# Investigation of Deep Learning Methods for Bee Behavior Recognition

**Name:** Ozan Berk Gultegin

**Supervisor:** Tomyslav Sledevič

Contents

[Investigation of Deep Learning Methods for Bee Behavior Recognition 1](#_Toc198133518)

[Abstract 1](#_Toc198133519)

[Introduction 1](#_Toc198133520)

[Methodology – Pipeline 2](#_Toc198133521)

[Data Collection 2](#_Toc198133522)

[Dataset 2](#_Toc198133523)

[Handling the Dataset with Python 3](#_Toc198133524)

[Setting up a Model for Training 5](#_Toc198133525)

[Colab Specific Export 6](#_Toc198133526)

[Cropping Bee Images 6](#_Toc198133527)

[Feature Extraction 7](#_Toc198133528)

[Unsupervised Clustering 8](#_Toc198133529)

[Conclusion for Current Pipeline 11](#_Toc198133530)

## Abstract

The health and productivity of bee colonies can be assessed through the analysis of behavior patterns exhibited by bees at the hive entrance. This study investigates the use of deep learning methods, specifically Vision Transformers (ViT) and YOLOv11, to automate the recognition and classification of bee behaviors in video footage from multiple beehives. The behaviors targeted for identification include fanning, defense, foraging, and liquid transfer (trophallaxis), which serve as critical indicators of colony activity, environmental stressors, and overall hive health.

## Introduction

Honeybee colonies are essential for pollination and ecological balance, making their health crucial for agriculture and biodiversity. Monitoring bee behavior at hive entrances offers valuable, non-invasive insights into colony dynamics, as these behaviors often reflect internal hive conditions and environmental interactions.

Visual observation can help beekeepers detect issues such as disease, resource shortages, and stress. Key entrance behaviors include:

* **Fanning**: Bees rapidly vibrate their wings to regulate temperature and humidity, often in coordination with water droplets. Fanning also helps distribute pheromones for hive signaling and coordination.
* **Defense**: Guard bees protect the hive by displaying aggression, releasing alarm pheromones, and blocking the entrance from threats like predators or intruders.
* **Foraging**: Foragers collect nectar, pollen, water, and propolis. Their return to the hive, often with pollen loads, indicates resource availability and overall colony productivity.
* **Trophallaxis**: Bees exchange liquid food mouth-to-mouth, distributing nutrients and pheromones. This behavior also supports hive communication and social cohesion.

Understanding these behaviors is critical for assessing colony health and efficiency. This study investigates the use of deep learning, specifically Vision Transformers (ViT) and YOLO-based models, for automatic recognition of bee behaviors at hive entrances. The goal is to build a foundation for automated, scalable hive monitoring systems that support sustainable beekeeping practices.

## Methodology – Pipeline

### Data Collection

The video data used in this study was provided by the thesis supervisor and originates from a fixed-camera setup observing a hive entrance. All these recordings are under consistent environmental conditions, clear and sunny days during the same season. Each video is recorded at a resolution of 1920×1080 pixels and a frame rate of 30 frames per second (FPS).

### Dataset

The raw data comprises approximately 4 to 6 hours of MP4-format video, with each file representing a different observation window during the day. Some segments of these videos were already labeled when received, but a significant portion of the data required manual annotation. For those, I manually labeled bees frame by frame using bounding boxes, preparing the dataset for deep learning workflows involving object detection.

To convert video into image data suitable for training, I extracted individual frames from the MP4 videos at the full frame rate (30 FPS), resulting in approximately ~18,000 JPEG images.

The images were uploaded to Roboflow, a web-based dataset management and annotation tool. In Roboflow, I created bounding box annotations for each bee in the scene, assigning all instances the single class "bee" since behavior labels were not yet available at this stage. Once annotated, the dataset was exported in YOLO format, which includes one .txt file per image. Each file contains normalized bounding box coordinates and class IDs, structured as:

**Example label:**

0 0.23671875 0.6234375 0.03515625 0.08046875

‘class\_id x\_center y\_center width height’

This format is compatible with the YOLOv11 training pipeline used later.

### Handling the Dataset with Python

The annotated dataset is retrieved programmatically using a custom Python script that utilizes Roboflow’s API. This setup allows easy access to specific dataset versions I’ve created and simplifies experimentation across different machines and environments. Due to local hardware limitations, particularly storage constraints, this automation has been especially valuable, letting me work with large datasets without the need to store them locally.

**What Roboflow is used for:**

Roboflow is a cloud-based tool that supports dataset annotation, preprocessing, augmentation, and model management. In this project, Roboflow is primarily used to:

* Annotate images by drawing bounding boxes around bees.
* Apply preprocessing and augmentation techniques (e.g., resizing, flipping, rotation).
* Manage versions of datasets, including train/val/test splits.
* Host datasets in the cloud, reducing dependency on local storage.
* Deploy pre-trained models or run inferences in the cloud.

This tool has significantly simplifed the data preparation workflow. In future stages of the project, Roboflow’s preprocessing and augmentation capabilities will be leveraged more heavily to scale the dataset, potentially reaching over a million high-quality training samples. Additionally, centralized cloud storage will help avoid local system bottlenecks during training and experimentation.

**Pipeline Overview**

The Python script used for dataset retrieval performs the following steps:

* Authentication with the Roboflow API using a personal token.
* Downloading a specific version of the dataset.
* Organize the downloaded files into a structure compatible with the training pipeline.
* Since Roboflow handles dataset splitting into training, validation, and testing subsets, no further manual splitting is required.

**Code for Roboflow Dataset Retrievel:**

%pip install roboflow

from roboflow import Roboflow

rf = Roboflow(api\_key="API-KEY")

project = rf.workspace("WorkSpaceName").project("Project Name")

version = project.version(1) # Dataset version

dataset = version.download("yolov11")

Once the dataset has been downloaded, the next step is to reorganize and format the folder structure to meet YOLOv11’s training requirements. The script that handles this transformation is shown below.

**Dataset Organization for YOLOv11:**

After downloading the dataset from Roboflow, the next task was to reorganize the folder structure to align with YOLOv11's training requirements. Roboflow exports datasets with subdirectories for, train, test, and valid, but YOLOv11 expects a specific structure with clearly separated images and labels directories, each containing subfolders for these splits.

The following Python script automates the reorganization process:

*A screen shot of a computer program

AI-generated content may be incorrect.*

Once the folders are reorganized appropriately and the valid directory is renamed to val, the dataset becomes compatible with YOLOv11’s expected directory format.

### Setting up a Model for Training

Once the dataset is downloaded and reorganized appropriately, the next step is setting up the YOLOv11 model for training. To prepare the environment, a few adjustments were necessary to avoid GPU memory issues, especially when running on machines with limited resources such as Google Colab or older GPUs.

The training environment is configured with the following commands:

**torch.cude.empty\_cache()**: This command is used to free up GPU memory in case of fragmentation or residual allocations before beginning the training session.

**%env PYTORCH\_CUDA\_ALLOC\_CONF=expandable\_segments:True:** This environment variable configures PyTorch's CUDA memory allocator to be more flexible. While not always reliable, it helps reduce memory allocation errors during training.

Once the environment is prepared, the YOLOv11 model is initialized with a selected variant:

**model = YOLO('yolo11m.pt')**

YOLOv11 provides several model variants of increasing size and complexity:

* **yolo11n.pt – Nano**
* **yolo11s.pt – Small**
* **yolo11m.pt – Medium**
* **yolo11l.pt – Large**
* **yolo11x.pt – Extra Large**

Each of these models can be trained independently and later compared to evaluating their trade-offs between speed, accuracy, and resource usage. In this initial setup, the medium model (**yolo11m.pt**) was selected for experimentation.

The actual training is initiated with the following configuration:

A computer screen shot of a computer program

AI-generated content may be incorrect.

These settings represent a baseline training configuration and will be subject to optimization in future iterations. Hyperparameter tuning across multiple configurations, potentially inspired by automated methods like those used in the FastAI library, remains a future goal.

### Colab Specific Export

As mentioned previously, due to limitations in local hardware resources, a significant portion of training and experimentation was conducted on Google Colab. While Colab provides free access to powerful GPUs, managing the output, particularly downloading the trained model and related training statistics, can be cumbersome if done manually.

To streamline this process, a small custom script was implemented. This script ensures that trained models and related assets such as evaluation results, bounding box visualizations, and logs are automatically compressed and made available for download after training completes.

The snippet below is specifically designed for use in the Colab environment and should be avoided when working on a local machine:

A computer screen shot of code

AI-generated content may be incorrect.

Once these steps are completed, I obtain a **.zip** archive containing:

* The trained YOLOv11 model weights
* Performance metrics and training logs
* Validation images annotated with predicted bounding boxes and confidence scores

This archive can be easily transferred to a local machine or stored in the cloud for later use, comparison or fine-tuning.

### Cropping Bee Images

This section marks the midpoint of the thesis project, where the object detection task has been successfully completed, and a trained YOLOv11 model is available for downstream tasks. This stage also marks the beginning of the most challenging part of the thesis, interpreting bee behavior without labeled action data.

As mentioned earlier, there is currently no labeled information for the bees' behaviors (such as foraging, fanning, or defense). Therefore, the next steps involve designing an approach to extract meaningful representations from the detected bees in a completely unsupervised manner. The goal is to develop a clustering pipeline that can later support behavior classification.

The first step in this direction is to crop individual bee instances from the full-frame images, using the detection model’s output. Each bee is only retained if the confidence score of the detection is greater than 50%. The following Python snippet demonstrates how each detected bounding box is cropped from the original image and saved separately:

A computer screen with colorful text

AI-generated content may be incorrect.

Here is an example output image cropped from a video frame:

A blurry image of a zebra

AI-generated content may be incorrect.cropped\_bee\_20230609a48\_0.jpg

Once this cropping step is completed, I am left with thousands of individual bee images. These cropped instances form the new dataset on which feature extraction and unsupervised learning will be applied.

### Feature Extraction

Once individual bees were cropped from the original frames, the next step was to generate numerical representations of each image. For this purpose, a pre-trained convolutional neural network model, ResNet50, was used as a feature extractor. The final classification layers of ResNet50 were removed (**include\_top=False**), and **global average pooling** was applied to produce a compact 2048-dimensional feature vector for each image.

The feature extraction process for each image is illustrated in the following code snippet:

A screenshot of a computer program

AI-generated content may be incorrect.

Each image is now represented by a fixed-length feature vector. These vectors form the input for the next step, clustering bees into visually similar groups using unsupervised techniques.

### Unsupervised Clustering

With each cropped bee image now represented as a high-dimensional feature vector, the next step was to explore potential patterns in the dataset by grouping similar instances together using unsupervised learning. This step is important because, in the absence of behavior labels, clustering provides an initial idea of how bees might naturally group based on their appearance.

**Dimensionality Reduction with PCA**

The feature vectors extracted from ResNet50 are 2048-dimensional. To make the clustering process more interpretable and visually accessible, Principal Component Analysis (PCA) was applied to reduce the feature space to just two dimensions. This reduction allows for a visual inspection of how the bees are distributed in feature space and whether any natural clusters emerge.

**Clustering with KMeans**

Following PCA, the reduced feature vectors were clustered using KMeans with k=4 clusters. This number was chosen heuristically to observe general grouping trends.

A computer screen shot of a program

AI-generated content may be incorrect.

This visualization gives a first look into potential behavioral clusters, even though the exact meaning of each cluster remains unknown. Future steps will involve examining the clustered images to attempt manual labeling or develop weak supervision rules.

Resulting clusters:

A diagram of a colorful chart

AI-generated content may be incorrect.

Despite the reduction, clear separability between clusters is observable, suggesting that there are distinguishable patterns among the bee instances, possibly reflecting differences in pose, orientation, lighting, or behavior-related cues.

**Manifold Learning**

To further investigate the structure of the feature space beyond linear PCA, manifold learning techniques were also applied. These techniques attempt to preserve local relationships between data points, often revealing more nuanced or curved structures in the data that PCA might miss.

**t-SNE:**

First, t-distributed Stochastic Neighbor Embedding (t-SNE) was applied to the feature vectors. t-SNE focuses on maintaining local similarities and is widely used for visualizing high-dimensional data in a 2D space. Below is the result of t-SNE with the same cluster assignments obtained from KMeans:

A computer code on a black background

AI-generated content may be incorrect.

A diagram of a bee feature

AI-generated content may be incorrect.

### Conclusion for Current Pipeline

In summary, the current pipeline begins by extracting frames from the video recordings. Using Roboflow, the images are labeled manually, followed by preprocessing and augmentation. The labeled dataset is then split randomly into training, validation, and test sets. This versioned dataset is retrieved using custom Python scripts and formatted appropriately to match YOLOv11's input requirements.

Once the YOLOv11 model is trained, it is used to detect bees in the video frames. Detections with confidence scores above 50% are retained, and the corresponding bee regions are cropped and saved as individual images. These cropped bee instances form the input for the feature extraction stage, where a pretrained CNN (e.g., ResNet50) is used to encode visual information. The extracted features are then analyzed using unsupervised learning techniques (such as PCA, KMeans, t-SNE, and UMAP) to identify meaningful patterns or clusters that could correlate with specific behaviors.

However, this pipeline will evolve significantly in the next phase of the thesis. For example:

* The YOLOv11 model will be integrated directly into a video-based processing pipeline, allowing for better temporal consistency and improved bee tracking across frames.
* Improved tracking is crucial, as the current method lacks robustness in following individual bees consistently.
* Multiple dataset versions and model variations will be explored, using optimized hyperparameters, to achieve the best possible performance in detection.
* Once detection results are sufficiently accurate, they will serve as input to a Vision Transformer (ViT) model for action recognition.
* Collaborations with my supervisor or a melittologist (bee specialist) will also help annotate specific behaviors (e.g., trophallaxis, fanning, foraging), enabling supervised learning for behavioral classification.

This expanded and refined pipeline aims to transform raw visual data into structured behavioral insights, ultimately contributing meaningful results to the field of bee behavior analysis.