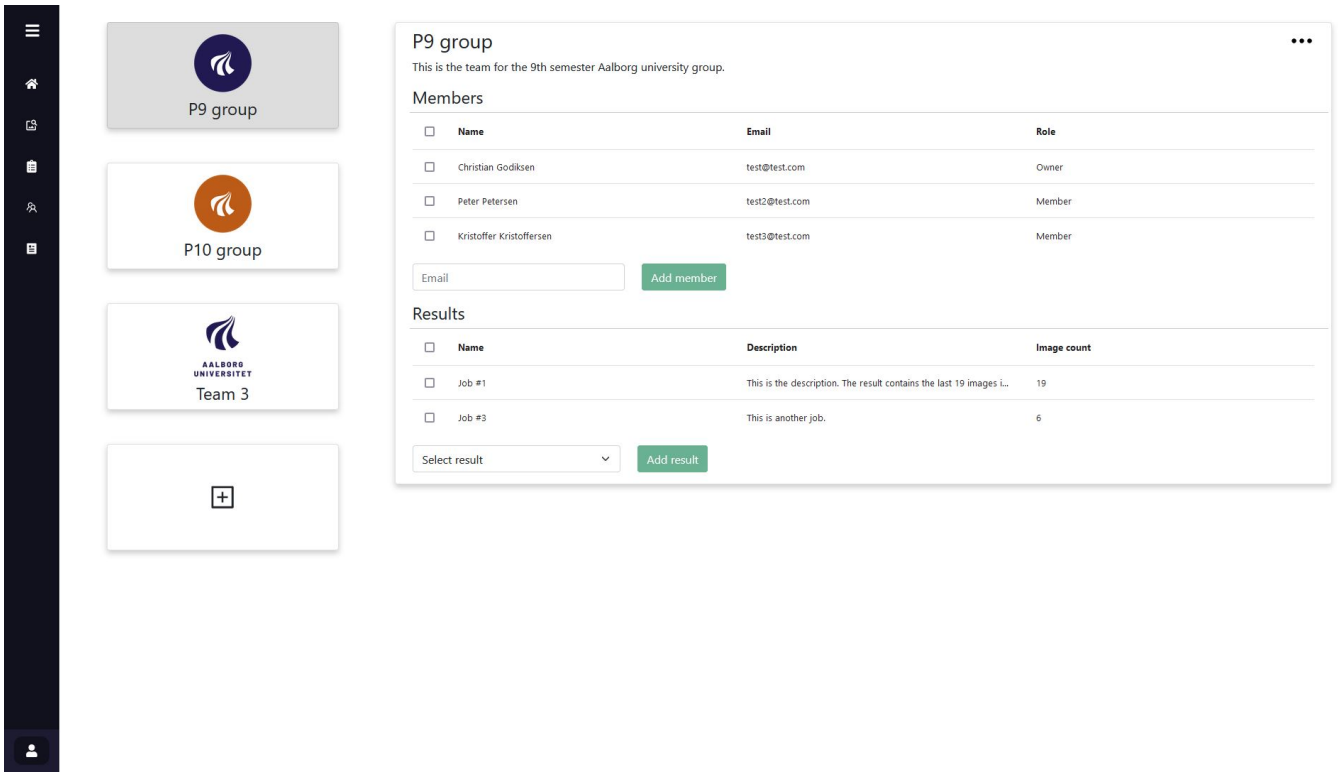


Figure 13. User interface for creating teams and sharing results with other users.

D. Inference results to examine models

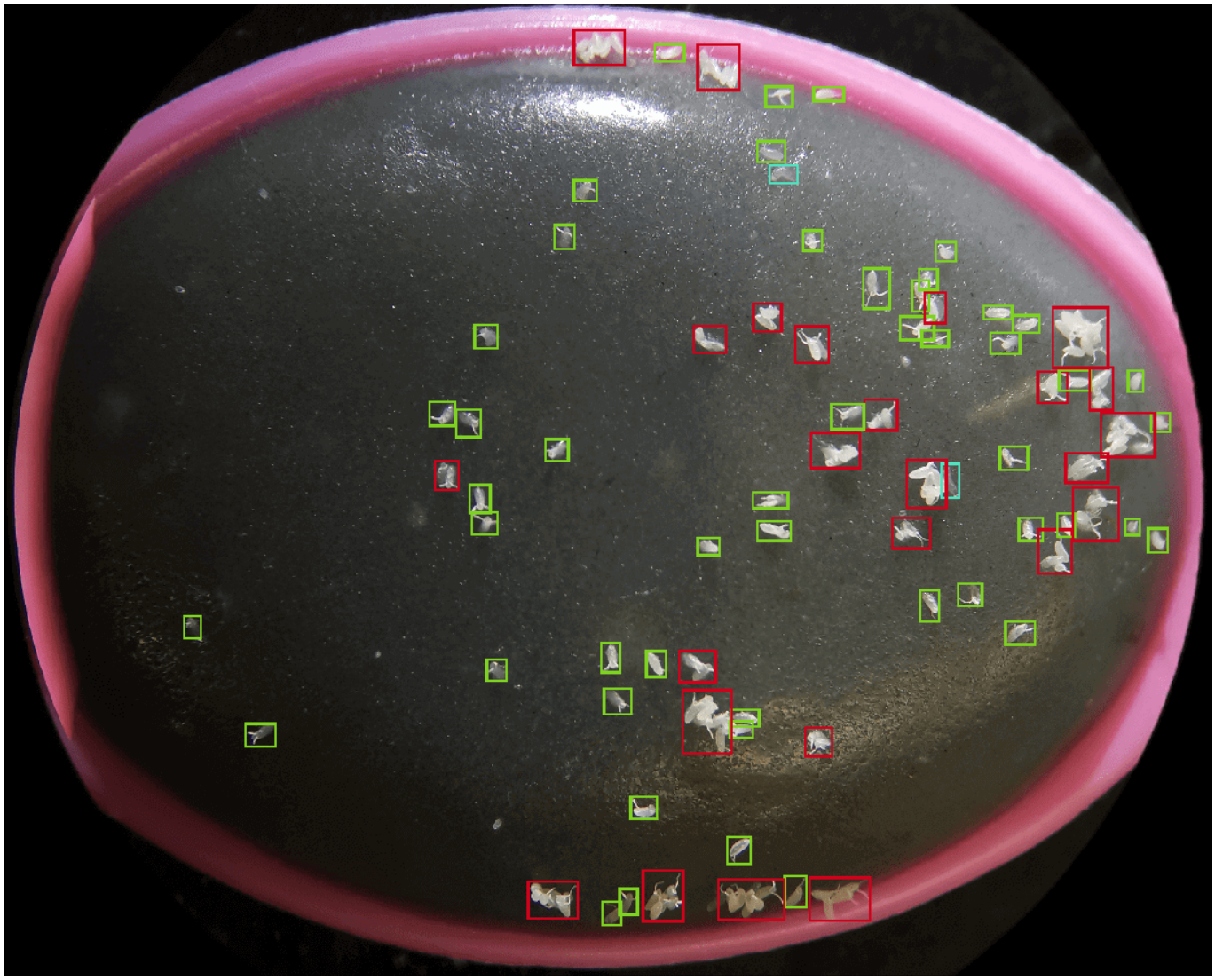


Figure 14. This image contains 53 eggs, 2 larvae and 26 lumps making it the most object dense image in the test set. The ground truths are annotated as follows: green rectangle indicates an egg, teal rectangle indicates a larva, while lumps are indicated by a red rectangle.

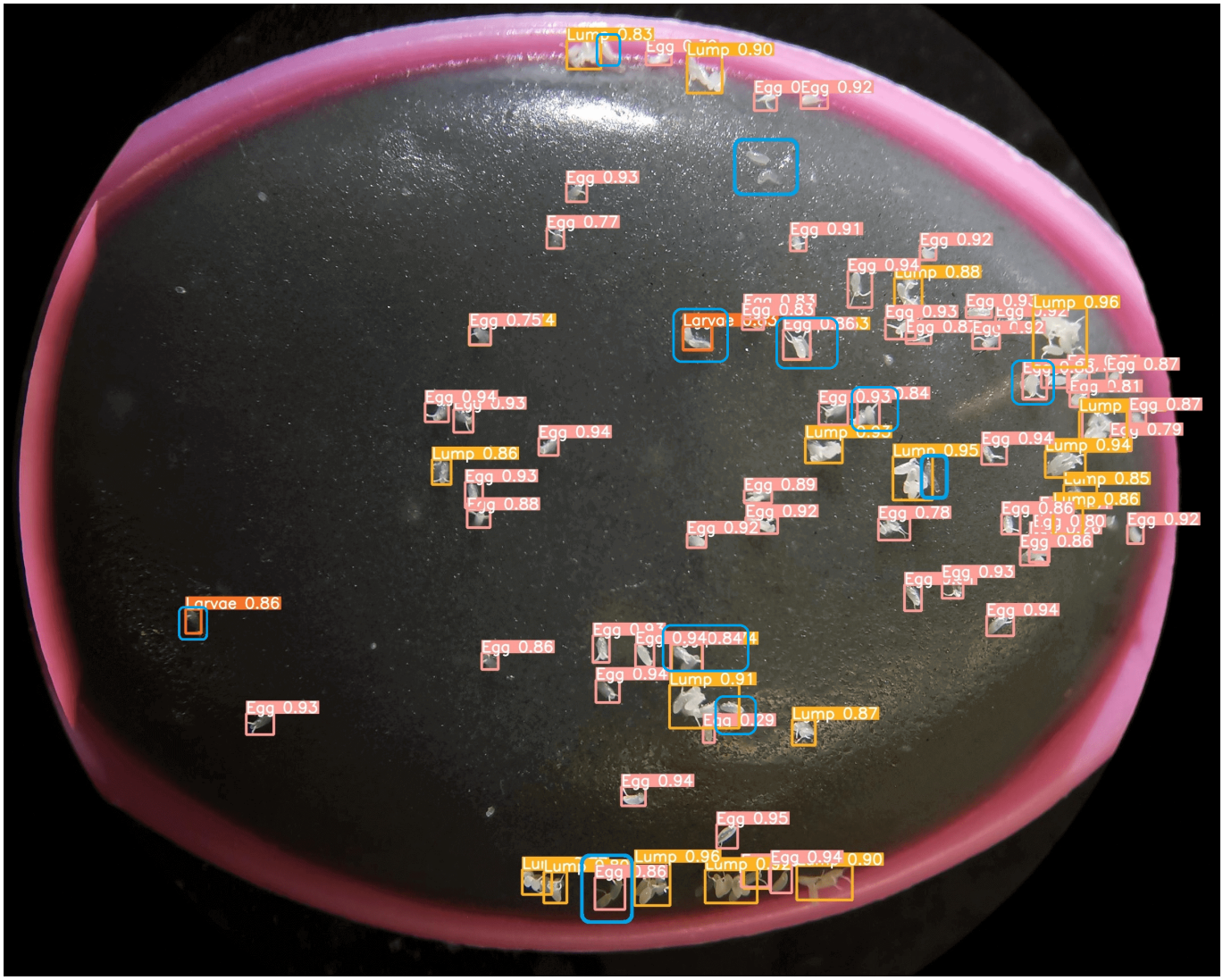


Figure 15. **YOLOv5m on full-sized image. Errors include: Missing mostly-transparent egg and larva, eggs misclassified as larvae, two close eggs being detected as a single egg, and missing a larva partially covered by a lump.**

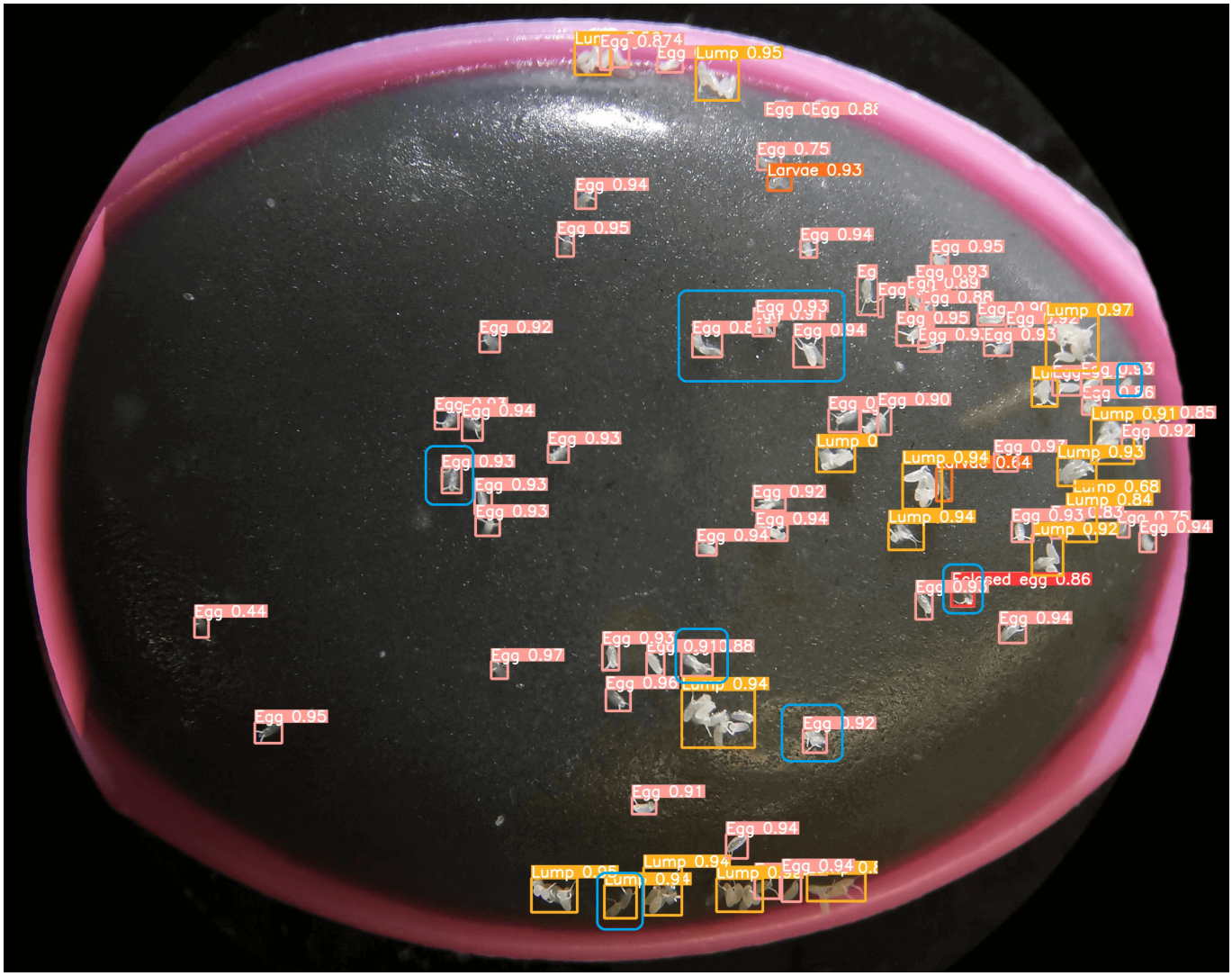


Figure 16. **YOLOv5m on tiled image. Errors include: Neighboring eggs misclassified as a single egg, neighboring eggs misclassified as multiple eggs rather than a lump, and an egg misclassified as an eclosed egg.**

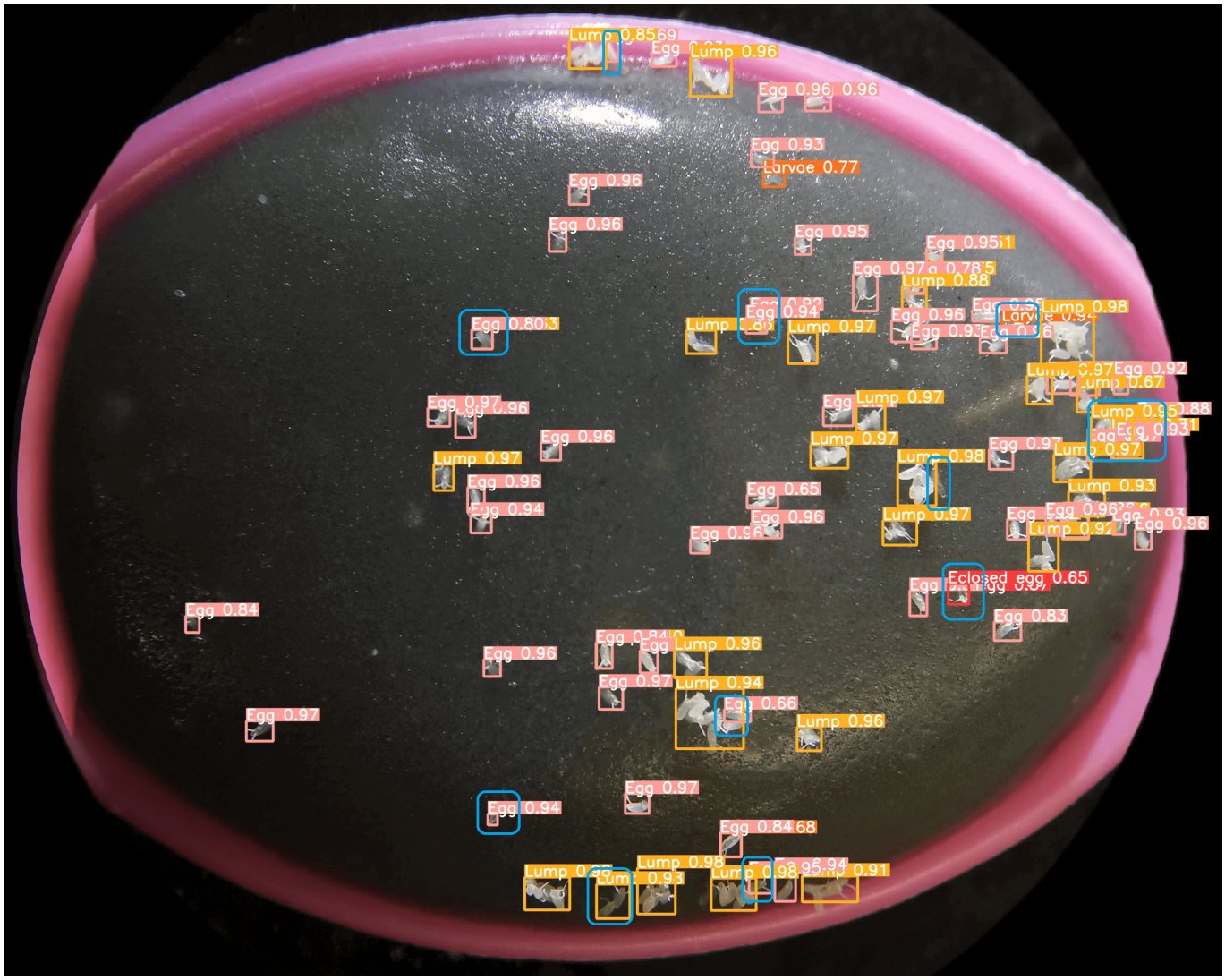


Figure 17. **YOLOv5x6 on full-sized image. Errors include an egg being marked both as an egg and part of a lump, noise being marked as an egg, two eggs being marked as one lump instead of two individual eggs, an egg misclassified as an eclosed egg, and an egg misclassified as a larva.**

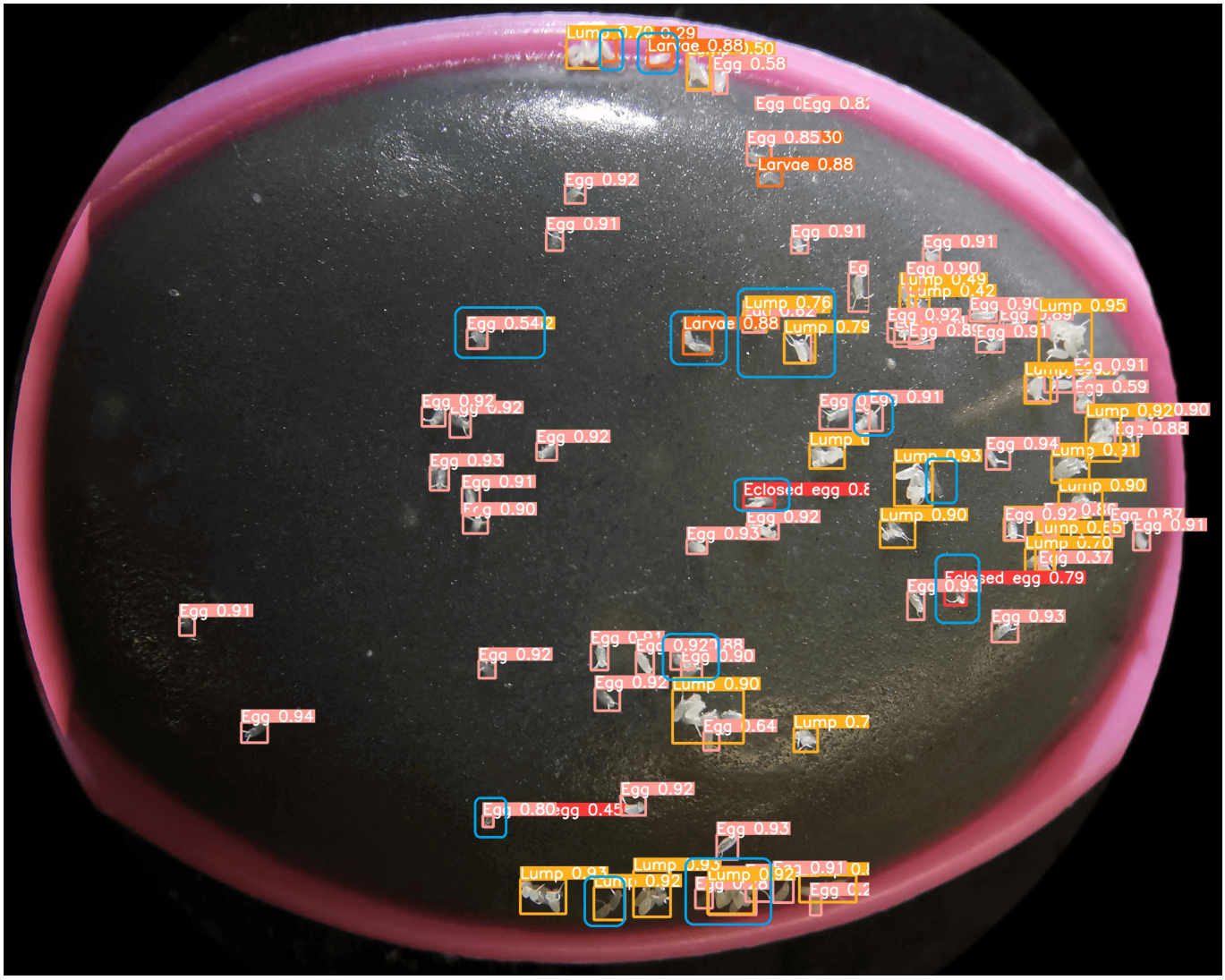


Figure 18. **YOLOv5x6 on tiled image. Errors include a single egg getting marked as both an egg and a lump, eggs misclassified as larvae or eclosed eggs, a partially covered larva missed, eggs classified as both part of a bigger lump as well as an individual egg, noise misdetected as an egg, multiple eggs forming a lump marked as individual eggs rather than a lump.**

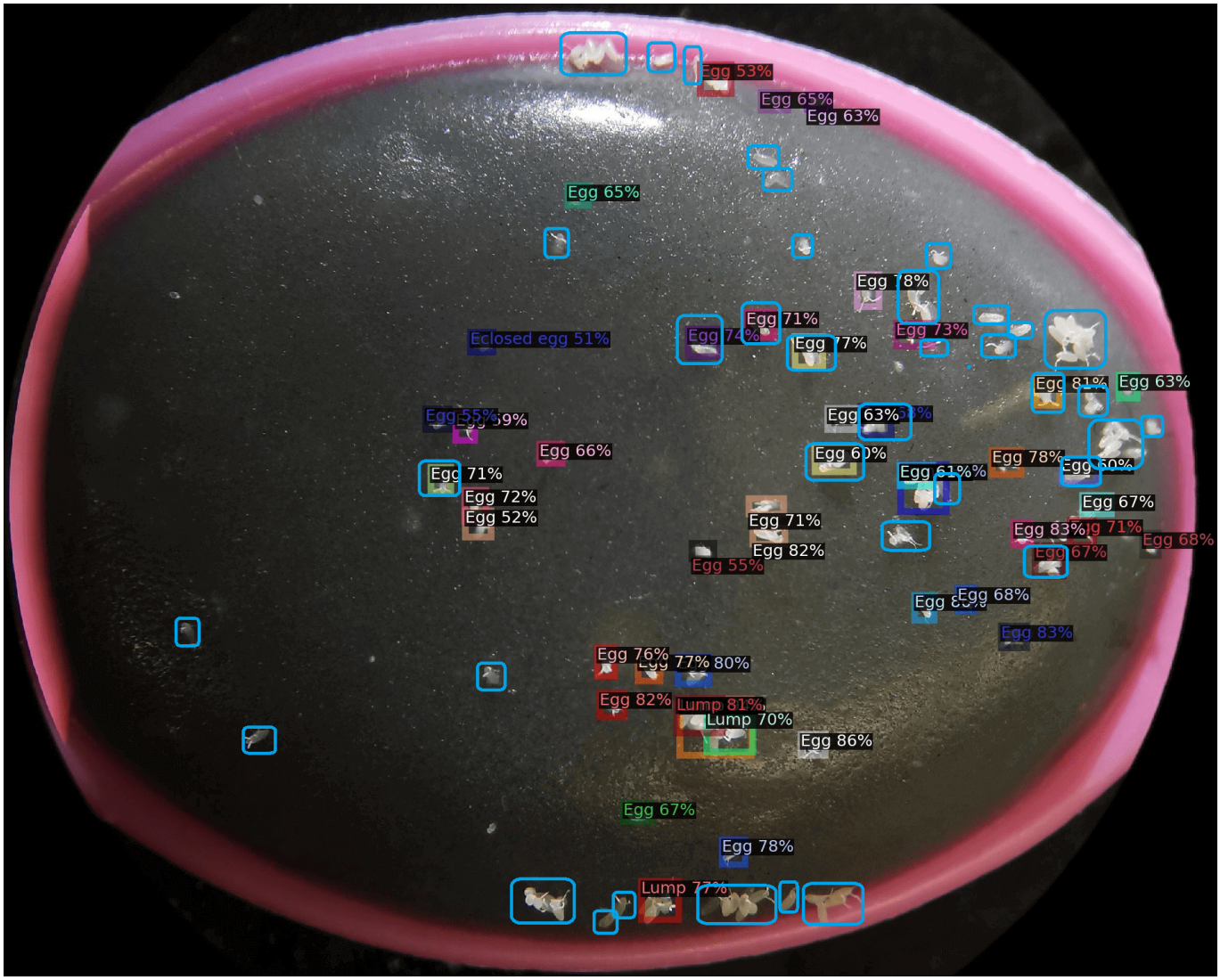


Figure 19. **Faster R-CNN on full-sized image. Errors especially include large quantity of missed objects as well as lumps misclassified as being single eggs. Errors also include a lump being detected as multiple lumps.**

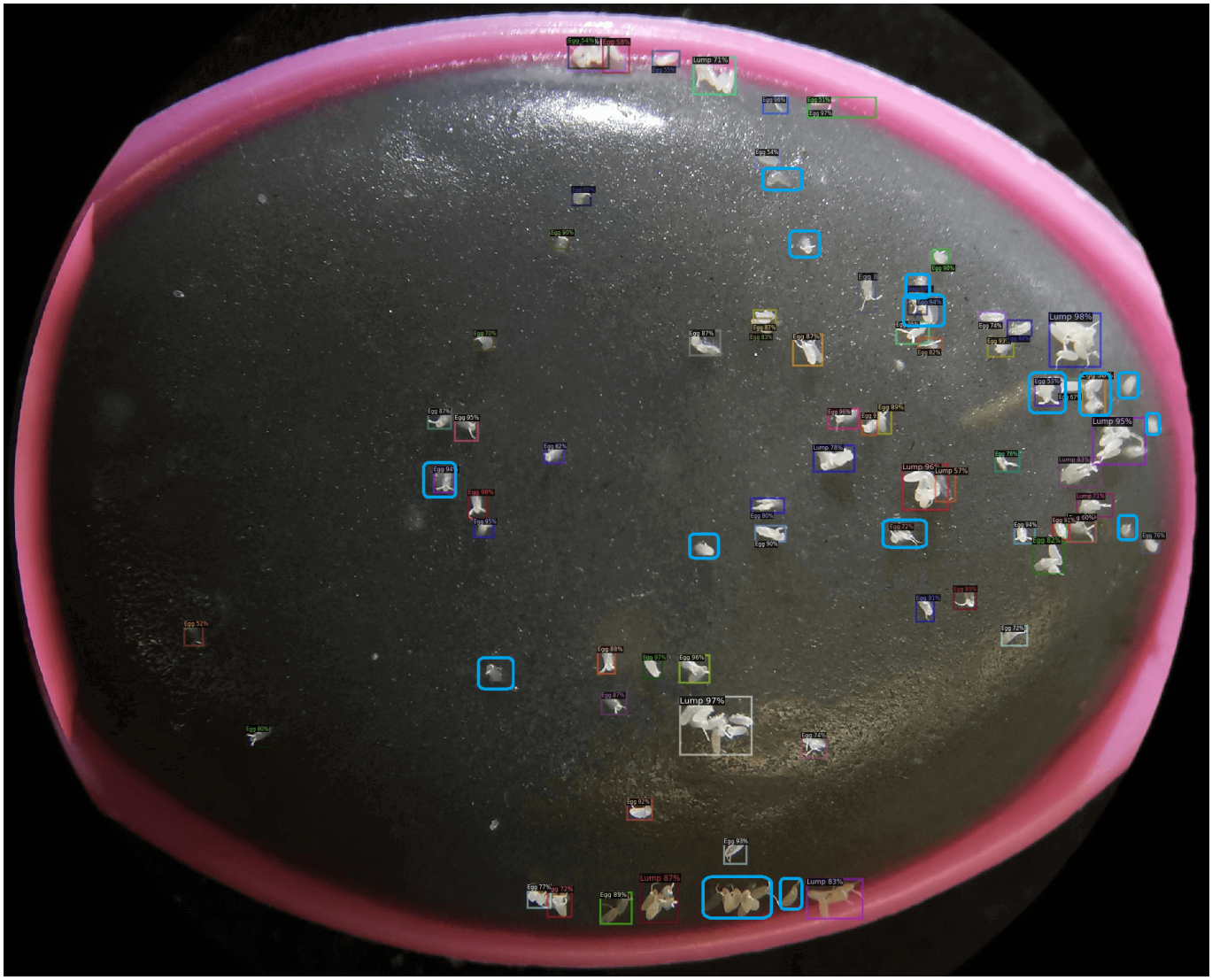


Figure 20. **Faster R-CNN on tiled image. Errors especially includes many missed objects and again lumps misclassified as being single eggs. In the top of the container we also observe an egg being detected but with an out-of-proportion bounding box.**

Figure 21. RetinaNet on full-sized image. Like Faster R-CNN, errors especially includes false negatives with both eggs and lumps being missed. Notably, all the lumps in the top of the container are missed. Most eggs, except those to the right, are also missed. Further, some lumps are misclassified as being a single egg.

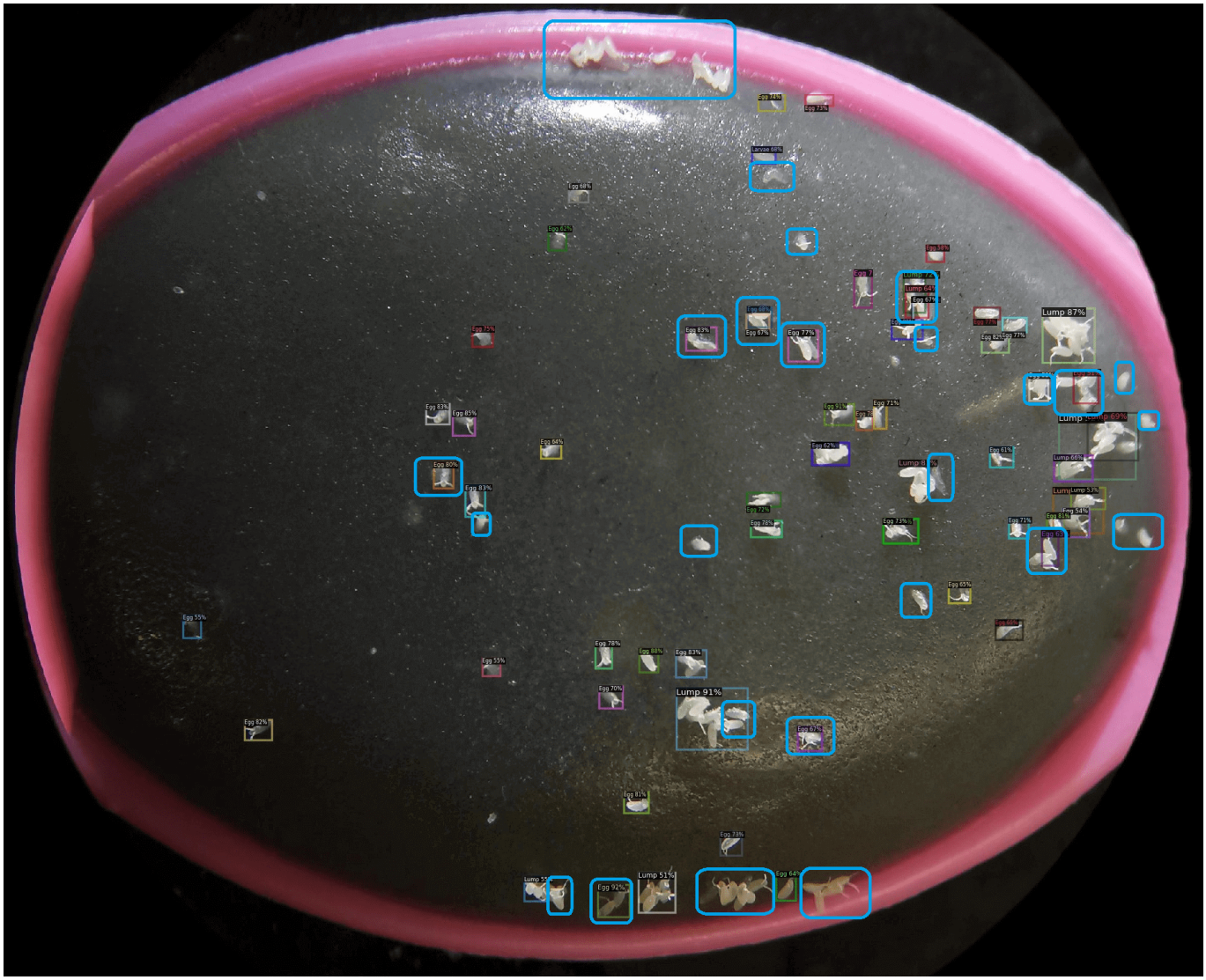


Figure 22. RetinaNet on tiled image. Using tiles, more of the eggs are now correctly detected compared to using full-sized images, but many objects are still missing and some lumps are again misclassified as being single eggs.

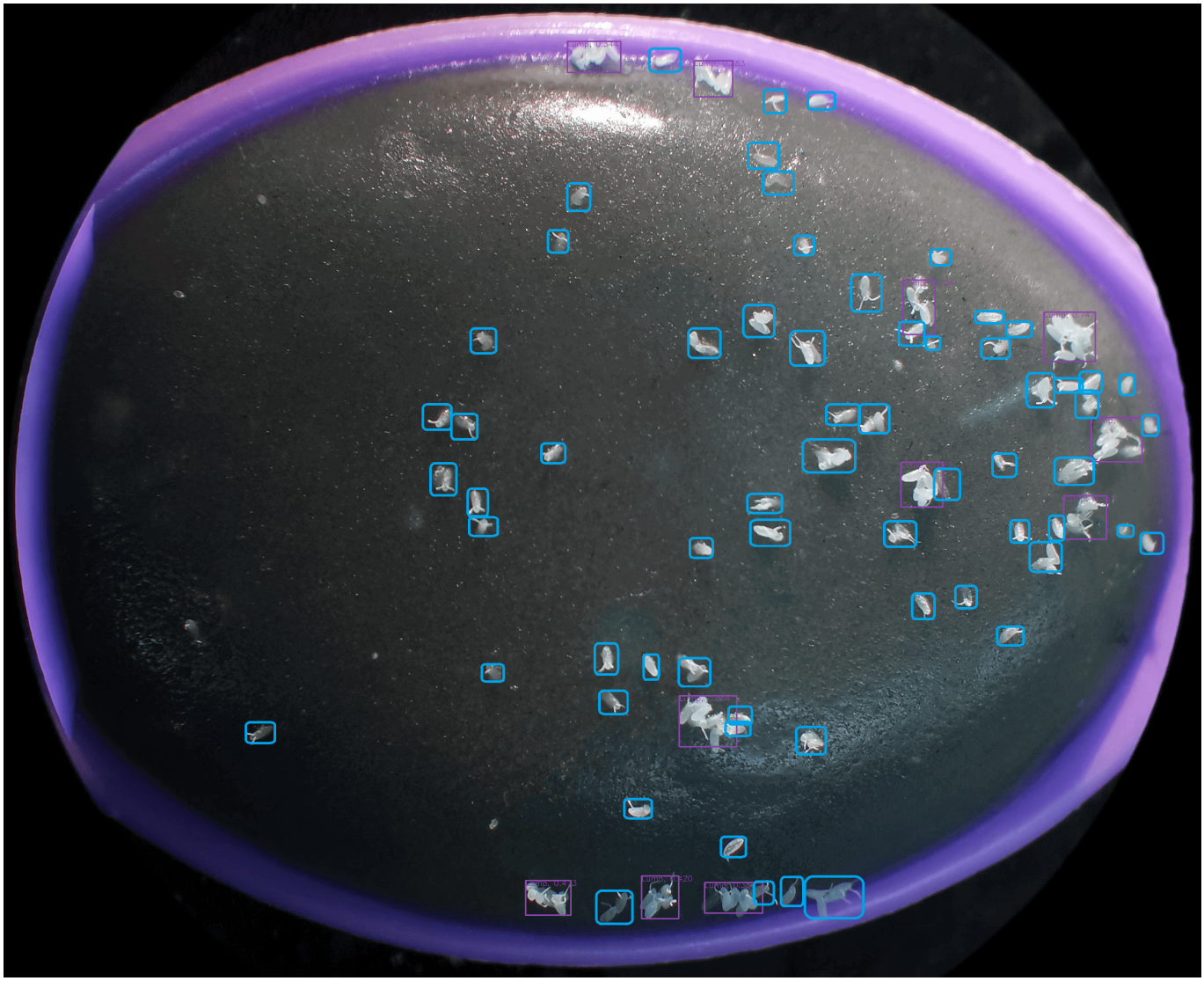


Figure 23. **EfficientDet on full-sized image. Only large lumps are detected, while smaller lumps, larvae, and eggs are all ignored. While this model has a high precision, it has a very low recall.**

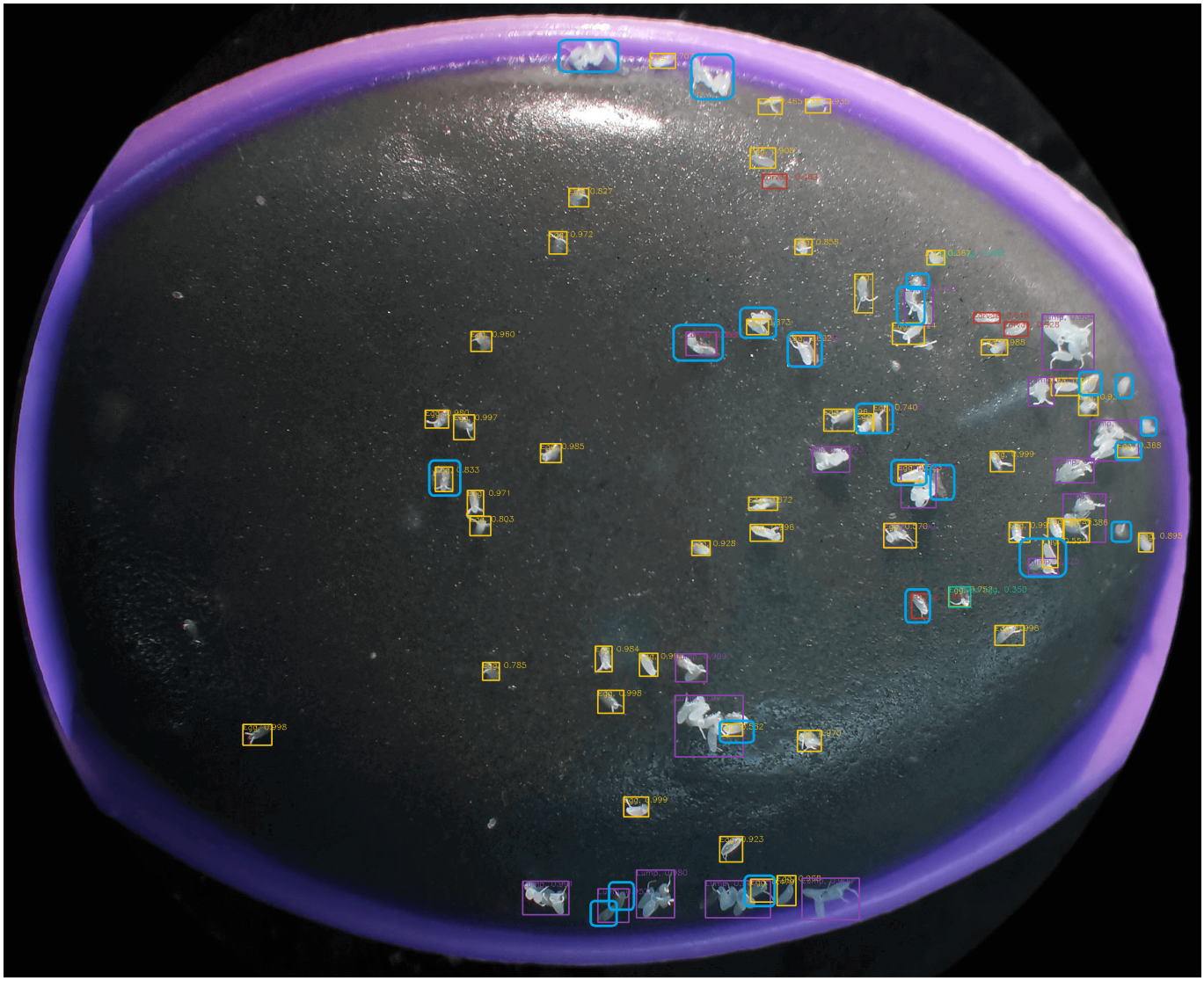


Figure 24. **EfficientDet on tiled image. Compared to the tiled model, we immediately observe tremendous differences, and most eggs are now correctly detected, though a few are still not detected. There are still some misclassifications, including lumps detected as a single egg, two eggs detected as one larva, and single eggs misclassified as a larva. We also observe some neighboring eggs misclassified as a lump rather than two individual eggs. At the top of the container, we also observe misdetected lumps.**