Baltic J. Modern Computing, Vol. 13 (2025), No. 1, 75-95
https://doi.org/10.22364/bjmc.2025.13.1.05

Mobile Device-Based Ants Recognition and Tracking System: Methodology and Frameworks

Dmytro KUSHNIR

Department of Computer Engineering, Lviv Polytechnic National University,
Stepana Bandery 12, Lviv, 79013, Ukraine

dmytro.o.kushnir@lpnu.ua

ORCID 0000-0001-6623-3382

Abstract. This research introduces a practical framework for ex situ ants recognition and tracking, enabling the continuous monitoring of their movements. This framework includes an iOS-based client application with an embedded Visual Intersection Over a Union (V-IOU) tracking module and a You Only Look Once (YOLO) recognition model. Another part of a framework is a scalable system for autonomous annotating, training, and converting the model to mobile format. The tracking algorithm is integrated into a Swift application using the JavaScriptCore library, while the Yolo model is integrated using the CoreML framework. To ensure system accuracy and stability, the method of improved clasterization K-means++ is employed. Simultaneously, the Affine Quantization method is used keep the model size as small as possible. An experimental benchmark on an indoor ant colony was conducted, using various recognition models to assess the accuracy and productivity of the system. The results confirm the practicality of our methods and frameworks for real-time small object detection, demonstrating their applicability in real-world scenarios.

Keywords: Affine Quantization, Ants Tracking, K-means++ Clustering, CoreML iOS Framework, JavaScriptCore Library, Object Detection, Real-time Tracking, Scalable System, V-IOU Tracking, YOLO Model

1. Introduction

The study of insect behavior, particularly ants, has long fascinated researchers due to these tiny creatures' complex social structures and behaviors. Accurate tracking and recognition of ants can provide invaluable insights into their collective behavior, foraging patterns, and colony dynamics (Popp et al., 2024). Beyond ecological curiosity, understanding ant movements and interactions can show how resources are distributed within colonies and help identify patterns in their decision-making and organization.

While studying ants in their natural habitat is important, this research focuses on ex-situ monitoring in controlled settings. By observing ants in a controlled environment, the study aims to systematically track their movements, identify areas of high activity, and analyze behavioral trends such as trail formation and resource

allocation. These insights are difficult to obtain through traditional methods, often relying on labor-intensive manual observations or non-scalable technologies.

To address these challenges, recent advancements in object detection and tracking technologies provide promising solutions. The You Only Look Once (YOLO) model, known for its high-speed and accurate object detection capabilities, has been successfully applied in various fields, including wildlife monitoring and urban surveillance. Similarly, tracking algorithms such as the Visual Intersection over Union (V-IOU) (Bochinski et al., 2018) tracker have shown promise in maintaining the identity of moving objects over time. Building on these technologies, this research aims to develop a practical, scalable solution for continuous ant monitoring by integrating these state-of-the-art methods.

1.1. Purpose of the study

The primary purpose of this study is to develop a practical and scalable framework for continuously monitoring ants using a combination of the V-IOU tracking module and the You Only Look Once (YOLO) recognition model. This framework is implemented on an iOS platform, making it accessible and user-friendly for field researchers and enthusiasts. By integrating these advanced technologies, the study aims to provide an efficient and accurate solution for real-time ant tracking, significantly reducing the need for manual observations.

1.2. Contributions of the study

This study makes several key contributions to the field of entomology and computer vision:

Innovative Framework: Introduction of a comprehensive iOS-based framework that integrates V-IOU tracking and YOLO recognition for real-time monitoring of ants.

Advanced Integration: Implement the tracking algorithm within a Swift application using the JavaScriptCore library and the embedding of the YOLO model using the CoreML framework.

Optimization Techniques: Improved clustering methods (K-means++) and Affine Quantization enhance system accuracy and stability while keeping the model lightweight and responsive.

Scalability: Development of an autonomous system capable of annotating, training, and converting models for mobile deployment, ensuring adaptability to various environments.

Experimental Validation: Experimental benchmarks were conducted on an indoor ant colony to validate the framework's accuracy and productivity and demonstrate its practicality in real-world scenarios.

1.3. Organization of the study

The remainder of this paper is structured as follows: Section 2 presents the literature review, offering an overview of insect tracking and recognition technologies research. Section 3 details the methodology, including the design and implementation of the

recognition model, its training, clusterization, and the integration of the tracking method. Section 4 discusses the frameworks used, focusing on the scalable system for autonomous annotating, training, and converting the recognition model to a mobile format. Section 5 describes the experimental setup and results, including the benchmark conducted on an indoor ant colony and the results obtained. In Section 6, the discussion analyzes the findings, compares them with existing solutions, and explores the study's implications. Finally, Section 7 provides the conclusions, summarizing the study's contributions, suggesting areas for further research, and recommending improvements for future implementations.

2. Related Work

Recent research on car traffic monitoring using UAVs conducted by Gudauskas et al. (2024) demonstrated the importance of quick custom object recognition and tracking in real-time environments. Similarly, advancements in insect monitoring technologies have enabled real-time tracking and behavioral analysis of various species. For instance, the IntelliBeeHive system integrates machine learning and computer vision to monitor honeybee activity, detect pests, and provide insights to beekeepers (Smith et al., 2023). The AROBA system further highlights the use of autonomous observation technologies for honeybee colonies, emphasizing continuous monitoring without human intervention (Ulrich et al., 2024). These innovations demonstrate the potential of leveraging advanced tools for studying insect behavior and health, paving the way for real-time, scalable monitoring solutions.

Building on these foundations, this study focuses on recognizing and tracking ants using mobile device-based systems. While honeybee monitoring often relies on fixed systems or specialized setups, my approach targets a more flexible and portable framework that can be adapted for recognizing and tracking various objects, including but not limited to ants. By implementing the YOLO model (Bochkovskiy et al., 2021) for efficient real-time object detection and the V-IOU algorithm (Bochinski et al., 2018) for robust tracking, the framework offers a novel solution tailored to ants' unique behaviors. This aligns with previous research on small dynamic object recognition, such as that conducted in my Ph.D. thesis (Kushnir, 2023), which emphasized the challenges of adapting such systems for mobile platforms.

On the other hand, integrating such recognition and tracking systems on embedded devices, as I did in the previous research (Kushnir, 2022), creates a high load on the Graphical Processing Unit (GPU), which, in theory, can be improved by using a Neural Processing Unit (NPU) from a mobile device. Also, it is worth noting that previously implemented recognition and tracking algorithms executed on separate Docker environments should be migrated to mobile devices like iOS without losing efficiency.

To resolve such issues, a practical approach involves utilizing the k-means++ clustering algorithm proposed by Arthur et al. (2007) on the YOLO recognition model to divide recognized clusters of objects into corresponding classes correctly. This is vital in the scope of multiple small object recognition, like ants, which can move fast and fit on each other. Research conducted by Wang et al. (2023) with an improved VV-YOLO model confirms such assumptions, showing an improved real-time vehicle

recognition process.

Additionally, it is crucial to minimize recognition model weights on mobile devices without sacrificing efficiency. As Li et al. (2023) demonstrated, model quantization methods can achieve this, where a fully quantized network with 4-bit quantization showed an acceptable accuracy loss.

CoreML tools, as presented by Marques (2020), were utilized to integrate the model into an iOS mobile device. These tools facilitate the seamless conversion and deployment of machine learning models onto iOS platforms, ensuring efficient performance and leveraging the advanced hardware capabilities of Apple's devices. By using CoreML, the model can take advantage of on-device processing, which enhances speed and privacy by minimizing the need for data to be sent to external servers. This integration is particularly beneficial for applications requiring real-time processing so that it can be used for ant processing.

For tracking module injection, I propose using the JavaScriptCore Swift framework analyzed by Novák (2020). This framework allows for the seamless integration of JavaScript code within Swift applications, enabling efficient execution and manipulation of JavaScript within the iOS environment in real-time. That can be achieved by creating JSContext for each recognition thread in the Swift application, efficiently increasing real-time tracking.

The decision to integrate the tracking algorithm using JavaScriptCore was motivated by its capacity to support a cross-platform implementation strategy. JavaScriptCore allows the tracking logic, written in JavaScript, to be readily adapted for deployment in various environments, including web servers and embedded systems, thereby enhancing the scalability and versatility of the proposed framework. Additionally, this choice facilitates modularity by decoupling the algorithm's implementation from the iOS application, ensuring its reuse across multiple platforms without significant modifications. In the context of this research, JavaScriptCore enabled precise and reproducible experimentation in a controlled indoor setting, while its dynamic runtime capabilities allowed iterative fine-tuning of the tracking logic.

3. Methodology

3.1. General workflow

To fulfill the research goals, a system for autonomous annotating, training, and converting the Recognition model to CoreML Mobile format is proposed (Figure 1).

This system receives datasets of images annotated either automatically or manually for specific classes of objects, trains the model, and converts it to the CoreML format. On the client side, an iOS Swift application retrieves the converted model and the necessary metadata. The tracking module is also integrated using the JavaScriptCore framework to facilitate real-time tracking of the required object classes. Each system module will be discussed in detail in the “Frameworks” section of the article.

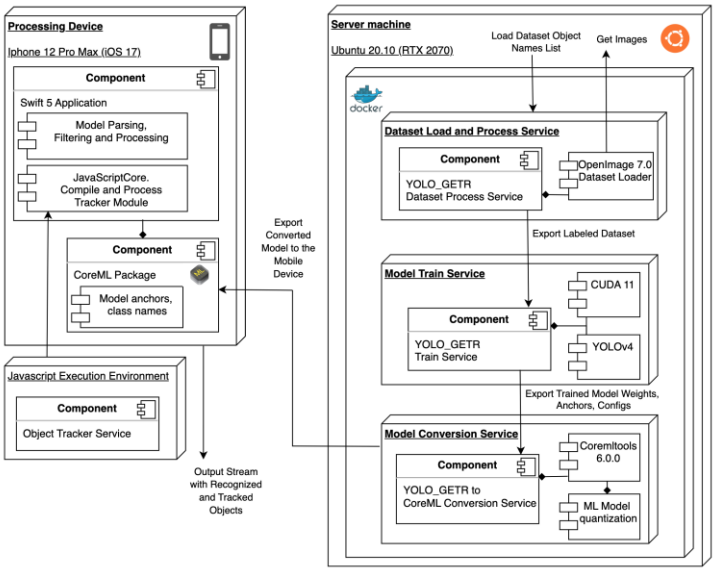

Figure 1. General workflow of the system

This system receives datasets of images annotated either automatically or manually for specific classes of objects, trains the model, and converts it to the CoreML format. On the client side, an iOS Swift application retrieves the converted model and the necessary metadata. The tracking module is also integrated using the JavaScriptCore framework to facilitate real-time tracking of the required object classes. Each system module will be discussed in detail in the "Frameworks" section of the article.

3.2. Dataset

The dataset used in this research is formed from two sources: a manually labeled indoor ants dataset and an automatically generated dataset from Open Images for a specified object class. These two datasets train the YOLOv4 model (Bochkovskiy et al., 2021).

The manually labeled dataset was created using the open-source tool LabelStudio (Tkachenko et al., 2024), which allows quickly annotating multiple small objects on the image frame, which is vital for ants labeling. This resulted in a dataset comprising 500 images (Kushnir, 2022).

Conversely, the autonomous dataset was created for a specific class of ants using OpenImage 7.0. This process involved forming a list of input classes and downloading the required number of annotated images and the necessary metadata for the recognition model. Com bining the automatically downloaded dataset with the manually labeled one enhances the diversity of the annotated images, which is expected to positively impact the model weight creation during the training process.

Ultimately, this approach increased the total number of annotated images to 1500, significantly enhancing the dataset for more robust model training.

3.3. Recognition model and evaluation metrics

Among the Convolutional Neural Networks (CNN) researched in my Ph.D. thesis (Kushnir, 2023), YOLOv4 was proposed as the recognition model due to its efficiency and accuracy in real-time object detection. The model's architecture, including the backbone and neck, remained unchanged. Two configurations were tested: a "tiny" model with two output layers and a larger model with three output layers. The RETR model represents a custom-created model designed for "Recognition and Tracking". The hyperparameters for these models are detailed in Table 1.

Table 1. Parameters used in model training

CNN Model	Batch	Subdivisions	height /width	Learning Rate (LR)	Decay	Momentum
Yolov4_retr	64	16	416/416	0.002	0.0005	0.95
Yolov4_retr_tiny	64	8	416/416	0.001	0.0005	0.9

The gradient descent algorithm used for optimizing hyperparameters was Nesterov Accelerated Gradient (NAG), utilizing specific values for Decay and Momentum. The loss function employed, particularly for the localization component, was the Complete Intersection Over the Union (CIoU) method. This approach minimizes the normalized distance between two analyzed objects, enhancing detection accuracy.

Several metrics were proposed to benchmark recognition results during the research to evaluate the model's performance. Among these, confusion matrix metrics are particularly important for recognition tasks using supervised learning and imbalanced classes. This method categorizes recognition objects into four categories based on the combination of positive response and algorithm: true positive (**TP**), true negative (**TN**), false positive (**FP**), and false negative (**FN**).

Let's introduce the intersection over union (IoU) minimization filter to better understand how these values are calculated.

$$\boldsymbol{IOU}(\boldsymbol{A},\boldsymbol{B}) = \frac{A \cap B}{A \cup B}$$

This metric determines how much the recognition region and the reference object overlap in internal volume. A recognition result is considered a true positive (TP) detection if the IoU equals or exceeds 0.5. An FP state occurs when the IoU values are below 0.5. FN is a state where true positive objects were not detected or were below the set threshold. FP defines a state where false negative results are interpreted as positive.

Based on that, the following metrics were applied:

$$\boldsymbol{R}\ (recall) = \left(\frac{TP}{TP + FN}\right)$$

$$\boldsymbol{FNR}\ (false\ negative\ rate) = 1.0 - R$$

$$\boldsymbol{P}\ (precision) = \left(\frac{TP}{TP + FP}\right)$$

$$\boldsymbol{FPR}\ (false\ positive\ \ rate) = 1.0 - P$$

Recall indicates the proportion of TP objects identified by the classifier. Precision indicates the proportion of objects identified as TP that are truly positive. FPR indicates the expected duration of FP states. FNR represents the fraction of all FNs that still yield positive results. The lower the values of FNR and FPR, the higher the model's performance.

It is important to note that recall and precision are independent of the input class size ratios. If the proportion of TP objects is significantly smaller than the number of TN class objects, these metric indicators will show the correct functioning of the tested algorithms.

There are two main ways to obtain a single quality criterion using recall and precision: the F-measure and the average precision score.

$$\boldsymbol{F1} = 2\ \times \left(\frac{P\ \times\ R}{P\ +\ R}\right)$$

The feature of obtaining the harmonic mean for the F-measure is that such a measure is close to zero. Thus, higher metric accuracy is achieved with incorrect sample distribution.

However, obtaining the harmonic mean for the F-measure often results in a measure close to zero, which achieves higher metric accuracy when the sample distribution is incorrect. When it is necessary not only to predict an object's class but also to perform ranking–solving object recognition tasks of search and localization, the mean Average Precision (mAP) metric is used; this metric is vital for calculating the average classification indicators across all categories (Kushnir, 2023).

$$\boldsymbol{mAP}\ = \frac{\sum_{i=1}^{n} AP_i}{n}$$

The mAP metric determines the level of confidence for recognition objects. Therefore, this method is appropriate for assessing the effectiveness of a CNN model during training. Additionally, it can be used to compare the implemented YOLO models at the inference stage.

It is important to note that real-time GPU, CPU, and NPU load performance metrics must be added for recognition tasks on mobile devices with limited hardware capabilities.

Let us introduce concepts that define real-time performance: **FLOPs** (number of floating-point operations per second), **FPS** (frames per second), the display time of results on the screen after the start of processing – $\boldsymbol{T_{frame}}$, the median time for performing an object prediction operation – $\boldsymbol{T_{predict}}$, and the median time for loading the model onto the tested device – $\boldsymbol{T_{load}}$.

Thus, to achieve the research goals, it is necessary to determine the inference speed of the developed recognition model on mobile devices, ensuring it is close to real-time at approximately 24 FPS. Additionally, it is essential to assess the usage of CPU, GPU, and other acceleration devices using metrics such as FLOPs, $\boldsymbol{T_{predict}}$, $\boldsymbol{T_{frame}}$, and $\boldsymbol{T_{load}}$.

3.4. K-means++ clustering during model training

Training the developed CNN model can be effectively divided into two stages. The first stage uses the available hardware to generate the model's primary weight coefficients. During this stage, particular attention is given to applying the clustering method to calculate recognition anchors and set input hyperparameters. These include the number of training iterations, the determination of block and sub-block sizes of the CNN, and additional parameters based on the selected optimization algorithm. A crucial part of this stage is determining the LOSS value, which significantly impacts the model's performance.

In the second stage, the trained CNN model is fine-tuned by verifying whether the current weight coefficient with the best mAP value has reached its highest value.

The original YOLO uses the k-means unsupervised clustering method to form recognition anchors (anchor boxes), a technique for dividing the input image into a grid of cells to which the object recognition region is attached. This method determines the position, width, and height of the object relative to the center of the grid cell, using k-means to identify the most optimal anchor sizes.

The principle of k-means involves iteratively using the Euclidean distance to calculate the distance between a manually set number of clusters. During the expectation phase, the distance is calculated from the initial cluster center (centroid) to the center of each object.

A disadvantage of this algorithm is the need to know the number of clusters in advance; the result of clustering and execution time – O(n) depends on the choice of initial centroids. If the initial centroids are chosen randomly, it may lead to convergence errors.

Therefore, the modified k-means++ algorithm is proposed for generating recognition anchors. This algorithm addresses the problem of random centroid placement. The algorithm prioritizes points at the maximum distance from the centroid to avoid overlapping two points.

A test with 2000 random data points was created on four separate clusters to compare the evaluation of the two clusters (Figure 2 and Figure 3) (Kushnir, 2023).

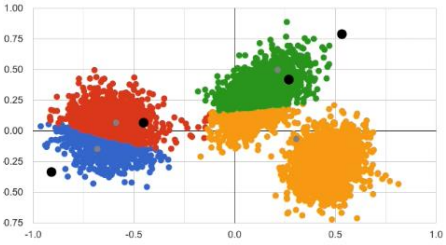

Figure 2. K-means clustering

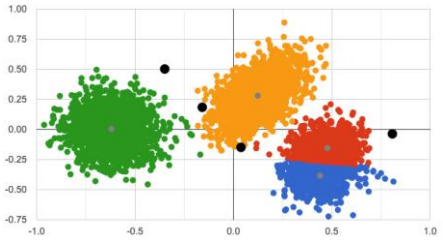

Figure 3. K-means++ clustering

Black dots on the images represent the starting coordinates of centroids, while grey dots represent the central coordinates of the clusters. As seen from the analysis results, the current organization of initial centroids in K-means clustering could have been

more successful, as data from one cluster intersected with data from another. However, with the k-means++ algorithm, points in the clusters are distributed correctly. It is worth noting that although convergence errors are significantly minimized, this comes at higher computational costs than the k-means algorithm.

Let us define the steps to apply to our YOLO model with k-means++ clustering integration, accommodating any output layer number (Algorithm 1).

Algorithm 1. Clustering using K-means++ for forming Anchors in YOLOv4

1. Determine the height and width of recognition rectangles from all recognition regions. Set the initial iteration i = 0;
2. **Repeat** iteration i;
3. Select a recognition anchor as the initial point (centroid) of the input cluster from all recognition regions;
4. Calculate the distance $D(x_i)$ between the centroid of all recognition regions and the centroid of existing recognition anchors. Calculate the probability $P(x_i)$ for each recognition region selected as the next centroid:

$$x \in X, \qquad P_i = \frac{D(x_i)^2}{\sum_{i=1}^{n}(x_i)^2}$$

The further the recognition region is from the initial centroid, the higher the probability of its selection;
5. Use the IoU minimization filter for each region and recognition anchor to select the most likely recognition anchors for the current recognition region. The higher this value, the more likely the object belongs to the desired class;
6. **Repeat** until the recognition regions do not change;
7. Obtain the final recognition anchors.

To summarize, for recognition tasks during model training, the k-means++ algorithm increases the accuracy of anchor selection, which can significantly improve error determination in image classification.

3.5. Tracking method and module injection

This study has selected the V-IOU tracking method, which I thoroughly tested in my previous research (Kushnir, 2022). On the other hand, in the current study, the JavaScriptCore framework, operating within an isolated virtual JavaScript environment, was employed to integrate this tracking method into the mobile Swift application. However, since JavaScript runs in a single thread (event loop), it imposes limitations on hardware performance. While this may be sufficient in a web browser, achieving high performance under heavy CPU and GPU loads is critical for iOS systems.

To address this issue, a multithreaded approach in Swift 5 was proposed. Each thread was given shared access to the JavaScriptCore instance (JsContext) and a separate asynchronous message queue, allowing tracking tasks to be distributed across multiple threads. Additionally, caching identical objects helps reduce the impact of the tracking process on performance.

The proposed algorithm outlines the process of initializing a JsContext instance and using it as a separate module for object tracking. Key steps include setting a batch limit for processing requests, initializing the JsRunner class, creating a shared global JsCore context, and asynchronously processing data through the tracking module.

To optimize the integration of modules on resource-constrained mobile platforms, the module size was minimized using tools like Rollup. JavaScript supports several module design patterns, and the UMD (Universal Module Definition) pattern was selected for this study. This pattern operates across various platforms, ensuring that each generated UMD module functions in an isolated environment and can be successfully integrated into the iOS mobile platform.

4. Frameworks

4.1. Scalable system for model annotation, training, and converting to mobile format

Achieving scalability and efficiency is crucial in developing machine learning models, especially for mobile platforms. Containerization presents a robust solution to meet these needs by isolating various system components through virtualization tools like Docker. This method enables modularization, allowing each container to function in its independent environment, ensuring flexibility, ease of deployment, and separation from other containers. As a result, the object recognition system crafted for mobile platforms is divided into distinct services within Docker containers, each responsible for a specific stage in the neural network model lifecycle: data annotation, model training, and model conversion to a mobile-friendly format. The proposed structural diagram of this system is illustrated in Figure 4.

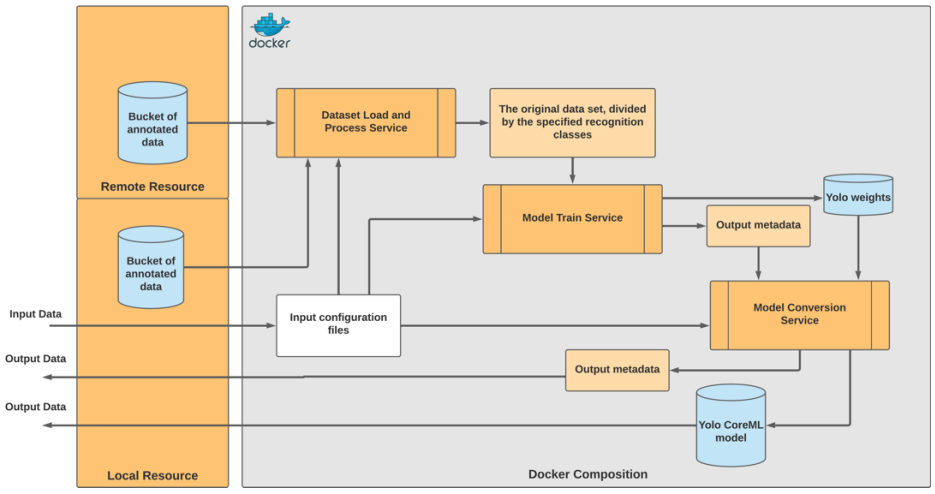

Figure 4. Structural scheme for autonomous data annotating, model training, and converting to mobile format system.

To ensure scalability, the framework supports simultaneous processing of multiple datasets by running several instances of the relevant containers. For instance, datasets can be annotated, trained, and converted concurrently, enabling efficient use of computational resources and faster model preparation. As a result, the system allows users to generate the required model weights simply by specifying the names of the classes to be trained. For example, classes such as '*ant-messor-structor'*, '*ant-camponotus-fellah'*, and others can be easily defined and processed, ensuring flexibility in handling diverse datasets.

Creating a resulting neural network for a mobile platform can be broken down into three core stages.

4.1.1. Annotation service

This service identifies and processes input datasets according to the specific recognition classes the neural network requires. It automates the loading and annotation of training and testing datasets from public databases such as OpenImage 7.0 (Figure 5). The service outputs annotated classes for each image in a format suitable for YOLO model training.

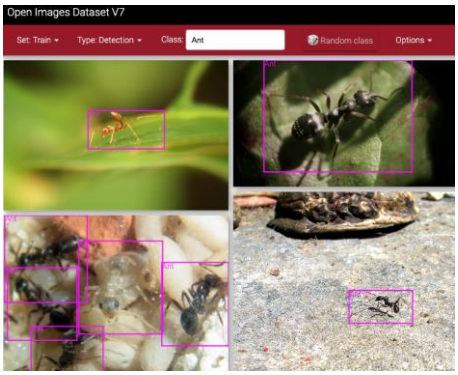

Figure 5. Annotated dataset for a specific class of objects.

Such images and annotated data are loaded from the Amazon Web Services S3 bucket by such link with defined input parameters for images, directories, and class names:

curl –location –request GET 'aws s3 cp s3://open-image-dataset/:imageDir/: image/:datasetDir/:ClassesName'

The proportion of train and validation images can also be defined, but by default, it is set as 70 train images to 30 validation images.

Additionally, the service supports manually adding annotated images using tools like Label Studio, providing flexibility for incorporating custom data in case automatically annotated data is insufficient.

4.1.2. Training service

Once the data is annotated, the model training service scales the training process to leverage available hardware resources, whether CPU or GPU. This service is designed to be versatile, supporting various operating systems, including Unix-based systems like Ubuntu, macOS, and Windows. The training service uses the input data, specified hyperparameters, loss functions, and clustering methods to produce optimized weight coefficients.

The training process also involves fine-tuning these weight coefficients to identify the most optimal features for different sizes of recognized objects. The output includes the trained model and analytical metrics such as mAP, essential for evaluating model performance.

It is important to note that the local directories obtained from the annotation step must be mounted to the Docker container through volumes. The input device should have the Compute Unified Device Architecture (CUDA) scaling system and the cuDNN library for GPU operations to enable process parallelization. CUDA access is then granted from within the Docker container. If CUDA is unavailable, the training can be executed on the CPU, though it will be significantly slower.

4.1.3. Conversion service and model quantization

After training, the conversion service processes the final model weights, optimizing them for deployment on mobile devices. This process involves quantization using affine transformations to reduce the size of the weight coefficients, making the model more efficient for mobile applications. The service also generates and records metadata, including recognition anchors and other characteristics necessary for the model's integration into the application. Finally, the model is converted into the CoreML format, specifically into the MLPackage format, using Swift 5 and Xcode tools, and is ready for deployment on iOS devices.

Specific quantization methods are recommended to further reduce the model's weight coefficients and enhance performance. The proposed affine transformation quantization method reduces precision to 8 bits. In contrast, using a lookup table formed through k-means clustering, the quantization method can reduce precision further to 4 bits.

As a result, the weights are quantized to 8-bit/4-bit precision for floating-point numbers, which reduces the model size by 2x or 4x, respectively. However, this reduction in model size leads to a linear decrease in recognition quality. Therefore, quantization may not be necessary for smaller models with fewer than three output layers when adapting the model to mobile platforms. In such cases, this step can be skipped.

Furthermore, the developed model's precision can be increased from 16-bit half-precision to 32-bit single-precision. However, this would significantly increase the hardware requirements for object recognition tasks, which are difficult to achieve with mobile platforms.

5. Results and analysis

For evaluating the results, it is essential to identify the key challenges that recognition systems encounter using predefined metrics. One of these challenges is assessing the efficiency of the recognition model based on input parameters during training. The second challenge is benchmarking the performance, which varies depending on the type of model used. In the following section, I will examine these results in detail, thoroughly analyzing the system's performance and effectiveness. The results described below were evaluated during my Ph.D. research (Kushnir, 2023).

5.1. Recognition models efficiency with k-means++ clustering

The key metrics discussed in this study include recall (**R**), precision (**P**), true positives (**TP**), false negatives (**FN**), F1-score (**F1**), model weight size (**w**), and mean Average Precision (**mAP**). Additionally, clustering methods like K-means/**K-means++** were defined, as long as the size of the training set and the input resolution of the neural network.

For this study, four main types of YOLOv4-based neural network models were trained and implemented. These models vary by several factors: the number of output layers, with some models having two layers (tiny models) and others three layers (regular models); the maximum input image resolution, set at either 512 or 416 pixels; and the clustering method used, which was either the enhanced K-means++ or the standard K-means.

To further refine the models, a Smoothing Compression Filter (SCF) was set at 0.9, and the Intersection Over Union (IoU) threshold was set at 0.2 to minimize the impact of unlikely results. The evaluation results are shown in **Table 2,** verified in my Ph.D. research (Kushnir, 2023).

Table 2. Efficiency metrics for the developed CNN models in object recognition tasks depend on the chosen clustering method, input image resolution, and number of output layers.

Metric	**R (%)**	**P (%)**	**FN**	**TP**	**F1 (%)**	**w (MB)**	**mAP (%)**
CNN Model	↑	↑	↑	↑	↑	↑	↑
Yolov4_retr_416 (k-means++)	**91.9**	97.4	**27**	**272**	**96.77**	256	**97.2**
Yolov4_retr_512 (k-means++)	**93.7**	98.2	**19**	**284**	95.32	256	**96.2**
Yolov4_416 (k-means)	91.3	**99.2**	32	267	91.87	246	94.2
Yolov4_512 (k-means)	92.2	**98.1**	18	277	**93.787**	246	93.6
Yolov4_retr_tiny_416 (k-means++)	**86.1**	90.4	41	**268**	**91.5**	24.2	**86.92**
Yolov4_retr_tiny_512 (k-means++)	**87.4**	92.5	**38**	261	**91.8**	25.2	82.99
Yolov4_tiny_416 (k-means)	81.5	**95.4**	42	256	88.6	24.3	81.21
Yolov4_tiny_512 (k-means)	82.2	**97.2**	**39**	**262**	87.2	25.1	**83.11**

The test involved four implemented CNN models and four standard CNN models of the same type for comparison. When using the K-means++ clustering algorithm during anchor generation, the generated model shows a 5% improvement in mAP.

Additionally, it is observed that precision (P) and recall (R) mutually constrain each other: as the R-value increases, the P-value decreases in a linear progression. Thus, when using the K-means++ clustering method, the R-value increases by an average of 3-4%, while the P-value decreases by 2-3%. The F1-score is used to balance the R and P values. Therefore, the higher the F1 score, the more accurate the model is overall. On average, the F1 score improved by 5-6% for most input models.

The quantitative error values of FN and TP are linear and depend on the CNN's effectiveness during testing. At the same time, the number of FN errors is slightly higher for models with two output layers (tiny models) than their counterparts with three output layers (Figure 6).

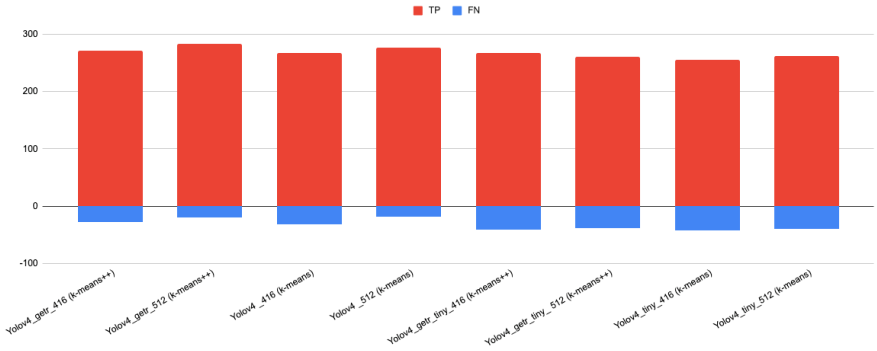

Figure 6. Comparison of FN (blue) and TP (red) values for the tested CNN models.

The difference in FN values is acceptable, as the recognition accuracy for these smaller models remains high, within the range of 85 mAP. The resolution of the input images directly influences the recognition efficiency of the CNN models. At a resolution of 512 pixels, the possible processing area for input images increases, thereby improving all efficiency characteristics of the model by 8-10%. However, the higher the resolution of the input images, the more hardware resources are required for object recognition tasks. Therefore, choosing a model with an optimal input image resolution of 416x416 pixels is advisable for mobile platforms.

5.2. Performance evaluation of the quantized CNN models on the mobile platform

The key metrics discussed in this study include **FPS**, **BFLOPs**, time to display results on the screen after processing begins ($\boldsymbol{T_{frame}}$), median time to perform an object prediction operation ($\boldsymbol{T_{predict}}$), median time to load the model onto the tested device ($\boldsymbol{T_{load}}$), model weight size (**w**), quantization level of the CNN models (**q**), hardware resources, processor types, and software tools. Additionally, **mAP** was used to evaluate the effectiveness of CNN on different mobile platforms.

To evaluate the performance of the developed CNN YOLO models on mobile platforms in real-time video imaging, it is appropriate to compare the available software and hardware resources according to the defined parameters and evaluation metrics. For testing, the following were selected:

- CNN models with two output layers (tiny models) and three output layers (regular models), with some models converted to CoreML format for iOS mobile devices;
- Input image resolution fixed at 416x416 pixels for all CNN models and devices on which they were tested;
- Hardware resources: For iOS, the test used the CPU, GPU, and NPU acceleration chips (depending on the selected floating-point computation precision). For the embedded Jetson Nano system, the CPU and GPU acceleration chip were used;
- The overall values of $\boldsymbol{T_{predict}}$ and $\boldsymbol{T_{load}}$ metrics were determined considering the use of all possible acceleration means (CPU/GPU/NPU);
- Software tools: the CoreML framework for iOS MOS and the OpenCV library for the Jetson Nano embedded system running on the Ubuntu operating system;
- Quantization levels (**q**) of the CNN model's weight coefficients for iOS MOS:
  - 16 bits: optimal for object recognition tasks using CPU and NPU;
  - 8 bits using affine transformations;
  - 4 bits using a lookup table created with the k-means clustering method;
  - 32 bits: double precision calculations for increased model performance using more extensive hardware resources, employing both CPU and GPU;
- The standard precision of 16 bits was applied for the embedded Jetson Nano system.

Table 3 and Table 4 present the performance results of the developed CNN models, verified in Ph.D. research (Kushnir, 2023)**.**

Table 3. Performance Metrics of Two-Layer CNN Models Depending on the Quantization Level and Model Type.

	Metric / **CNN Model**	**FPS** **(frames/s)**	**BFLOP's** **(billion op.)**	**w** **(MB)**	T_{frame} (s)	$T_{predict}$ (s)	T_{load} (s)	**mAP** **(%)**
	Yolov4_retr_tiny_coreml	8.4	6.454	**23.2**	0.02	**0.111**	**0.358**	86.9
q = 32	**Yolov4 _tiny_coreml**	**9**	**8.76**	24.5	0.02	0.134	0.42	87.1
	Yolov4_retr_tiny_coreml	**30.2**	7.34	**12**	0.03	**0.042**	**0.183**	82.1
	Yolov4_retr_tiny_nano	19.1	7.84	24	0.02	0.123	0.67	86.92
q = 16	Yolov4 _tiny_coreml	30.1	8.21	12.3	0.03	0.43	0.212	86.2
	Yolov4 _tiny_nano	18.0	**8.44**	24	0.02	0.234	0.69	87.21
	Yolov4_retr_tiny_coreml	**33.3**	4.22	**6.1**	0.02	**0.067**	0.434	82.1
q = 8	**Yolov4 _tiny_coreml**	33	**4.94**	7.9	0.02	0.074	**0.383**	81.7
	Yolov4_retr_tiny_coreml	32	**2.44**	**3.2**	0.03	0.08	0.46	68.1
q = 4	Yolov4 _tiny_coreml	32	2.56	3.3	0.04	0.08	0.41	61.2

The test results should be analyzed based on the metric values. As seen in Table 3

and Table 4, depending on the quantization level (**q**) increases, the **FPS** value also increases. However, the object recognition efficiency metrics, such as **mAP** and $\boldsymbol{T_{predict}}$, decrease proportionally. When quantization is reduced to 4 bits, recognition quality drops sharply.

The values of **BFLOPs**, **w,** and $\boldsymbol{T_{load}}$ decrease linearly depending on the increase in quantization level and changes in the number of NNM output layers. In most tests, the proposed CNN model shows improved results compared to its direct analogs, with an average improvement of 5-10% across most metrics.

Table 4. Performance Metrics of Three-Layer CNN Models Depending on the Quantization Level and Model Type.

	Metric	**FPS**	**BFLOP's**	**w**	$\boldsymbol{T_{frame}}$	$\boldsymbol{T_{predict}}$	$\boldsymbol{T_{load}}$	**mAP**
	CNN Model	**(frames/s)**	**(billion op.)**	**(MB)**	(s)	(s)	(s)	**(%)**
	Yolov4_retr_coreml	5.1	49.2	257	4.6	0.427	2.6	98.1
q = 32	**Yolov4 _coreml**	5.2	**52.8**	258.5	4.2	0.428	2.5	97.9
	Yolov4_retr_coreml	**6.3**	**45.1**	**129**	3.7	**0.32**	3.55	97.3
	Yolov4_retr_nano	3.2	42.1	256	**3.2**	0.39	**2.44**	97.2
q = 16	**Yolov4 _coreml**	**6.4**	**46.2**	**129.7**	3.6	**0.34**	3.63	94.8
	Yolov4 _nano	3.3	42.3	257	**3.3**	0.4	**2.37**	94.2
	Yolov4_retr_coreml	8.1	**29.1**	64.2	2.1	**0.101**	**3.21**	82.1
q = 8	Yolov4 _coreml	8.0	28.2	65.2	2.32	0.14	3.39	85.5
	Yolov4_retr_coreml	**13.7**	18.3	33.3	1.35	0.081	**2.12**	73.1
q = 4	Yolov4 _coreml	13.4	18.1	34.1	1.43	0.083	2.43	54.2
	Metric	**FPS**	**BFLOP's**	**w**	$\boldsymbol{T_{frame}}$	$\boldsymbol{T_{predict}}$	$\boldsymbol{T_{load}}$	**mAP**
	CNN Model	**(frames/s)**	**(billion op.)**	**(MB)**	(s)	(s)	(s)	**(%)**
	Yolov4_retr_coreml	5.1	49.2	257	4.6	0.427	2.6	98.1
q = 32	**Yolov4 _coreml**	5.2	**52.8**	258.5	4.2	0.428	2.5	97.9
	Yolov4_retr_coreml	**6.3**	**45.1**	**129**	3.7	**0.32**	3.55	97.3
	Yolov4_retr_nano	3.2	42.1	256	**3.2**	0.39	**2.44**	97.2
q = 16	**Yolov4 _coreml**	**6.4**	**46.2**	**129.7**	3.6	**0.34**	3.63	94.8
	Yolov4 _nano	3.3	42.3	257	**3.3**	0.4	**2.37**	94.2
	Yolov4_retr_coreml	8.1	**29.1**	64.2	2.1	**0.101**	**3.21**	82.1
q = 8	Yolov4 _coreml	8.0	28.2	65.2	2.32	0.14	3.39	85.5
	Yolov4_retr_coreml	**13.7**	18.3	33.3	1.35	0.081	**2.12**	73.1
q = 4	Yolov4 _coreml	13.4	18.1	34.1	1.43	0.083	2.43	54.2

When comparing the performance of the CNN model on iOS MOS using the iPhone 12 hardware versus the embedded Jetson Nano device (at a 16-bit quantization level), iOS MOS has a significant advantage. This advantage is achieved through the successful combination of system processors (**NPU** and **GPU**) when solving object recognition tasks. In contrast, a roughly equivalent Jeston ARM Nvidia processor cannot provide a sufficient number of **BFLOPs**.

To verify this hypothesis regarding the use of hardware resources, a test was conducted on iOS mobile device using limited hardware resources for the $T_{predict}$ and T_{load} metrics (Figure 7 and Figure 7).

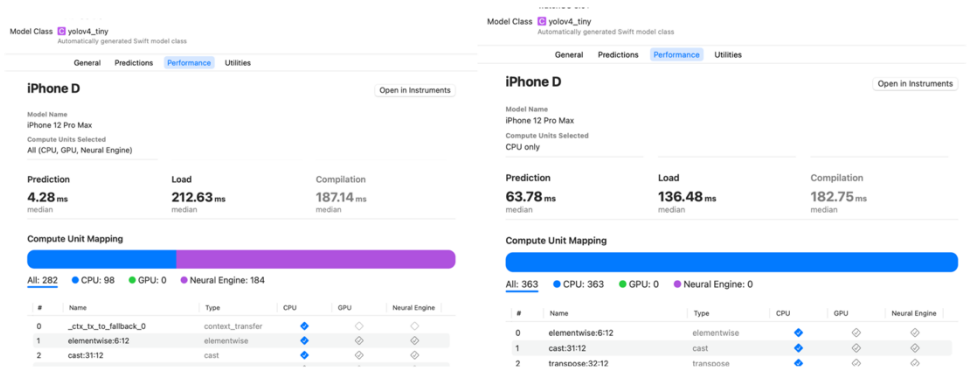

Figure 7. Usage of CPU and NPU (left) vs. CPU only (right) during testing the two-layer YOLO CNN model.

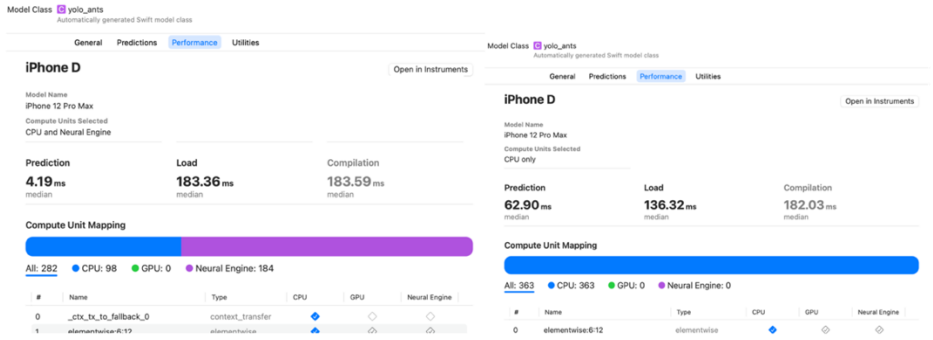

Figure 8. Usage of CPU and NPU (left) vs. CPU only (right) during testing the two-layer RETR YOLO CNN model.

The results indicate that using only the **CPU** for object recognition tasks on mobile platforms reduces the prediction (recognition) and data processing performance by 12 times. Additionally, the model loading time increases by 64%, which does not significantly impact development efficiency.

Meanwhile, using 32-bit quantization activates the **GPU** acceleration chip instead of the **NPU**, reducing the model's performance by 70%. Thus, the performance of the CNN model on iOS, when using both **CPU** and **GPU**, is approximately equivalent to the metrics of the embedded Jetson Nano OS (with a quantization level of 16 bits, $T_{predict}$ is 0.111 for iOS and 0.123 for the embedded Jetson Nano system).

The **FPS** for most tiny models exceeds the threshold value of 24 frames per second, making the developed models suitable for real-time tasks.

5.3. Model evaluation in the indoor environment with ants

The proposed methodology and frameworks were tested on an indoor ant colony of Camponotus fellah ants as they moved through the nest from one entrance to another (Figure 9). The ex situ recording setup featured a static tripod for stability, a macro lens (Apexel APL-HB2XT) with a 60mm focal length for capturing detailed images of the ants, and LED lighting to ensure optimal visibility of their movements.

Figure 9. Recognition and tracking of Camponotus fellah ants during movement. Each recognized ant is assigned a unique identifier (e.g., recognized rectangle #311 for static ant), allowing for real-time tracking of their movements using a specific color.

For this research, images and video footage were captured using the rear wide camera of an iPhone 12. The device's 12-megapixel resolution optical image stabilization camera can also handle 30FPS for CNN processing. While the camera performed well in controlled settings, challenges like ants on vertical surfaces or blurred frames affected recognition performance. Expanding the dataset to include images under these conditions could improve model robustness.

The numbers assigned near each ant indicate a unique identifier generated by the V-IOU algorithm during tracking. As previously noted, to minimize overlap between recognized bounding boxes, the SCF was set to 0.9, and the IOU threshold was configured at 0.2. These parameters ensured precise object tracking while reducing the likelihood of redundant or overlapping detections.

The injected V-IOU algorithm was applied using the specified JavaScriptCore methods for tracking. The path is defined by displaying the centroids of each unique object in distinct colors, with the option to retain this information for a specified duration. Each unique object within a particular class is identified by its color.

This information can then be applied to an analytics system for counting recognized

and tracked objects, like ants crossing specific entrances in the nest by crossing a boundary line. By leveraging the tracking algorithm's ability to distinguish individual objects, the system could provide detailed insights into movement patterns and colony dynamics.

6. Discussion

The analysis results indicated that using the k-means++ clustering method significantly improved object recognition efficiency, with an increase of 5% in mAP, 5-6% in F1, and 3-4% in R. Considering the available hardware capabilities, a model with two output layers and a resolution of 416x416 pixels was determined to be optimal for the NNM model's performance on mobile platforms.

Performance metrics analysis of the developed CNN model suggested that the optimal quantization level is 16 bits for the 2-layer model and 8 bits for the 3-layer model. The FPS for most models remained around 24 FPS, which is sufficient for real-time tasks.

Models on the iOS platform, converted to CoreML format, demonstrated the highest performance, as this mobile operating system effectively leverages the system's processors (NNA and GPU) for object recognition tasks. In contrast, the Jeston ARM Nvidia processor could not provide sufficient BFLOPS despite having similar characteristics. The research revealed that using NPU and CPU chips increased prediction (recognition) and data processing performance ($T_{predict}$) 12 times compared to using only the CPU on iOS mobile devices.

In most performance tests, the proposed CNN RETR YOLO model showed improved results compared to its direct analogs, achieving an average of 5-10% improvement across most metrics.

Overall, this CNN model for mobile devices can effectively recognize and track small, fast-moving objects in real time.

Future work may explore migrating the tracking algorithm to a native Swift implementation to further optimize performance and fully leverage hardware-specific accelerations available on iOS devices.

The system's compatibility with any iOS device offers significant potential for scalable and accessible monitoring solutions for possible in situ applications. Future deployment in natural habitats could involve additional hardware, such as a stabilizer and retainer for consistent image capture and a portable power source to support extended operation. These enhancements, combined with the framework's portability and ease of integration, suggest that it could be effectively adapted for efficient field applications.

7. Conclusions

The study demonstrated that optimizing the CNN model with the k-means++ clustering method and specific quantization levels significantly improves object recognition and tracking on mobile platforms. Converting models to CoreML for iOS proved remarkably effective, leveraging NPU and GPU chips to enhance performance. The research also successfully applied these methods to the recognition and tracking

of ants, illustrating the model's capability to handle small, dynamic objects in real time. These findings contribute to developing advanced mobile applications for recognizing and tracking small, fast-moving objects, paving the way for further advancements in mobile-based recognition systems.

Acknowledgments

Ukraine's Ministry of Education and Science supported part of the study through the "Intelligent Design Methods and Tools for the Modular Autonomous Cyber-Physical Systems" project (registration #0119U100609).

References

Arthur, D., Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (pp. 1027-1035). Society for Industrial and Applied Mathematics. https://doi.org/10.5555/1283383.1283494

Bochinski, E., Senst, T., Sikora, T. (2018). Extending IOU based multi-object tracking by visual information. In *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)* (pp. 1-6). Auckland, New Zealand. https://doi.org/10.1109/AVSS.2018.8639144

Bochkovskiy, A., Redmon, J., Sinigardi, S., Hager, T., Jaled M.C. et al. (2021). AlexeyAB/darknet (version yolov4). Zenodo. https://doi.org/10.5281/zenodo.562267

Gudauskas, J., Petkutė, G., Trakšelis, K., Kriščiūnas, A. (2024). UAV-based traffic intensity analysis framework: A case study on pedestrian crossings. *Baltic Journal of Modern Computing*, **12**(1), 102-115. https://doi.org/10.22364/bjmc.2024.12.1.07

Kushnir, D. (2022). Methods and means for small dynamic objects recognition and tracking. Computers, Materials & Continua, **73(**1), 1933-1949. https://doi.org/10.32604/cmc.2022.030016

Kushnir, D. (2022). Ants dataset (indoor/outdoor Messor Structor) + trained YOLOv4 weights. Mendeley Data. https://doi.org/10.17632/zprk7wkf9j.1

Kushnir, D. (2023). Methods and means of searching and recognizing objects in video images on the mobile platform in real-time, PhD thesis, Lviv Polytechnic National University, Lviv, Ukraine. https://lpnu.ua/sites/default/files/2023/radaphd/23565/diskushnir.pdf

Li, R., Wang, Y., Liang, F., Qin, H., Yan, J., Fan, R. (2019). Fully quantized network for object detection. *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 2810-2819).

Marques, O. (2020). Machine learning with Core ML. In *Image Processing and Computer Vision in iOS* (pp. 29-40). Springer. https://doi.org/10.1007/978-3-030-54032-6_4

Novák, M. (2020). Secure JavaScript UI rendering for iOS using Swift.

Popp, S., Dornhaus, A. (2024). Collective search in ants: Movement determines footprints, and footprints influence movement. *PLOS ONE*, **19**(4), e0299432. https://doi.org/10.1371/journal.pone.0299432

Redmon, J., Divvala, S. K., Girshick, R. B., Farhadi, A. (2015). You only look once: Unified, real-time object detection. *CoRR*, 779 - 788.

Smith, R., Patel, K. (2023). IntelliBeeHive: Real-time monitoring of honeybee activity using machine learning. arXiv Preprint. https://doi.org/10.48550/arXiv.2309.08955

Tkachenko, M., Malyuk, M., Holmanyuk, A., Liubimov, N. (2020). Label Studio: Data labeling software. https://github.com/heartexlabs/label-studio

Ulrich, J., Arvin, F., Rojas, N. et al. (2024). Autonomous tracking of honey bee behaviors over long-term periods with cooperating robots. *Science Robotics*, **9**(47), eadn6848. https://doi.org/10.1126/scirobotics.adn6848

Wang, Y., Guan, Y., Liu, H., Jin, L., Li, X., Guo, B., Zhang, Z. (2023). VV-YOLO: A vehicle view object detection model based on improved YOLOv4. *Sensors*, **23**(7), 3385. https://doi.org/10.3390/s23073385

Received August 14, 2024, revised January 23, 2025, accepted January 25, 2025