Fachhochschule Kärnten
Carinthia University of Applied Sciences

KÄRNTEN
University of
Applied Sciences

MASTER THESIS

A Comparison of Convolutional Neural Network Architectures for Insect Detection of Different Sizes

A thesis submitted for the degree of

Master of Science (M.Sc.)

Author:	Saghar Ghaffari, B.Sc.
Registration number:	2210876015
Study program:	Applied Data Science
Department:	Engineering & IT
First Supervisor:	FH-PROF. DR.-ING. KARL-HEINRICH ANDERS Department of Geoinformation and Environmental Technologies, Fachhochschule Kärnten Europastraße 4 9524 Villach
Second Supervisor:	FH-PROF. DIPL.-ING. DR.TECHN. ADRIJANA CAR Department of Geoinformation and Environmental Technologies, Fachhochschule Kärnten Europastraße 4 9524 Villach

Villach, September 2024

Abstract

This study evaluates the performance of several CNN models, including YOLOv8, Faster R-CNN, and RetinaNet, for insect detection and classification at both the insect and species levels using high-resolution images. Two training approaches were applied, general insect-level detection and species-level detection into six insect groups. The results demonstrate that You Only Look Once (YOLO)v8 consistently outperformed other models, particularly in small insect detection, such as Chelicerata (spiders measuring 20 mm to 30 mm in length) and Diptera (flies measuring 12 mm to 45 mm in length), which were challenging for other models.

At the insect level, YOLOv8 achieved a Weighted (mAP@50:95) of 0.93177% and an F1 score of 0.994%, surpassing Faster R-CNN and RetinaNet. Faster R-CNN with a ResNeXt-101 backbone followed by a Weighted (mAP@50:95) of 0.873% and F1 score of 0.989%. At the species level, YOLOv8 Approach 2 achieved a Weighted (mAP@50:95) of 0.6990% and F1 score of 0.7310% reflecting its superior detection capabilities across all species. However, when tested on original, unprocessed images, the model performance dropped, with YOLOv8 Approach 2 achieving an overall mAP of 0.34524% on species-level test data.

Despite its high precision and resilience in detecting small species and handling various challenging conditions (such as blurry images and occlusions), YOLOv8 demonstrated higher computational costs compared to other models, making it less suitable for real-time applications in resource-limited environments. Meanwhile, Faster R-CNN and RetinaNet showed a better balance between precision and processing efficiency but struggled with smaller insects and noisy environments. These findings highlight YOLOv8 potential for use in ecological monitoring and pest management, particularly when accuracy in small object detection is paramount. Future work should focus on improving performance for underrepresented species and addressing the computational demands of these models.

Statutory declaration

I hereby declare that

- I have independently written the presented Master thesis by myself;
- I have prepared this Master thesis without outside help and without using any sources or aids other than those cited by me; moreover, I have identified as such any passages taken verbatim or in terms of content from the sources used;
- in addition, I have fully indicated the use of generative AI models (e.g. ChatGPT) by specifying the product name and the reference source (e.g. URL);
- I have not used any other unauthorized aids and have consistently worked independently and when using generative AI models, I realize that I am responsible how the content will be used and to what extent
- I have not yet submitted this Master thesis in the same or similar form to any (other) educational institution as an examination performance or (scientific) thesis.

I am aware that any violation ("use of unauthorized aids") violates academic integrety and may result in (academic-related) legal consequences.

8.9.2024 , Villach

Place and Date

Student's signature

Acknowledgements

Me Saghar Ghaffari

I would like to extend my heartfelt gratitude to everyone who has supported and guided me throughout the past two semesters. My deepest thanks go to my supervisors, **FH-PROF. DR.-ING. Karl-Heinrich Anders** and **FH-PROF. DIPL.-ING. DR. TECHN. Adrijana Car**, for their invaluable guidance, clear explanations, and encouragement during this project. I am also profoundly grateful to **Mohammad Mustafa Sa'doun** for his assistance and insight.

Additionally, I wish to express my sincerest appreciation to my family and friends, whose unwavering support and encouragement have been vital in my journey to achieve this endeavor. Your belief in me has been a constant source of motivation.

Contents

CHAPTER 1 Introduction

In this chapter, we explore the motivation behind utilizing Convolutional Neural Network (CNN)s for insect detection and the advantages they offer over traditional methods. We define the specific problem this study addresses, articulate our research questions, and outline the anticipated outcomes. Finally, we provide an overview of the thesis structure.

1.1 Motivation

Accurately detecting and classifying insects in their natural habitats have become increasingly crucial for ecological studies, pest management, and biodiversity conservation. Traditional methods for insect monitoring, which involve manual collection and identification, are not only time-consuming and labor-intensive but also invasive. These practices can disturb the natural habitats, harm the insects being studied, and sometimes lead to their death. This underscores the need for more sustainable and non-invasive monitoring techniques (Buehler et al. 2019). With the rise of Artificial Intelligence (AI) and Machine Leraning (ML), the use of image-based identification and recognition of insects has been significantly enhanced. CNNs have been particularly successful in this domain due to their ability to automatically learn features from images, making them ideal for various classification tasks (Chen et al. 2012; Howard et al. 2017; Girshick 2015).

Previous research by Schaffhauser (2021) at the Carinthia University of Applied Science (CUAS) has demonstrated the potential of CNNs for automatic identification of individual animals, such as monkeys, in controlled environments. However, these approaches often involve substantial manual effort for training and validation, which limits their scalability. It also highlights the feasibility of using CNNs for identifying individual monkeys, though the process was still heavily reliant on manual intervention for data annotation and model training.

The application of CNNs to insect detection and classification poses unique challenges due to the diversity and variability in insect appearances and behaviors. The use of wildlife cameras and other automated imaging systems can capture a large volume of data, but processing this data manually is impractical. Automated methods that can efficiently handle the training and adaptation of CNN models are essential to overcome these limitations (Pavlíček et al. 2018). Additionally, current CNN-based systems often struggle with the identification of new or previously unseen insect species, necessitating some level of supervised intervention to maintain precision (Schaffhauser 2021).

Recent advancements in computer vision and CNNs have shown significant improvements in detecting and classifying insects in complex environments. Studies such as those by Ren et al. (2016) and Redmon et al. (2016) have demonstrated the efficacy of advanced CNNs architectures like Faster Region-based Convolutional Neural Networks (R-CNN) and You Only Look Once (YOLO) in object detection tasks, including challenging scenarios with background noise and varying lighting conditions. These advancements are particularly relevant for our project, which involves capturing images of insects in their natural habitats using camera traps. These images often contain background elements such as flowers and leaves, which can complicate the detection process.

Moreover, the integration of Feature Pyramid Network (FPN)s and region-based CNNs has further improved the ability to detect small objects within larger scenes, making them suitable for insect detection in the wild (Lin, Dollar, et al. 2017). The use of such state-of-the-art techniques allows for the handling of high-resolution images and the extraction of fine-grained details necessary for accurate insect identification.

Our study compares various CNN architectures for detecting and classifying insects in their natural habitats. It leverages automated image-processing techniques such as data augmentation and Non-Maximum Suppression (NMS), which will be explained in detail in chapter 3, to improve the precision and robustness of insect monitoring. By evaluating various CNN models, we seek to identify the most effective approaches for handling the complexities of insect imagery, including varying environmental conditions and diverse insect appearances. The findings of this research will contribute to the development of more robust and scalable insect monitoring systems, facilitating better ecological studies and pest management strategies.

1.2 Research Goals and Research Questions

The main goal of this research is to comprehensively compare current state-of-the-art CNN architectures for insect detection and classification. Specifically, the study aims to evaluate the performance of different CNNs models, including YOLO, Faster R-CNN, and RetinaNet, in accurately identifying and categorizing insects in images. By experimenting with these models on locally collected data from camera traps in the Austrian Alps, we aim to understand their suitability for small object detection.

The research objectives include assessing the precision, robustness, and adaptability of these CNN architectures in the context of insect detection. The expected outcome is a thorough understanding of how different CNN models perform in identifying and classifying insects, contributing to the development of optimal approaches for automated insect identification.

To address these objectives, the research will focus on the following questions:

- Precision and robustness of CNN Models

  1.How do YOLO, Faster R-CNN, and RetinaNet perform in terms of precision when detecting and classifying insect species in images?

  2.What is the robustness of these CNN models in processing insect images from camera traps in natural habitats?

- Handling environmental and image quality challenges

  3.How well do YOLO, Faster R-CNN, and RetinaNet detect insects when there are multiple insects in one image?

  4.How does varying the image scales in the dataset impact the performance of the selected CNN models?

  6.How well do the selected CNN models handle scale sensitivity while detecting insects of varying sizes?

  5.How effective are chosen CNN models in detecting insects in images with partial occlusion caused by natural elements?

  6.How do YOLO, Faster R-CNN, and RetinaNet models perform when background pixels are not clear or when images contain significant noise?

By investigating these questions, this research aims to optimize targeted CNN architectures for improved insect detection and classification, ultimately enhancing computer vision for ecological studies and biodiversity conservation.

1.3 Methodology

The primary goal of this research is to use advanced CNN architectures for accurate detection of insect species at two levels: counting insects and species detection. This study will entail a comparative analysis of various CNN models, including YOLOv8, RetinaNet, and Faster R-CNN, leveraging a pre-existing dataset of 1303 high-resolution, labeled insect images obtained by the UNESCO CHAIR on Sustainable management of conservation areas at CUAS. These images, amassed from St.Margarethen im Lungau, Weinitzen, and Sittersdorf (Carinthia), encompass 3178 labels. The preprocessing procedures will involve image cropping, removal of redundant labels, and normalization of label formats. The overarching aim is to identify a CNN architecture that yields the highest mean Average Precision (mAP) for the discernment of different insect classes. Our experiment consists of three main parts. The process workflow is described in Figure 1.1. Each task will be broken down into smaller steps in advance to ensure a comprehensive approach.

- Conceptualization of research framework: In this stage, we aim to define the study's goals and scope and select CNN models (YOLOv8, RetinaNet, Faster R-CNN) for evaluation. We will also develop the research hypothesis, understand dataset characteristics, and determine key performance metrics such as mAP, see Figure 1.2.

- Implementation of the experiment: During this phase, we will implement and train the selected CNN models using the preprocessed insect dataset. This involves preparing the dataset by cropping images, normalizing label formats, and removing redundant labels. We will also develop a pipeline for model training, validation, and testing, and set up a system for logging and monitoring the training process to ensure optimal performance, see Figure 1.3.

- Results analysis and evaluation: In this final phase, we will conduct a detailed analysis of the model's performance, focusing on its ability to detect and classify insect species accurately. We'll use metrics like mAP, precision, recall, and F1-score to compare the robustness of each CNN model and address their strengths and weaknesses in handling diverse insect species of

different sizes. The study will conclude by identifying the best CNN model for the task and proposing potential improvements or future research directions, see Figure 1.4.

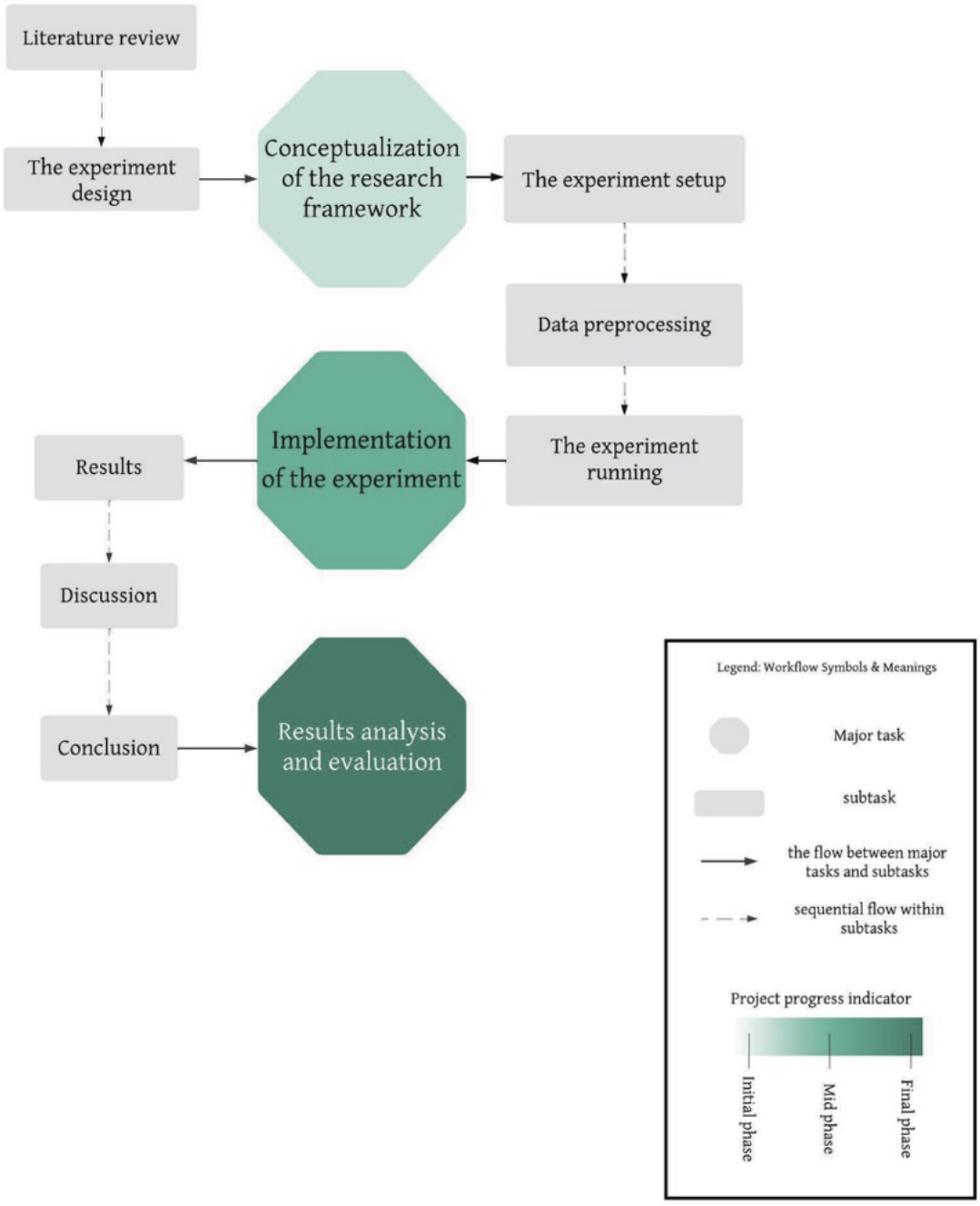

Figure 1.1: Project workflow of the master thesis

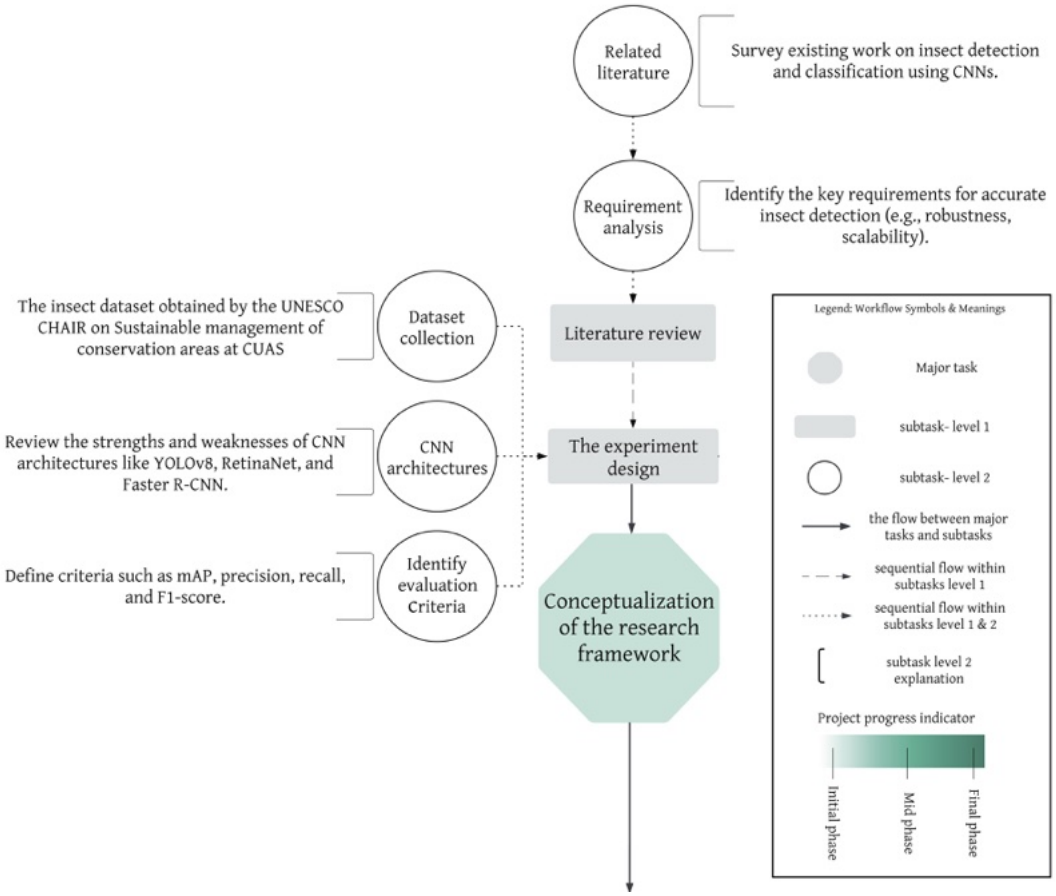

Figure 1.2: Workflow of conceptualization of the research framework

Experiment Design

The experiment design involves a thorough parameter selection based on an extensive literature review, followed by the training and evaluation of each CNN model (Krizhevsky, Sutskever, and Hinton 2012). We will assess model performance using metrics such as F1 score and mAP, focusing on the model's ability to handle challenges like partial occlusion, background noise, and varying insect scales. The training process involves feeding the CNN with the preprocessed dataset, iterating to minimize prediction errors. Performance comparison will reveal each model's strengths and limitations, providing insights for developing more efficient and accurate insect detection systems. This research builds on existing studies by Butera et al. (2021) and Li, Zhu, and Li (2021) and aims to advance ecological monitoring and pest management strategies through improved computer vision techniques.

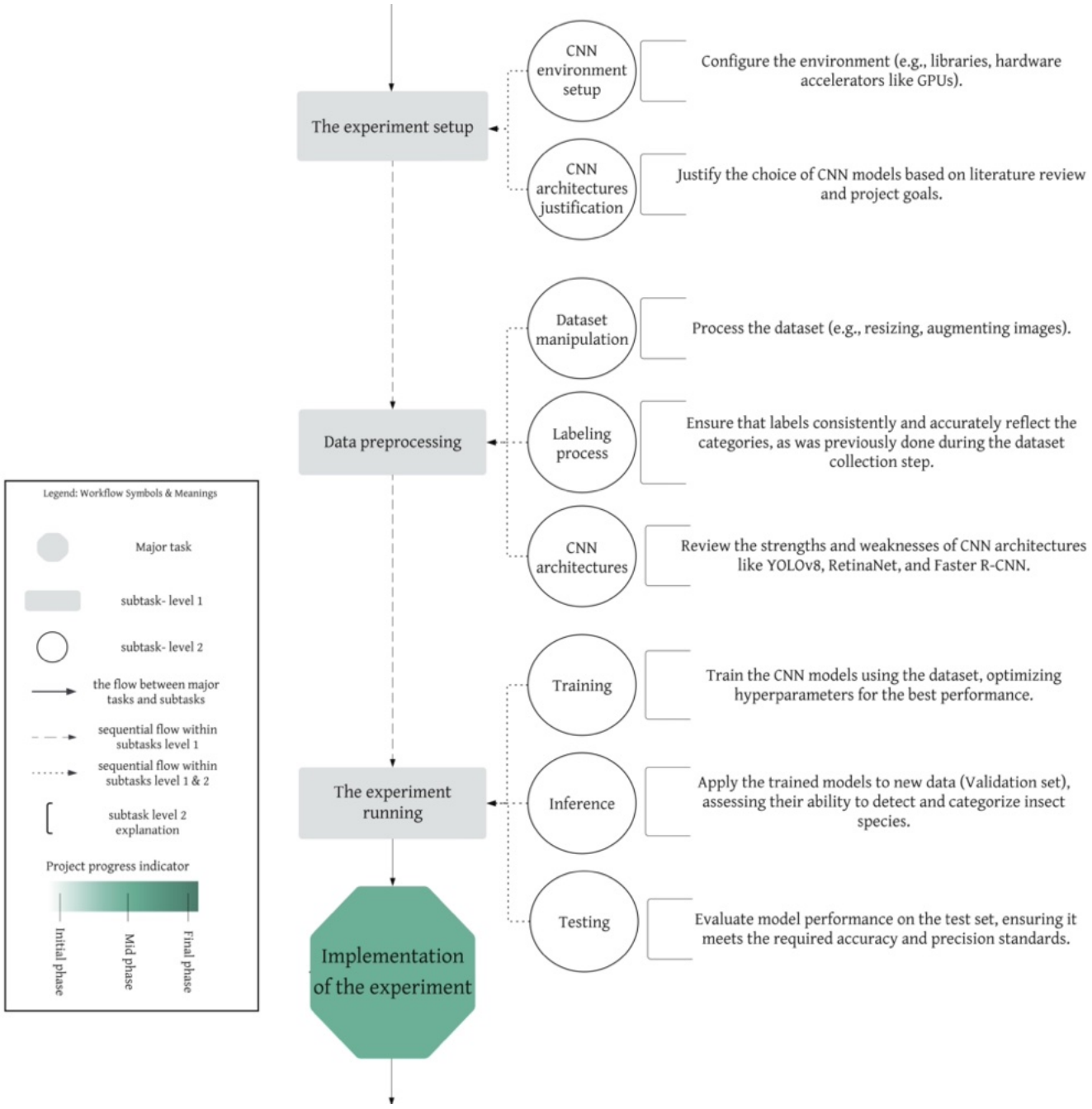

Figure 1.3: Workflow of implementation of the experiment

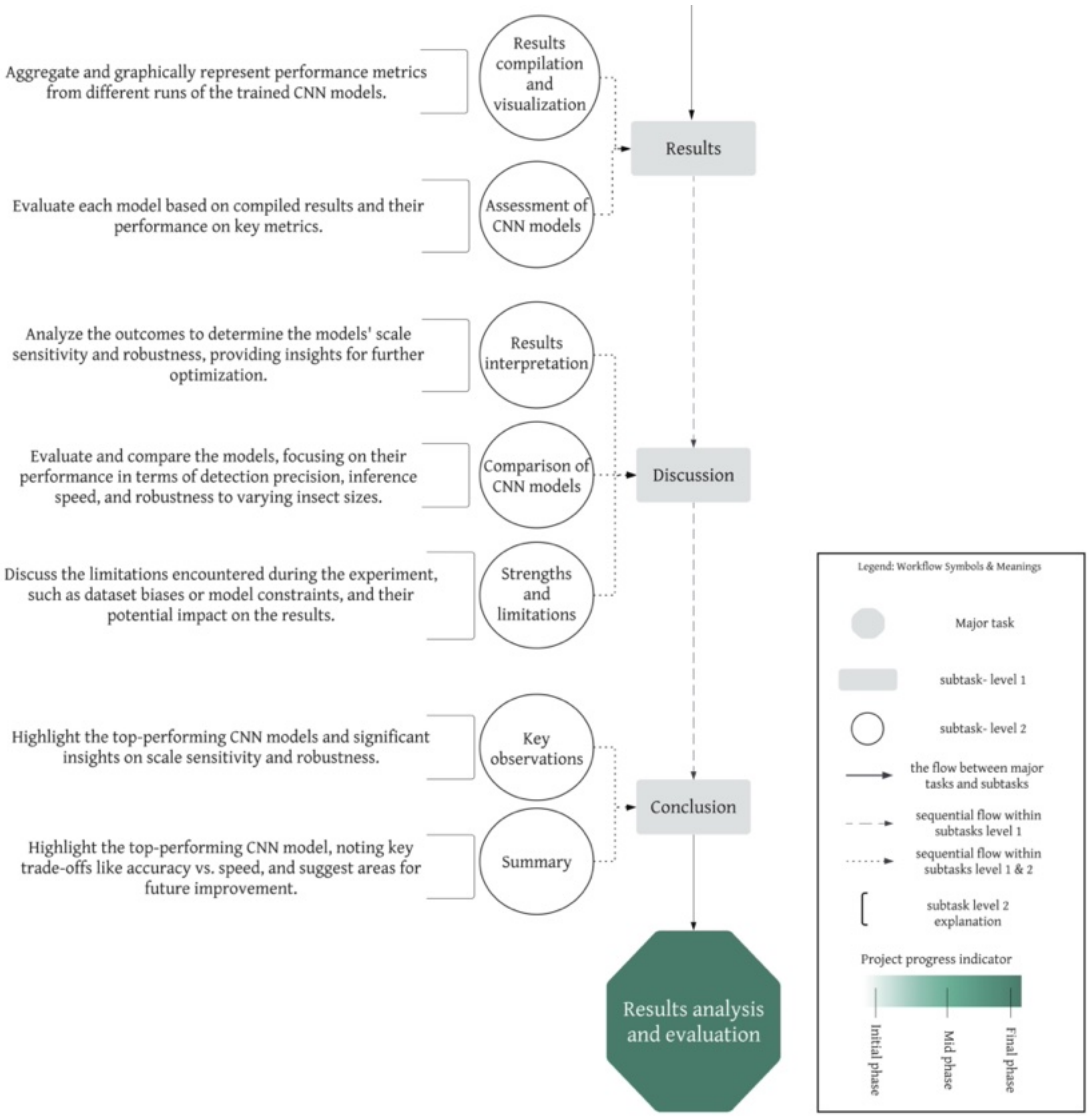

Figure 1.4: Workflow of result analysis and evaluation

1.4 Expected Results

The purpose of this research is to provide a comprehensive evaluation of various state-of-the-art CNN architectures, including YOLOv8, Faster R-CNN, and RetinaNet, particularly in the context of insect detection of different species. The study aims to offer insights into the effectiveness of these models in accurately identifying and counting insects, as well as detecting different species across various environmental conditions, with a focus on challenging scenarios such as small object detection, partial occlusion, and varying insect scales.

The assessment will be guided by a predetermined set of criteria, including precision, F1 score, robustness to environmental challenges, and scalability. The results

are expected to highlight the strengths and limitations of each model, providing a clear understanding of how well they meet these criteria.

In addition, the research aims to identify effective preprocessing techniques that enhance the precision and reliability of insect detection and classification. These findings are anticipated to significantly contribute to the optimization of CNN-based methods for insect surveying.

Furthermore, the study seeks to establish a robust implementation framework that can be applied to other insect datasets, ensuring the consistency of results across different contexts. This framework will serve as a valuable tool for non-invasive insect detection, adaptable to various environmental conditions and capable of addressing the identified challenges.

Ultimately, the findings aim to advance the field of ecological monitoring and pest management, showcasing the broader applicability of automated CNNs in wildlife monitoring and conservation. This research will provide a detailed assessment based on the predefined set of criteria of CNN models in this domain, offering practical recommendations for their use in real-world applications.

1.5 Intended Audience

This research is targeted towards professionals and academics in the fields of ecology, entomology, computer vision, and ML. Ecologists and entomologists can benefit from the automated insect detection and classification techniques developed in this study, as these methods provide a non-invasive and efficient way to monitor insect populations and biodiversity. This research is also relevant to conservation biologists and environmental scientists interested in sustainable management practices and the impacts of insects on ecosystems.

Computer vision and ML researchers will find this study valuable for its comparative analysis of different CNN architectures. This can inform future advancements in object detection and classification. The methodologies and findings presented will serve as a resource for developing more accurate and robust models for small object detection in diverse and uncontrolled environments.

This research holds significance for agricultural scientists and pest management professionals, offering innovative approaches to monitor and control pest populations without harming the insects or the environment. Policymakers and environmental agencies can also utilize the insights from this study to inform decisions

on biodiversity conservation and ecological management.

1.6 Thesis Structure

The thesis is structured as follows:

- Section 1: Introduction to the research problem, objectives, and the significance of the study.
- Section 2: Comprehensive literature review covering traditional insect monitoring methods, recent advancements in monitoring through camera traps, and the current state of machine-based surveying techniques.
- Section 3: Detailed methodology, including data acquisition processes, a thorough overview of the CNN architectures employed, and a step-by-step implementation workflow.
- Section 4: Implementation of CNN models, including the preprocessing steps applied to the dataset.
- Section 5: Presentation of results, both statistically and visually, to effectively illustrate the outcomes.
- Section 6: Discussion that compares the performance of various CNN architectures, interpreting the key findings in the context of the research objectives.
- Section 7: Conclusion summarizing the study's contributions, with suggestions for better future research directions.
- Section 8: Futureworks address possible suggestions for further studies.

CHAPTER

2 Literature Overview and Technical Background

To provide a comprehensive overview of the main topics covered in this research thesis, an extensive literature review has been conducted. This review explains the theory behind the techniques and methods used and is essential for understanding the research presented in the thesis. Initially, the review covers the basic knowledge of Artificial Neural Network (ANN), with a focus on CNN. It provides a detailed description of the key components of CNNs and their influence on CNN models performances, as well as the application of image classification in the context of insect detection, highlighting the relevance and importance of CNNs for this task. The review also introduces various technical tools and programs used in the project. To conclude, it showcases best practice examples of animal detection and recognition using CNNs, demonstrating established techniques and introducing alternative methods. Additionally, traditional surveying methods are reviewed, comparing their advantages and disadvantages. The benefits of using computer vision for machine-based insect counts from imagery are discussed, culminating in the justification for selecting CNNs for this research.

2.1 Traditional Insect Monitoring Methods

Insect monitoring has undergone significant evolution due to advancements in methodology and technology. This review examines both traditional and contemporary approaches, analyzing their specifications, advantages, and limitations.

2.1.1 Direct Observation

The traditional method of directly observing insects has always been important for monitoring insect populations and identifying different species. Researchers record insect behavior in their natural habitats, which provides valuable qualitative data on how insects interact, use their habitats, and behave. This approach is especially essential for studying community dynamics in complex ecosystems such as tropical rainforests. However, direct observation has its challenges. It is labor-intensive

and requires researchers to spend extended periods in the field. The accuracy of the data depends on the observer's expertise and ability to identify insects in real-time. Additionally, human error and limited visibility in dense or inaccessible habitats can reduce the method's effectiveness. These limitations highlight the need for more efficient and less intrusive monitoring techniques, particularly in challenging environments (Häuser and Riede 2015).

2.1.2 Trapping Techniques

Trap techniques are crucial for traditional insect monitoring, allowing researchers to capture and study insects in diverse environments. Common traps include pitfall traps, light traps, malaise traps, and pheromone traps, see Figure 2.1. Each trap operates based on different principles such as attraction by light, pheromones, or physical entrapment, and is chosen depending on the study's specific goals and the target insect species. For example, as discussed by Häuser and Riede (2015), pitfall traps are very effective in sampling ground-dwelling insects, while light traps are widely used in monitoring nocturnal insects like moths. The primary advantage of trapping techniques is their ability to collect a large number of specimens over time, providing valuable data for population estimates and biodiversity assessments. Traps can be left unattended for extended periods, making them less labor-intensive than direct observation.

However, trapping techniques also have limitations. They can be selective, often biased towards certain types of insects or life stages, and may not effectively capture all species present in an area. Traps can also cause harm to the insects, which is a significant ethical consideration. Moreover, they require regular maintenance and monitoring, and the data collected often needs to be supplemented with other methods to provide a comprehensive understanding of insect populations (Montgomery et al. 2021).

2.1.3 Sampling Methods

Sampling methods such as sweep netting, vacuum sampling, and beat sampling are widely used for collecting insects from vegetation or soil. These methods allow researchers to cover large areas quickly, making them useful for quantitative studies in various habitats. Sweep netting (Figure 2.2) is particularly effective in open habitats like grasslands, while vacuum sampling (Figure 2.3) and beat sampling (Figure 2.4) are more suited to dense vegetation where insects are difficult to capture by hand.

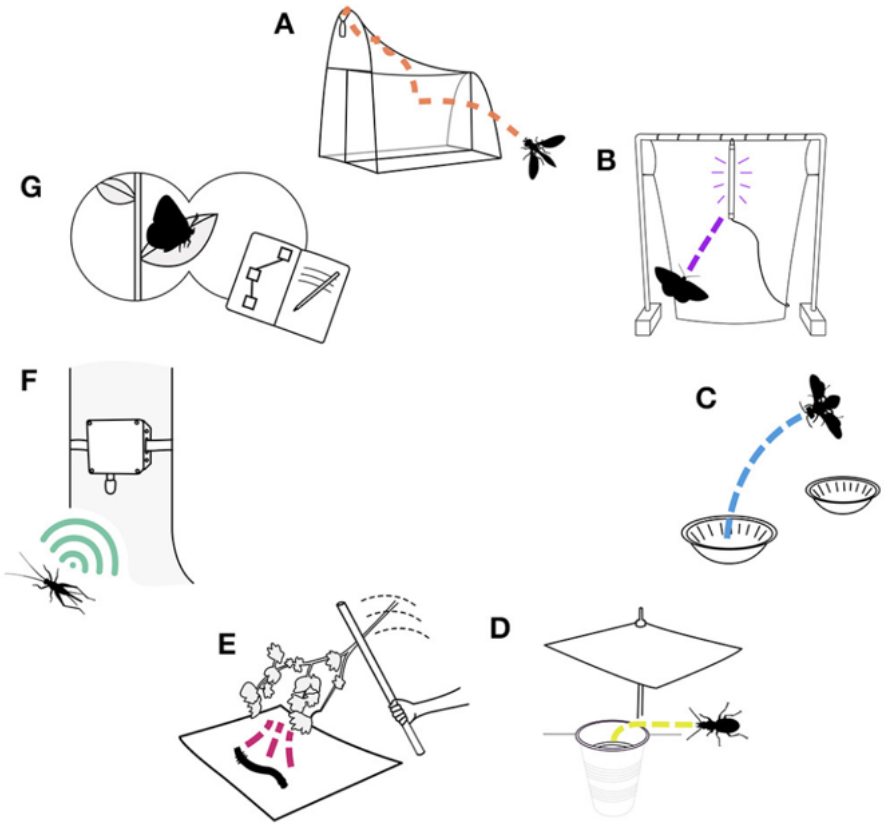

Figure 2.1: A visual overview of the seven insect benchmarking methods summarized here: (A) malaise trapping, (B) light trapping, (C) pan trapping, (D) pitfall trapping, (E) beating sheet, (F) acoustic monitoring, and (G) active visual surveys, adopted from Montgomery et al. (2021).

These techniques are advantageous for their efficiency and ability to sample a wide range of species in a relatively short amount of time. However, like other traditional methods, they come with certain drawbacks (Montgomery et al. 2021). Sampling can be destructive, potentially damaging the habitat and the specimens collected. Additionally, accurate identification of the collected insects often requires specialized knowledge and expertise, which may not always be available. Despite these challenges, sampling remains a vital tool in the field of entomology, particularly for studies requiring large datasets or those conducted in complex ecosystems (O'Connor et al. 2019).

Figure 2.2: Sweep sampling method, adopted from Vyavhare et al. (2015).

Figure 2.3: Vacuum sampling method, adopted from EcoTech (2024).

Figure 2.4: Beat sampling method, adopted from Holthouse et al. (2017).

2.2 Contemporary Insect Monitoring Methods

Advancements in modern technology have profoundly transformed insect monitoring, facilitating the development of more efficient, accurate, and scalable approaches. These contemporary methods leverage advanced tools such as automated image analysis and Unmanned Aerial Systems (UAS), significantly enhancing the ability to monitor insect populations across diverse environments (Valan et al. 2019).

2.2.1 Automated Image Analysis

Automated image analysis has revolutionized insect monitoring by integrating high-resolution cameras and sophisticated imaging devices with ML algorithms. This approach allows for the accurate identification and counting of insects within images, significantly reducing the dependency on manual, labor-intensive methods. For example, CNNs have shown particular effectiveness in classifying and counting insects with high precision, enabling real-time, large-scale monitoring (Valan et al. 2019).

2.2.2 Unmanned Aerial Systems

Unmanned Aerial Systems (UAS), or drones, have become essential tools in large-scale insect monitoring. These drones, equipped with high-resolution cameras, can survey extensive areas and capture detailed images, which are then analyzed using

advanced image processing algorithms (Carrasco-Escobar et al. 2022). This capability allows for comprehensive monitoring of insect populations and their habitats, including those in remote or otherwise inaccessible locations. Despite these advantages, the use of UAS in insect monitoring presents challenges, such as the high initial cost of equipment, complexities in data management, and the necessity for specialized software and expertise to process and analyze the data (Kumar et al. 2023), (Carrasco-Escobar et al. 2022). Moreover, the integration of UAS with ML algorithms has been shown to enhance the detection and classification of insect species, though this integration further increases the demand for computational power and sophisticated data processing (Kumar et al. 2023).

2.2.3 Integration of Machine Learning

The integration of ML techniques, including CNNs and Deep Learning (DL) models, has been transformative in insect monitoring. These methods automate the processes of detection, classification, and counting of insects from imagery, significantly improving both efficiency and accuracy. ML models are trained on labeled datasets to recognize patterns and features associated with different insect species. However, these methods face challenges such as the need for high-quality training data, the risk of overfitting, and substantial computational demands (Valan et al. 2019). Furthermore, maintaining model performance over time requires regular updates and retraining, especially in dynamic environments where insect populations and habitats may change (Kumar et al. 2023).

Traditional methods of insect monitoring, such as direct observation, trapping, and sampling, remain valuable but are increasingly supplemented by these advanced technologies. The shift towards automated, scalable, and less invasive methods reflects broader trends in ecological monitoring, aiming to improve accuracy and reduce the environmental impact of data collection. As these technologies continue to evolve, they offer promising opportunities for more effective and efficient insect monitoring (Valan et al. 2019), (Kumar et al. 2023).

2.3 Introduction to Artificial Intelligence

Artificial intelligence (AI) is a broad field of computer science aimed at creating systems capable of performing tasks that usually require human intelligence, such as problem-solving, decision-making, and pattern recognition (Russell and Norvig 2016). AI includes various techniques, from simple rule-based systems to advanced learning algorithms that can adapt and improve over time (Mitchell 1997).

2.3.1 Machine Learning

Machine learning (ML) is a subfield of AI focused on developing algorithms that can learn and make predictions or decisions based on data. Unlike traditional programming, where a computer follows explicit instructions, ML models identify patterns in data and improve their performance as they are exposed to more data (Mitchell 1997). This ability to learn from data makes ML an essential tool for tackling complex problems across various domains, such as image recognition, natural language processing, and recommendation systems (Goodfellow, Bengio, and Courville 2016).

Supervised Learning

Supervised learning is a key approach within ML where models are trained on labeled datasets. In this context, "labeled" means that each training example is paired with the correct output. This method is highly effective for tasks that require the classification of data into predefined categories, see Figure 2.5. For instance, in the field of entomology, supervised learning can be employed to detect and classify insect species from images, see Figure 2.6. Here, the training data consists of images annotated with species labels, enabling the model to learn the relationship between the visual features of the insects and their corresponding species. Once trained, the model can accurately classify new, unseen images of insects (Ramalingam et al. 2020). Examples of supervised learning algorithms include CNNs for image processing and traditional algorithms like K-Nearest Neighbour (KNN) and Support Vector Machine (SVM) (Goodfellow, Bengio, and Courville 2016).

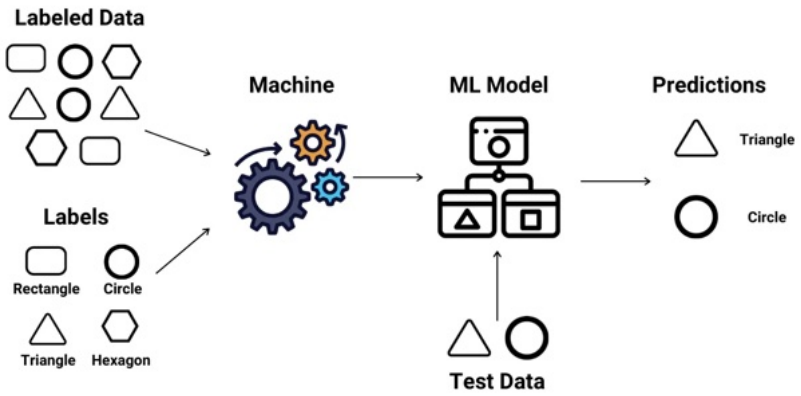

Figure 2.5: Understanding supervised learning: Example and diagram adopted from Kumar (2024).

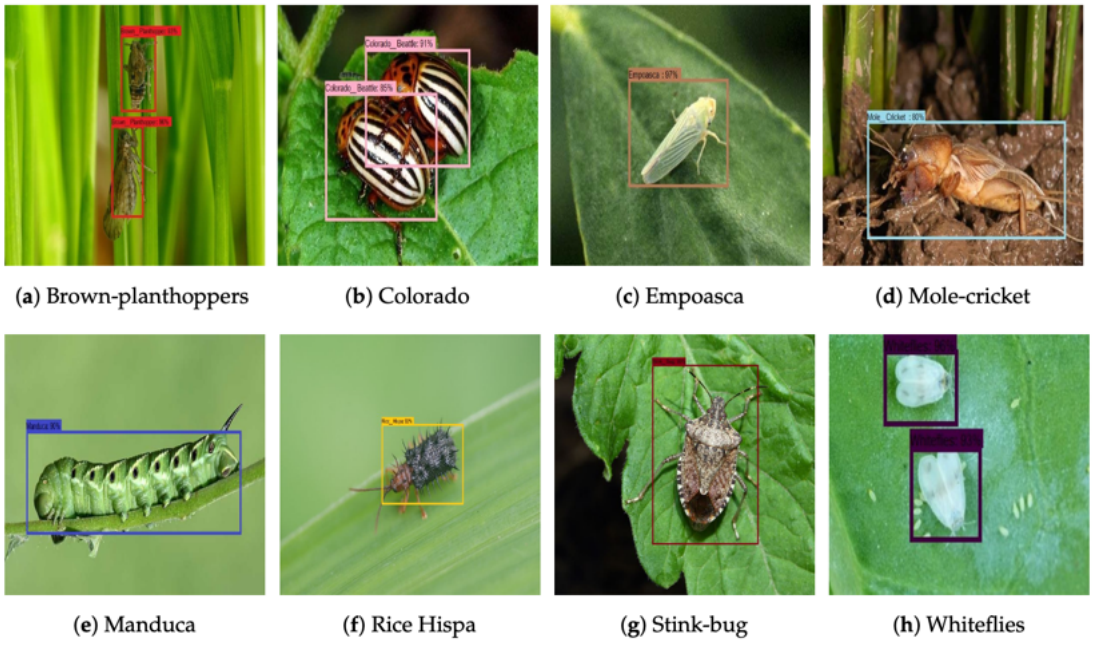

Figure 2.6: Species level labeling example for supervised learning, adapted from Ramalingam et al. (2020).

Unsupervised Learning

Unsupervised learning deals with unlabeled data, meaning the algorithm attempts to find patterns or structures in the data without prior knowledge of what the output should be, see Figure 2.7. This type of learning is particularly useful for exploratory data analysis, where the goal is to uncover hidden structures in the data.

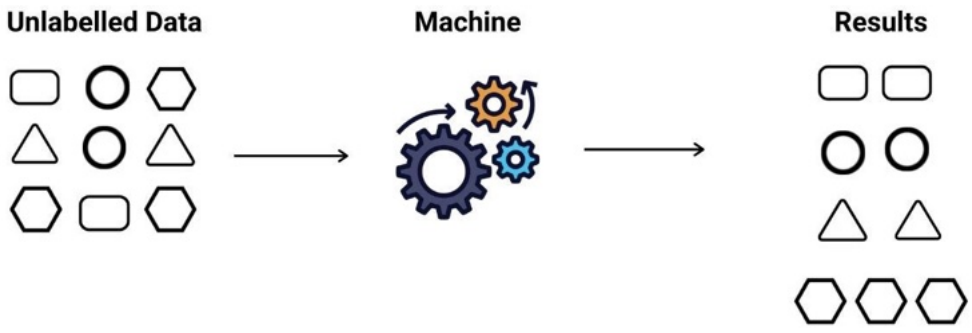

Figure 2.7: Understanding Unsupervised learning: Example and diagram adopted from Kumar (2024).

Techniques such as clustering and dimensionality reduction fall under unsupervised

learning. For example, clustering algorithms like K-means can group similar data points, which can be beneficial in tasks like market segmentation, where customers are grouped based on purchasing behavior (Mitchell 1997).

Reinforcement Learning

Reinforcement learning involves training an agent to make decisions by interacting with an environment. The agent receives rewards or penalties based on its actions, which guide it to learn an optimal strategy over time. This approach is particularly useful in scenarios where decision-making is sequential and the outcomes of actions are dependent on previous actions (Sutton and Barto 2018). Applications of reinforcement learning include game playing, where an agent learns to win games like AlphaGo by DeepMind (2014), where an agent learns to perform tasks such as walking or manipulating objects.

Figure 2.8 offers a clear and detailed illustration of the different types of ML, enhancing comprehension of the various approaches.

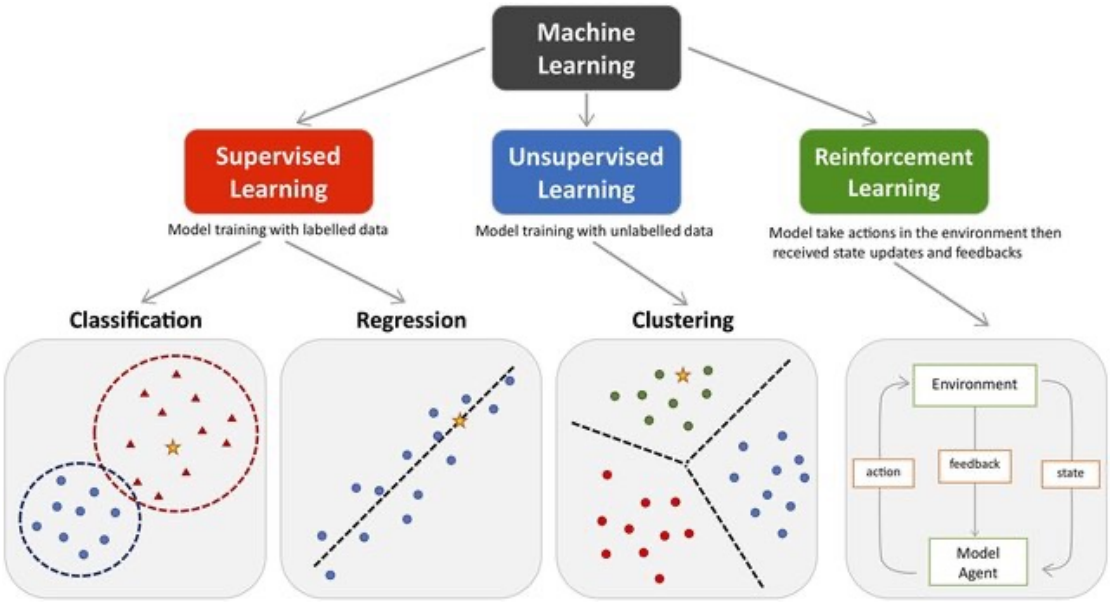

Figure 2.8: The main types of ML. The Main approaches include classification and regression under supervised learning and clustering under unsupervised learning. Reinforcement learning enhances the model performance by interacting with the environment. Colored dots and triangles represent the training data. Yellow stars represent the new data that can be predicted by the trained model, adopted from Peng et al. (2021).

2.3.2 Deep Learning

DL is a subset of ML that uses neural networks with multiple layers to analyze and comprehend complex patterns within large-scale datasets (LeCun, Bengio, and Hinton 2015). Figure 2.9 illustrates the relationship between AI, ML, DL.

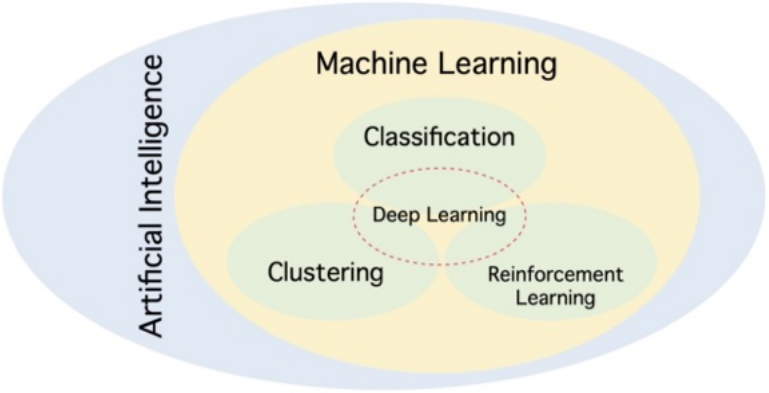

Figure 2.9: Relationship between AI, ML and DL, adapted from Date (2019).

Unlike traditional ML algorithms, which usually need manual feature extraction, DL models can automatically learn hierarchical features from raw data, making them highly effective in tasks that involve high-dimensional inputs, such as images and speech recognition (Goodfellow, Bengio, and Courville 2016).

Neural Networks and Their Components

Neural networks are composed of layers of interconnected neurons, which are basic computational units that process input data. Each neuron applies a weighted sum to its inputs, followed by a non-linear activation function to produce an output. These neurons are organized into layers that sequentially transform the data, with each layer learning increasingly abstract features from the input (Goodfellow, Bengio, and Courville 2016).

- Neurons and layers: At the heart of DL models are neurons, which are computational units that apply a weighted sum of inputs, followed by a non-linear activation function to produce outputs (Sharma, Sharma, and Athaiya 2017). These neurons are organized into layers as shown in Figure 2.10.
  - Input layer: receives raw data (e.g., pixel values of an image) and passes it to subsequent layers.
  - Hidden layers: comprise multiple layers of neurons where each layer learns increasingly abstract features from the data. For instance, in

image processing, early layers might detect edges, while deeper layers could recognize more complex structures like shapes or textures (Zeiler and Fergus 2014).

- Output layer: produces the final prediction or classification result. For object detection, this may involve predicting bounding boxes and class labels (Krizhevsky, Sutskever, and Hinton 2012).

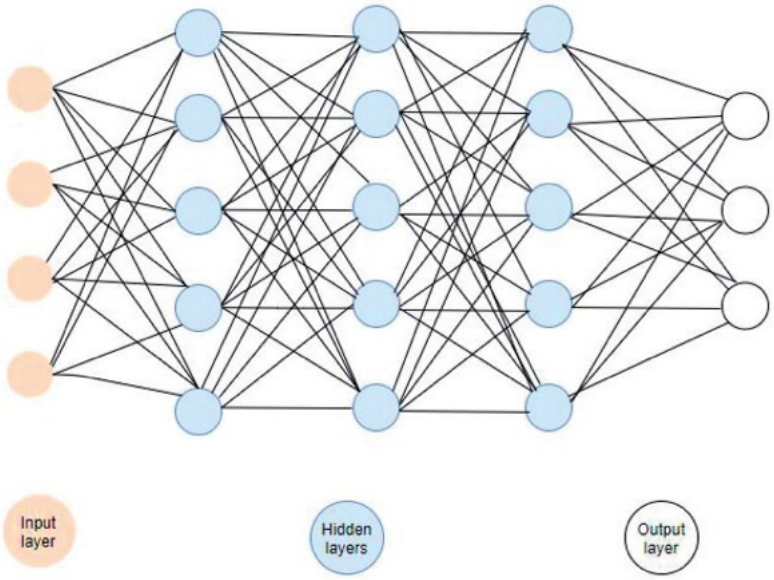

Figure 2.10: Deep neural network layers, adopted from Saadat and Shuaib (2020).

- Activation functions: Activation functions play a critical role in neural networks by introducing non-linearity into the model, which allows the network to learn complex patterns and relationships between the input data and the output. Without non-linearity, no matter how many layers the network has, it would behave as a linear model, which severely limits its capability to model real-world data that is often non-linear in nature (Sharma, Sharma, and Athaiya 2017). Several commonly used activation functions in DL include:
  - Rectified linear unit: Rectified Linear Unit (ReLU) is one of the most widely used activation functions in DL due to its simplicity and efficiency. Meaning it outputs the input directly if the value is positive, and outputs zero otherwise. The non-linearity introduced by ReLU allows the network to handle complex data, and it addresses the vanishing gradient problem by allowing gradients to flow effectively through deep layers. Since ReLU outputs zero for negative values, it ensures sparse

activations (where only a subset of neurons are activated at a time), improving both efficiency and model performance.The ReLU function is defined as Equation 2.1 and shown as Figure 2.11.

$$f(x) = \max(0, x) \quad \text{(Sharma, Sharma, and Athaiya 2017)} \tag{2.1}$$

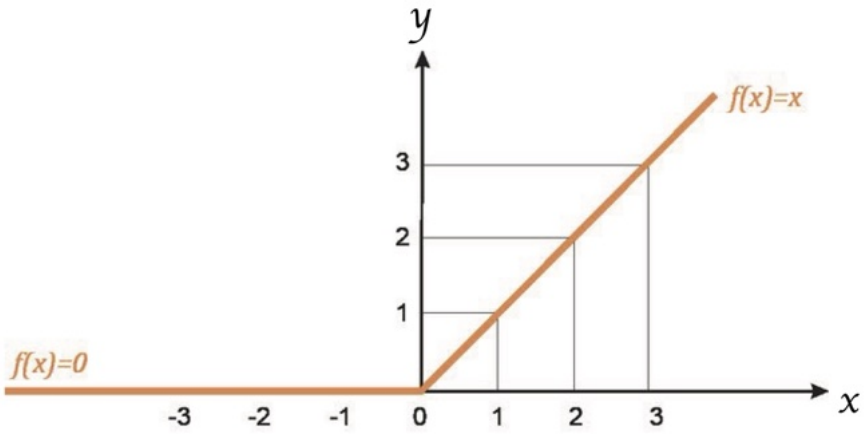

Figure 2.11: ReLU activation function, adapted from Es-sabery et al. (2021).

– Leaky ReLU: Leaky ReLU addresses one of the limitations of standard ReLU, its tendency to "die" during training when inputs are negative. In Leaky ReLU, a small slope is applied to negative inputs instead of outputting zero, which ensures that negative values can still propagate some gradient during backpropagation. Mathematically, it's defined as This small adjustment helps prevent neurons from "dying," especially when many negative inputs are encountered. The ReLU function is defined as Equation 2.2 and shown as Figure 2.12.

$$f(x) = \max(0.01x, x) \quad \text{(Sharma, Sharma, and Athaiya 2017)} \tag{2.2}$$

This ensures that gradients can still flow even for negative inputs (Li, Nash, et al. 2022).

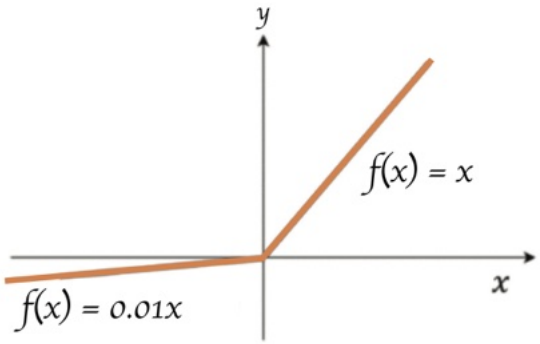

Figure 2.12: Leaky ReLU activation function, adapted from Li, Nash, et al. (2022).

– Sigmoid: The Sigmoid function maps input values to an output range between 0 and 1, making it useful for binary classification problems. The function is defined as Euler's number. Although the Sigmoid function can effectively map outputs into probabilities, it tends to suffer from the vanishing gradient problem because its gradients become very small for large positive or negative inputs, slowing down learning in deep networks (Sharma, Sharma, and Athaiya 2017). The Sigmoid function is defined as Equation 2.3 and shown as Figure 2.13.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad \text{(Goodfellow, Bengio, and Courville 2016)} \qquad (2.3)$$

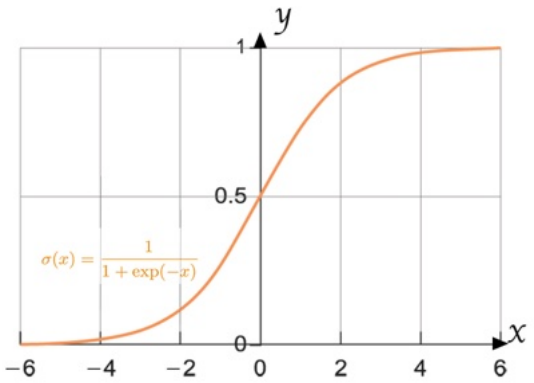

Figure 2.13: Sigmoid activation function, adapted from Es-sabery et al. (2021).

– Tanh (Hyperbolic Tangent): The Tanh function is similar to the Sigmoid function but with outputs ranging from -1 to 1, making it more

suitable for situations where negative values are needed to represent the data. Tanh is defined as Like Sigmoid, Tanh is non-linear and differentiable, allowing backpropagation, but it performs better than Sigmoid in practice, because its output is zero-centered, meaning the data passed to subsequent layers, will have both negative and positive values, leading to faster convergence during training. However, like Sigmoid, Tanh can also experience the vanishing gradient problem. The Tanh function is defined as Equation 2.4 and shown as Figure 2.14.

$$\sigma(x) = \frac{2}{1 + e^{-2x}} - 1 \quad \text{(Sharma, Sharma, and Athaiya 2017)} \qquad (2.4)$$

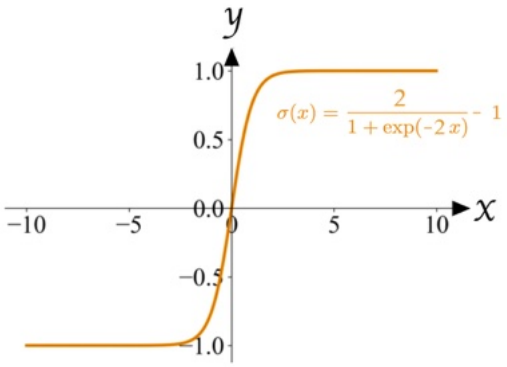

Figure 2.14: Tanh activation function, adapted from Feng et al. (2019).

– SoftMax: Softmax is typically used in the output layer for multi-class classification problems. It converts the raw outputs (logits) of the network into probabilities that sum to 1 across different classes, making it possible to interpret the output as a probability distribution over multiple categories. The SoftMax function is defined as Equation 2.5 and shown as Figure 2.15.

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \quad \text{(Goodfellow, Bengio, and Courville 2016)} \qquad (2.5)$$

where x_i represents the input to the SoftMax function for the i-th class, and the denominator sums over all classes j.

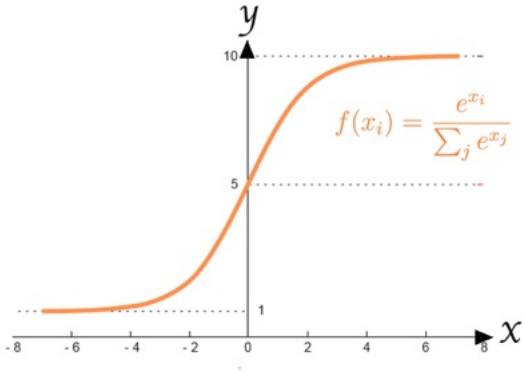

Figure 2.15: SoftMax activation function, adapted from Es-sabery et al. (2021).

Softmax is particularly useful when the model must choose among multiple classes and return the most likely one (Sharma, Sharma, and Athaiya 2017).

- Training: Training DL models is the process of optimizing the model's parameters (weights and biases) to minimize a loss function, which quantifies the difference between the predicted output and the actual target values (Kingma and Ba 2014). The training process involves several key steps:
  - Loss Function: The loss function measures how far the model's predictions are from the true labels. In classification tasks, cross-entropy loss is often used to measure the difference between the predicted probability distribution and the actual class label. For regression tasks, mean squared error (MSE) is commonly used, which calculates the squared difference between the predicted and actual values. Minimizing the loss function is the main goal of training (Goodfellow, Bengio, and Courville 2016). Common loss functions include:

    - **Mean Squared Error (MSE)**: typically used in regression tasks, MSE is defined as Equation 2.6

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \quad \text{(Goodfellow, Bengio, and Courville 2016)} \tag{2.6}$$

    where y_i is the true value, $\hat{y}_i$ is the predicted value, and n is the number of samples.

- **Cross-Entropy Loss**: used for classification, it measures the difference between predicted probabilities and true labels, see Equation 2.7 (Goodfellow, Bengio, and Courville 2016):

$$\text{Cross-Entropy Loss} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \quad (2.7)$$

where y_i is the true label (0 or 1), and $\hat{y}_i$ is the predicted probability.

– In the forward pass, input data is passed through the network, and each layer performs two key operations: applying weights and biases, and using an activation function (LeCun, Bengio, and Hinton 2015). The output of each layer becomes the input for the next layer. Mathematically, The input x is multiplied by the weight w, and bias b is added, see Equation 2.8.

$$z = w \cdot x + b \quad \text{(Goodfellow, Bengio, and Courville 2016)} \quad (2.8)$$

The result z is passed through an activation function such as ReLU or Sigmoid to introduce non-linearity, see Equation 2.9.

$$a = \text{ReLU}(z) = \max(0, z) \quad \text{(Goodfellow, Bengio, and Courville 2016)} \quad (2.9)$$

– Backpropagation: Backpropagation is the process of calculating how much each weight and bias contributed to the error. It uses the chain rule to calculate gradients layer by layer (LeCun, Bottou, et al. 1998). The gradient for a single weight w is computed as Equation 2.10.

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w} \quad \text{(LeCun, Bottou, et al. 1998)} \quad (2.10)$$

where L is the loss and w is the weight.

– Optimization Algorithms: After calculating gradients, optimization algorithms are used to update the model's weights. The most basic algorithm is Stochastic Gradient Descent (SGD), where the weights are

updated as Equation 2.11 (LeCun, Bottou, et al. 1998)).

$$w = w - \eta \cdot \frac{\partial L}{\partial w} \quad \text{(LeCun, Bottou, et al. 1998)} \tag{2.11}$$

where η is the learning rate.

– Iterative Process: Training is an iterative process where the model goes through the entire dataset multiple times, with each full pass through the data called an epoch (Goodfellow, Bengio, and Courville 2016). During each epoch, the model learns from the data and adjusts its weights to minimize the loss function. Instead of updating the weights after processing every single data point (which can be slow), the dataset is typically divided into smaller groups called batches. After processing each batch, the model updates its parameters. The batch size refers to how many samples are in each batch. Training with batches can speed up the learning process, and using mini-batches introduces some noise into the updates, which can help avoid local minima (LeCun, Bengio, and Hinton 2015). Figure 2.16 illustrates the local minima problem.

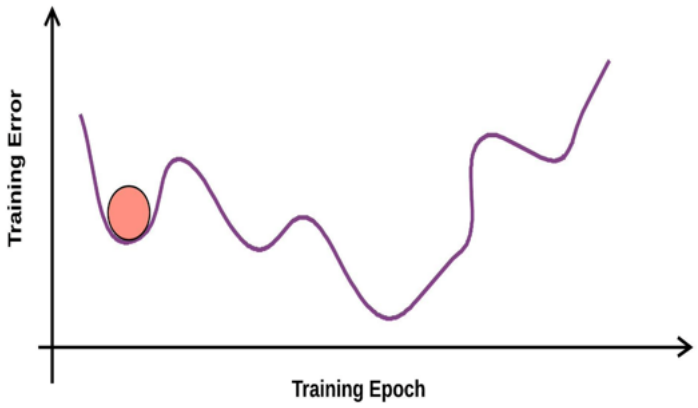

Figure 2.16: Illustration the sample of optimizer stuck in a local minima problem, adopted from Ghosh et al. (2020).

Convolutional Neural Networks

CNNs are a specialized class of ANNs designed to process data with a grid-like topology, such as images. Unlike traditional ANNs, CNNs use the spatial structure of the input data to perform pattern recognition tasks efficiently. Initially developed for image analysis, CNNs have been adapted for various applications,

including natural language processing and other domains requiring spatial feature extraction (Yamashita et al. 2018).

CNNs share several characteristics with traditional ANNs, such as the use of input, hidden, and output layers. However, CNNs introduce unique layer types within the hidden layers that are crucial for their success in pattern recognition. These include convolutional layers, pooling layers, and fully connected layers (Gopika et al. 2020). Additionally, activation function layers (or nonlinearity layers) and normalization layers are often integrated into the network to enhance performance (Paoletti et al. 2018).An example of CNN architecture is shown in Figure 2.17.

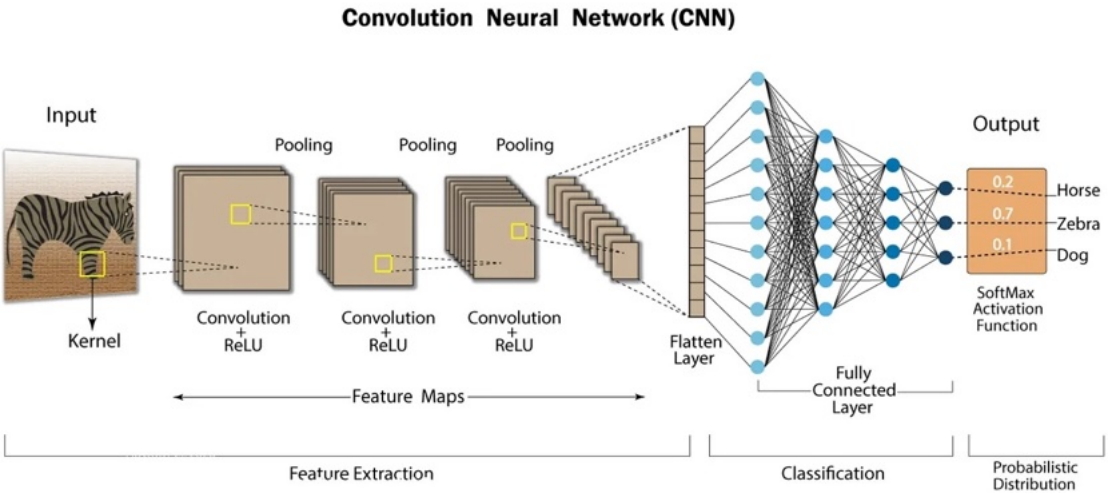

Figure 2.17: Convolutional neural network — CNN architecture, adopted from Shahriar (2021).

- Convolutional Layers: Convolutional layers are fundamental components of a CNN, responsible for learning and detecting features from input data such as images, audio, or video. Unlike fully connected layers, convolutional layers use a mathematical operation called convolution to process input data in a way that is well-suited for tasks involving spatial data, like image recognition (Gopika et al. 2020). Convolution is a specialized linear operation that multiplies a small matrix of weights, known as a filter or kernel, with a section of the input data (such as a portion of an image). The kernel slides across the input data, performing a dot product between the kernel's values and the values in the local receptive field of the input. The result of this operation is a feature map, which highlights important patterns in the data, such as edges, textures, or specific shapes (O'shea and Nash 2015), see Figure 2.17. In a CNN, the convolution operation between the input data I and filter F can be defined as Equation 2.12 adapted from Wu (2017).

$$S(i,j) = (I * F)(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i+m, j+n) \cdot F(m,n) \tag{2.12}$$

Where I is the input matrix (e.g., an image or feature map from a previous layer), I is the filter (also known as the kernel), $S(i,j)$ is the resulting feature map after convolution, $M \times N$ is the size of the filter. The filter slides over the input, and at each position, the dot product of the filter and the corresponding local region of the input are computed. The output is a feature map, which indicates the presence of certain features (like edges or textures) learned by the filter.

- Convolutional kernel: The kernel (filter) is a small matrix, usually 3x3 or 5x5, which is initialized with random weights. During training, the CNN learns the optimal weights of these filters to detect various patterns such as edges, corners, textures, and more complex structures in the data, see Figure 2.18. Each convolutional layer can have multiple filters, each detecting different features (Ghosh et al. 2020).

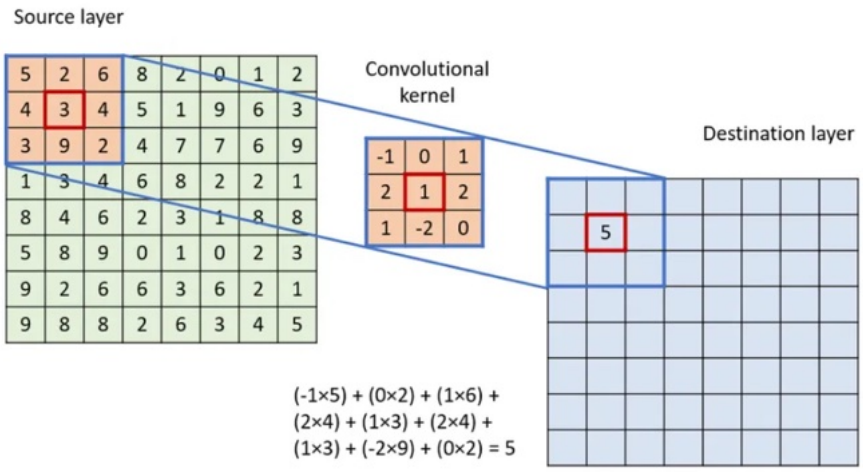

Figure 2.18: Convolutional kernels (also known as filters) are updated during the backpropagation process in CNNs, adopted from Abhishek (2022).

- Feature map: The result of applying a filter to the input is called a feature map. Each feature map highlights specific patterns detected by a particular filter. For example, in an image, one feature map might detect horizontal edges, while another might detect vertical edges. As more convolutional layers are added, the feature maps become more abstract, capturing higher-level patterns such as shapes or objects (Wu 2017), see Figure 2.19.

- Pooling Layers: Pooling layers, or subsampling layers, follow convolutional layers to reduce the spatial dimensions of the feature maps. This down-sampling process helps minimze the computational complexity and mitigate overfitting. The most common pooling method is max-pooling, which retains the highest value from each region of the feature map, although average-pooling is also used in some cases (Zhu et al. 2018). Figure 2.19 illustrates the steps from an input image to the pooling layer, with the computation of two different pooling layers.

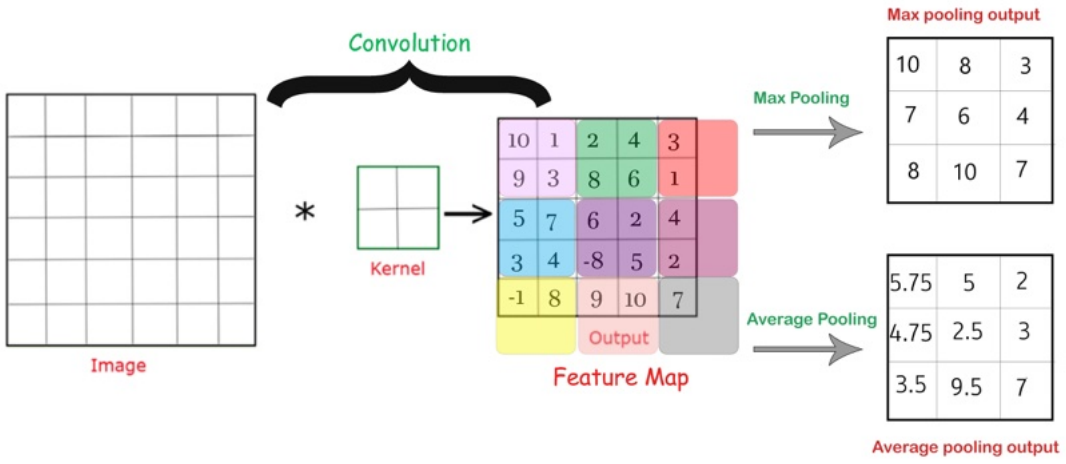

Figure 2.19: Max pooling and average pooling example on a $4times4$ matrix, adapted from Dixit (2021).

- Fully Connected Layers: The fully connected layer, also known as the dense layer, is typically positioned towards the end of a CNNs. It serves as the final stage where the high-level feature representations learned by the preceding convolutional and pooling layers are transformed into a format suitable for the network's output task, such as classification or regression (Li, Gu, et al. 2019; Ranjbar et al. 2020). Unlike convolutional layers, which maintain spatial relationships and local connectivity, the fully connected layer connects every neuron in one layer to every neuron in the next layer. This means each neuron in the fully connected layer receives input from all neurons in the preceding layer, allowing it to integrate information from the entire feature map produced by the convolutional layers (Ranjbar et al. 2020). The primary role of the fully connected layer is to transform the learned feature maps into a

1-dimensional vector that can be used for output purposes. This transformation involves a series of linear operations (weighted sums) followed by an activation function. The fully connected layer aggregates and combines the features in a way that highlights the most relevant characteristics for the final task (Li, Gu, et al. 2019).

In classification tasks, the final fully connected layer often employs a SoftMax activation function to convert the raw output scores into a probability distribution over the target classes. The SoftMax function ensures that the sum of the probabilities is 1, making it suitable for multi-class classification problems (Yamashita et al. 2018).

2.3.3 Deep Learning Frameworks for Insect Detection and Classification

The following DL frameworks for detection and classification will be discussed as they are used for this thesis:

- TensorFlow, developed by Google, is a robust open-source library for DL. It allows for the extensive distribution of computational tasks across Central Processing Unit (CPU)s and Graphics Processing Unit (GPU)s. TensorFlow comes with a comprehensive ecosystem that includes TensorFlow Lite for mobile devices and TensorFlow Extended (TFX) for production pipelines. TensorFlow is known for its scalability, flexibility, and extensive community support. It provides robust tools for deploying models in production and integrates seamlessly with Keras, enhancing its usability for complex tasks like insect detection (Martín Abadi et al. 2015). TensorFlow is easily accessible via Python in our current setup, which includes Python 3.11.8, making integration with our insect detection tasks straightforward.

- Keras is another open-source neural network computing library mainly developed in the Python language by Chollet et al. (2015). It's highly modular and allows for rapid model building and training without excessive professional knowledge. Keras is user-friendly and ideal for rapid prototyping, making it great for developing and testing deep-learning models. Its integration with TensorFlow provides access to advanced features and optimizations, ensuring it remains powerful and accessible (Long and Zeng 2022). Keras, integrated with TensorFlow, is also available via Python 3.11.8, supporting efficient model development in our environment.

- PyTorch is a Python package that provides tensor computation (like NumPy) with strong GPU acceleration and deep neural networks built on a tape-based autograd system. It allows users to reuse their favorite Python packages such as NumPy, and SciPy, to extend PyTorch when needed. PyTorch is appreciated for its dynamic computation graph, flexibility, and simplicity in model development (Ansel et al. 2024). It is particularly useful for research and experimental tasks, making it suitable for developing and fine-tuning insect detection models. PyTorch's imperative execution makes debugging and developing models straightforward, ensuring efficient and effective model training. PyTorch, along with our current setup (Python 3.11.8 and Torch 2.0.0+cu117), is fully available for seamless model training and experimentation.

2.3.4 Related Work in Insect Detection Using Convolutional Neural Networks

In recent years, significant advancements have been made in insect detection using CNN, particularly in the domain of outdoor image classification. Two prominent studies form the foundation of this exploration. One study, conducted in Europe, focused on the application of DL models for detecting pest insects in the context of Precise Agriculture Butera et al. (2021). The other carried out in China, investigated insect detection and counting using an improved YOLOv3 model Li, Zhu, and Li (2021). Both studies highlight the potential of CNN-based detection systems, though they target different insect species and deploy distinct methodologies.

Precision Agriculture and Pest Monitoring by Butera et al. (2021)

The study by Butera et al. (2021) aimed to explore various DL models for the detection of pest insects, particularly focusing on distinguishing harmful species like Popillia japonica from harmless ones such as Cetonia aurata and Phyllopertha horticola. The research employed two types of object detection models:

- Two-stage detectors: Prioritized accuracy, with Faster R-CNN being selected for its precision.
- One-stage detectors: Focused on speed, with Single Shot MultiBox Detector (SSD) and RetinaNet discussed for their balanced performance.

A key contribution of this study was the creation of a high-quality dataset tailored to the agricultural context, addressing issues of data duplication and relevance. The results demonstrated that Faster R-CNN with a MobileNetV3 backbone achieved a mAP score of 92.66% and maintained a low false-positive rate of 2.26%, as shown in Table 2.1. These metrics emphasize the model's ability to effectively detect target insects while minimizing misclassifications, an essential factor for precision in agricultural pest control systems.

In comparison, our research expands the scope to the detection and classification of a broader array of insect species, beyond the agricultural pest context. The aim is to develop a generalizable and robust detection system across six insect categories. Similar to Buetra's study, our work emphasizes the robustness of CNN models in insect detection and classification, using key metrics such as mAP, false positive rate, and inference speed to assess model performance.

Table 2.1: Performance of the compared models in terms of mean average precision on the test set and frames per second at inference time. False positives count for class Popillia Japonica is also shown, adapted from Butera et al. (2021).

Backbone	**Detector**	**mAP**	**FPS (GPU)**	**FPS (CPU)**	**PJ FP Count**
VGG16_FPN	FasterRCNN	0.9242	11.90	0.55	65
ResNet101_FPN	FasterRCNN	0.9214	11.23	0.17	20
DenseNet169_FPN	FasterRCNN	0.9107	17.59	0.26	28
MobileNetV3_FPN	FasterRCNN	0.9266	60.92	0.84	12
VGG16_FPN	RetinaNet	0.9125	12.37	0.36	196
ResNet101_FPN	RetinaNet	0.9330	11.83	0.20	169
DenseNet169_FPN	RetinaNet	0.9170	18.49	0.29	206
MobileNetV3_FPN	RetinaNet	0.9066	48.91	0.78	364
VGG16	SSD	0.5570	42.58	4.27	221
ResNet101	SSD	0.8731	29.76	0.74	55
DenseNet169	SSD	0.9238	28.59	0.94	35
MobileNetV3	SSD	0.8044	33.20	1.41	67

Insect Detection and Counting Based on YOLOv3 Model by Li, Zhu, and Li (2021)

Li, Zhu, and Li (2021) took a different approach by focusing on insect detection and counting, utilizing the YOLOv3 model with a Cross-Stage Partial (CSP)Darknet53 backbone and Complete Intersection Over Union (CIOU) loss function. This model was specifically designed to optimize speed and accuracy in counting insect populations, using a dataset of over 5,500 images containing five insect types: scarabs, mosquitoes, bees, moths, and crickets. Manual annotations were meticulously prepared with the LabelImg tool, ensuring that the dataset represented a realistic and diverse sample of insect images.

Li, Zhu, and Li (2021) reported a mAP score of 90.62% for their improved YOLOv3 model, showcasing its robustness in detection tasks. Table 2.2 presents the final results of this study by Li, Zhu, and Li (2021) comparing the robustness of the YOLOv3 model to the Faster R-CNN (VGG16) and Faster R-CNN (Residual Network (ResNet)50) models in detecting five different insects in the dataset. While YOLOv3 excels in speed, it presents challenges in distinguishing densely populated insect clusters, which is a critical aspect of practical applications. Our thesis builds upon these insights, as the detection of insects in complex environments requires not only high accuracy but also the ability to scale across different species categories without significant performance loss.

While Li's study provided meaningful contributions in terms of speed and counting accuracy, the architectural limitations of YOLOv3, particularly its difficulty with dense populations, highlight a gap that our thesis addresses. By evaluating additional models like YOLOv8 and Faster R-CNN, we seek to optimize both detection accuracy and computational efficiency, making the system more robust in real-world scenarios.

Table 2.2: Comparison of Faster-RCNN and YOLOv3 models in detecting five different insects, adapted from Li, Zhu, and Li (2021).

Model	**Moth**	**Bee**	**Cricket**	**Scarab**	**Mosquito**	**Map**
Faster-RCNN (VGG16)	88.55%	80.78%	82.73%	89.81%	84.74%	85.33%
Faster-RCNN (ResNet50)	88.85%	81.30%	83.02%	90.11%	85.11%	87.04%
YOLOV3 (IOU)	80.02%	94.12%	85.26%	89.32%	88.16%	87.37%
YOLOV3 (CIOU)	97.10%	91.23%	90.11%	90.14%	85.04%	90.62%

2.3.5 Related Work in Waterfowl Bird Detection Using Convolutional Neural Networks by Sa'doun et al. (2021)

Building on the previous effort by Sa'doun et al. (2021), which focused on comparing different CNN architectures for waterfowl species detection and classification using UAV imagery, our research aims to extend the application of CNN architectures to the field of entomology. Specifically, our target is to develop and evaluate CNN-based methods for the detection and classification of various insect species. Sa'doun et al. (2021) compared several CNN architectures, including Faster R-CNN, YOLO, and RetinaNet, to identify the best-performing models for detecting waterfowl species across multiple detection levels. Their work provides a valuable comparative framework for evaluating CNN-based models in object detection tasks and underscores several critical challenges:

- Increasing the number of species classes led to a reduction in detection accuracy, a challenge also present in insect detection when expanding species categories.
- Dense populations and unclear image backgrounds posed significant obstacles to classifying species precisely. This mirrors the challenge faced in insect detection, where similar species often share overlapping visual characteristics in crowded environments.
- Faster R-CNN exhibited consistent labeling across different detection levels, outperforming YOLO and RetinaNet in class-level accuracy. This is particularly relevant in contexts where model consistency is crucial, such as differentiating between closely related insect species.

Their work, the combination of UAV imagery and CNNs proved highly effective in species detection but was hindered by dataset limitations, class imbalance, and environmental factors like high population density. These limitations directly inform our research, as similar challenges arise in the task of insect detection. By applying similar architectures, including YOLOv8 and Faster R-CNN and RetinaNet, our thesis builds upon Sa'doun et al. (2021) findings to enhance detection accuracy across multiple insect species by highlighting species scale sensitivity

Their work provides a foundational understanding of how CNNs can be deployed for species detection in natural environments. Their findings regarding class imbalance, dataset size, and environmental challenges are directly applicable to insect detection tasks, where diverse environmental conditions and species overlap to create additional complexities.

By expanding on their approach, our research seeks to apply these architectures to insect detection, evaluating how well they perform when detecting multiple species with diffrent sizes across diverse categories. Similar to their focus on UAV imagery, our thesis considers how environmental factors, such as image density and clutter, impact model performance and aims to address these through improved dataset curation and model training.

CHAPTER 3 Approach

In this chapter, we provide a comprehensive overview of the necessary steps for implementing our insect detection and classification system. We outline the entire workflow, from the data acquisition to the deployment of our models.

Our project is divided into eight main phases, as explained in 1.6, each crucial for the successful implementation of an insect detection and classification system using advanced CNN architectures. We begin with data acquisition, followed by the justification of our chosen models, data preprocessing, model implementation, and finally, the evaluation phase. Figure 3.1 provides a visual breakdown of these phases from data acquisition to the evaluation phase.

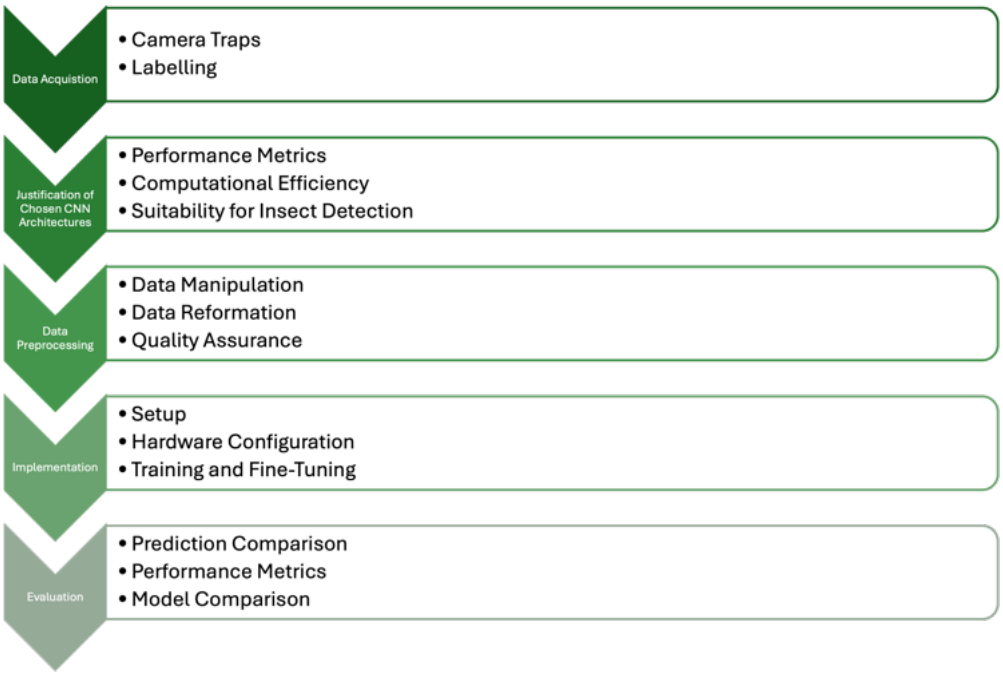

Figure 3.1: Workflow from data acquisition to the evaluation phase of the thesis

.

3.1 Data Acquisition

The first phase involves the acquisition of data, which was conducted by the UNESCO CHAIR on Sustainable Management of Conservation Areas in summer 2022 at Fachhochschule Kärnten led by Daniel Dalton. Although our primary focus is on utilizing this dataset for model training and evaluation, the data acquisition phase is included to provide a comprehensive understanding of the application domain and its significance. This phase represents a substantial asset to the study and enriches the overall research context. The dataset for our insect detection project was collected using camera traps installed at three different locations: St. Margarethen in Lungau, Weinitzen, and Sittersdorf (Carinthia). The focus was on capturing images of insects visiting blooming annual plants, which provided a diverse set of images under various environmental conditions.

- St. Margarethen in Lungau: Camera traps were set up on six different dates in a private garden, positioned along a willow hedge by a small stream. The cameras were focused on blooming plants, including Borago officinalis, Filipendula ulmaria, Inula helenium, and Hypericum perforatum. Tripods were attached to strong willow branches to photograph flowers on tall-standing plants.
- Weinitzen: The Malaise trapping program, managed by Landesmuseum Kärnten, has been in operation for a long time and it facilitated data collection. The cameras were placed near the Malaise trap to capture images of insects visiting flowers. The traps primarily focused on local blooming plants near the Malaise trap.
- Sittersdorf: Camera traps were set up in two locations within a private garden; an intensively managed vegetable and flower garden and a wildflower meadow. Focus plants included Achillea millefolium and Borago officinalis.

Figure 3.3 shows camera traps were installed on flat ground with intermittent shade to document each plant's flower-visiting insect.

Figure 3.2: Flat area on which the cameras were placed. Cameras are circled in red.

3.1.1 Camera Settings and Challenges

In the Insect Monitoring project led by Dalton, the camera setup and operation posed several significant challenges during the monitoring trials. Specifically, the use of Ricoh WG-70 all-weather digital cameras, see Figure 3.3, with macro photographic capabilities was chosen to capture images of insects visiting flowers. These cameras were mounted on tripods and programmed to take up to 1000 images per day through a time-lapse function. In addition, sticky traps were used initially to potentially allow genetic barcoding of captured insects but were later discontinued due to interference with macro settings and excessive capture of non-target insects.

Figure 3.3: Ricoh WG-70 digital cameras are used for capturing images.

The Insect Monitoring project encountered several issues related to camera overheating and autofocus problems, which are critical challenges for the effectiveness of long-duration outdoor insect monitoring:

- Camera overheating: The Ricoh WG-70 cameras experienced overheating when exposed to direct sunlight for extended periods. The surface temperature of the cameras was monitored and, at one point, reached between 60°C and 68°C. Overheating caused the cameras to shut down intermittently, leading to gaps in the time-lapse image capture. Although attempts were made to shield the cameras using parasols, wind made this approach ineffective. Overheating was a persistent issue, especially when cameras were exposed to full sunlight, as seen in Sittersdorf.

- Autofocus issues: Autofocus settings led to inconsistent image quality. While the cameras were supposed to focus on insects visiting flowers, many images were blurred or focused on the background instead of the targeted flower,

making it difficult to accurately detect insects. In several instances, such as with the plant Borago officinalis, the camera repeatedly focused on the background, rendering the images unusable for the project's purposes.

- Battery limitations: Another notable challenge was the inaccurate battery indicator on the Ricoh cameras. The cameras displayed a full battery even when not fully charged, leading to early battery depletion during the photo sessions. This issue could have led to missed data collection during critical periods of insect activity.

- Environmental factors: The positioning and orientation of the cameras had a substantial impact on their performance. Cameras placed with their screens facing upward were more prone to overheating, and adjusting the tripod height and position was critical to ensuring that the target flowers remained within the camera's view throughout the day. Uneven terrain and plant growth further complicated the setup, as some plants were not tall enough for the cameras to capture efficiently.

3.1.2 Data Annotation

Labelbox (2024) is a platform designed to streamline the process of annotating and labeling data for ML projects. It offers a user-friendly interface, supports various data formats, provides collaboration tools, and integrates with third-party applications, see Figure 3.4. In Sa'doun et al. (2021)'s waterfowl detection research, YOLOv3 was trained to recognize different waterfowl species, with Labelbox playing a key role in precisely labeling the training data. The platform's features were instrumental in ensuring accurate species identification.

In our insect detection project, we built upon the Insect Monitoring project led by Dalton, utilizing their pre-labeled datasets as part of our data collection. This significantly eased the data acquisition phase, which was crucial for the development of the YOLOv8 model. By leveraging the annotated datasets rom Dalton's team, we were able to focus on refining the classification of insect species, bypassing the need for initial, labor-intensive data collection. Similar to the waterfowl detection task, Labelbox facilitated the efficient management and annotation of insect images, contributing to an accurate and streamlined training process for our model. This collaborative use of annotated data expedited the development of our insect detection system. Figure 3.5 illustrates an example of an exported .txt file from Labelbox that contains the bounding box coordinates related to YOLOv8 for a labeled insect as class 1 which is Cloeptera.

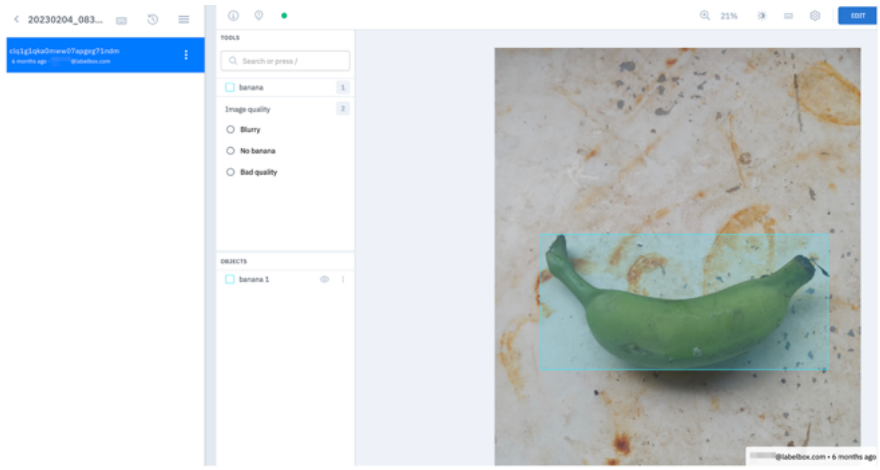

Figure 3.4: Labelbox interface for creating bounding boxes for custom datasets such as banana, adapted from Labelbox (2024)

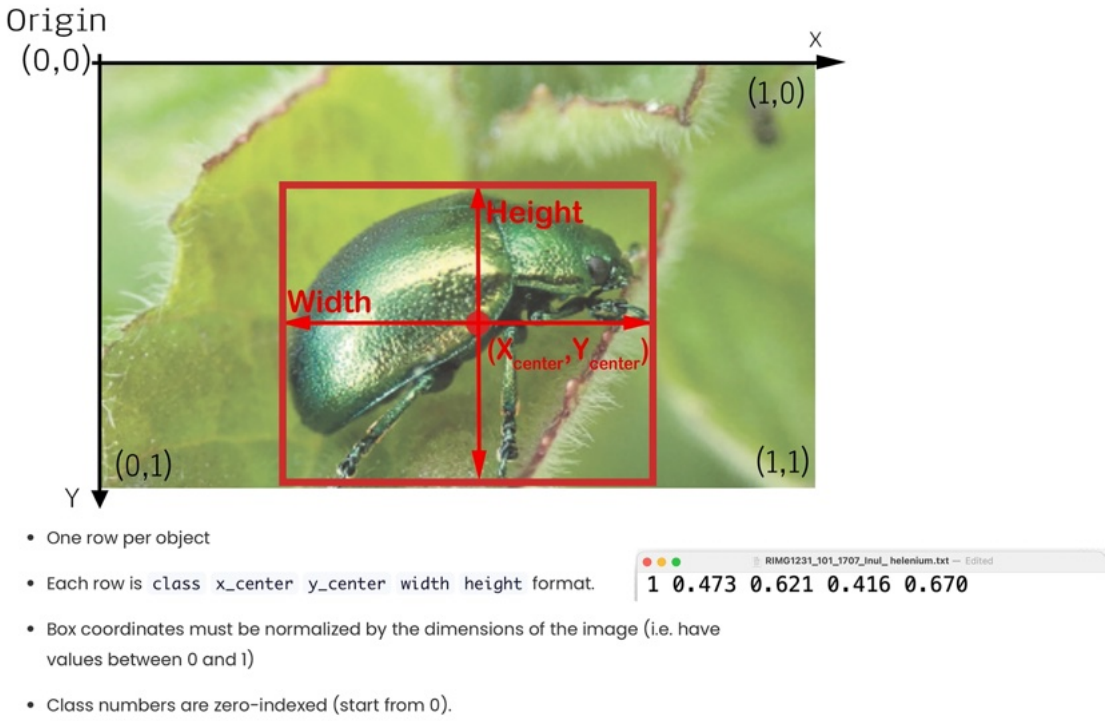

Figure 3.5: An example of bounding box coordinates in YOLOv8, adapted from Jocher, Chaurasia, and Qiu (2023)

3.2 Justification of Chosen CNN Architectures

The next phase involves selecting the most suitable CNN architectures for our task. This decision-making process is based on the results presented by Sa'doun et al. (2021). The criteria for choosing the top three CNN models include:

- Performance Metrics: mAP, precision, recall, and F1 score.
- Computational Efficiency: Model complexity and inference time.
- Suitability for Insect Detection: The ability to accurately detect and classify insects in diverse and complex backgrounds.

Based on the findings from Sa'doun et al. (2021), we have excluded certain models from our consideration:

- R-FCN with ResNet 101: This model was excluded due to its performance being very similar to Faster R-CNN with ResNet101, but with slightly less precision and better time. Since both models share the same feature extractor, including both would not provide significant comparative insights.
- YOLOv2: This model was excluded in favor of YOLOv8, which was developed to achieve better precision. Precision being the prime focus of this study, the latest version of YOLOv8 was preferred.

For our thesis on object detection for insects, we have selected the following models, which are implemented using Detectron2 and Ultralytics frameworks:

- Faster R-CNN with ResNet 50 and 101 and Extended Residual Network (ResNeXt)101: These models are well-established in the object detection domain, providing a good balance between precision and computational efficiency. Additionally Faster R-CNN with X-101-32x8d offers enhanced feature extraction capabilities due to its deeper architecture, making it suitable for complex object detection tasks.
- RetinaNet with ResNet 50 and ResNet 101: it is known for its ability to handle class imbalances in object detection, which is particularly useful in detecting smaller objects such as insects.
- YOLOv8: As the latest iteration of the YOLO family, YOLOv8 incorporates several advancements in speed and precision, making it a strong candidate for real-time object detection applications.

In the following subsections, we will delve into the characteristics of each of these

models, providing a thorough analysis of their capabilities particularly in the context of detection and classification tasks.

3.2.1 Faster-RCNN

Faster R-CNN, short for Region-based Convolutional Neural Network, is a powerful object detection framework. It builds upon previous models like R-CNN and Fast R-CNN, which were designed for object detection. The key improvement in Faster R-CNN, as introduced by Ren et al. (2016), is its ability to generate region proposals (potential areas where objects may be located) directly within the network. This improves both the speed and accuracy of the detection process. It's efficient because it combines two tasks:

- Finding regions that might contain objects.
- Classifying what the objects are.

By performing both tasks in a single system, it works faster and more accurately than older methods. Figure 3.6 gives a basic overview of the Faster R-CNN structure.

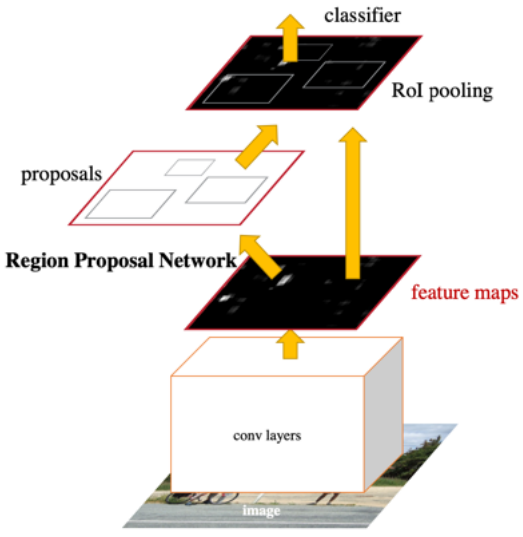

Figure 3.6: Faster R-CNN is a unified network for object detection, where a RPN after the last convolutional layer proposals to help focus on the important areas in the image, RPN trained to produce region proposals directly, adapted from Ren et al. (2016).

The following is a breakdown of Faster-RCNN components (Girshick 2015).

- Region Proposal Network: RPN is the core innovation of Faster R-CNN, enabling the model to propose regions likely to contain objects directly within the network, thus eliminating the need for external methods or searching the whole image blindly. Figure 3.7 illustrates RPN that the following explains in details.

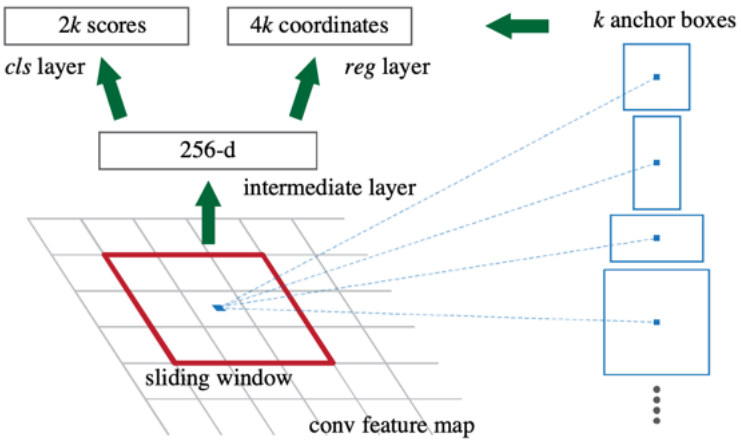

Figure 3.7: RPN uses a sliding window across the image and predicts both the objectness score and the precise location (bounding box) for each region. RPN uses k anchor boxes at each location. After that regression gives offsets from anchor boxes and classification gives the probability that each (regressed) anchor shows an object, adapted from Ren et al. (2016).

  - Sliding window: RPN looks at the image piece by piece using a small window that moves across the image.
  - Anchor Boxes: RPN uses predefined boxes (called anchor boxes) of different sizes and shapes at each location of the image to detect objects of different scales. Anchor boxes are like templates that help the RPN detect objects of different shapes and sizes. At each location in the image Several anchor boxes are tested at once. Each box is checked to see if it matches an object in that part of the image, if an anchor box matches well with an object, it is kept as a region proposal, else, it's discarded. Figure 3.8 illustrates the steps from an anchor box to a bounding box.
  - Generate Region Proposals: RPN scans the image and proposes regions

where objects may be present.

- Objectness Score: For each proposed region, the network predicts a score indicating how likely it is that the region contains an object, based on this score adjusts the box to fit better if needed.
- Bounding Box Coordinates: RPN predicts the location of the object within each proposed region.

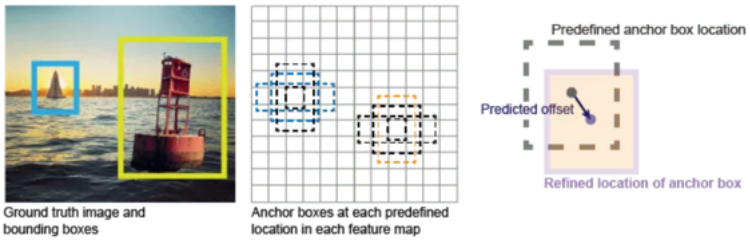

Figure 3.8: Anchor boxes are pre-defined regions with various sizes and aspect ratios that are placed over an image to detect objects. The network predicts offsets to refine these anchor boxes, adjusting their size and position to better fit the objects. These refined boxes, called bounding boxes, are the final predictions that tightly enclose objects like cars or pedestrians, adapted from Cohen (n.d.).

- ROI pooling: In Faster R-CNN, regions of interest (ROI) Pooling is a process that takes regions proposed by the RPN and converts them into a fixed size, typically 7x7. This ensures that regions of different sizes can be uniformly processed by the final classification and bounding box regression layers. ROI pooling helps preserve spatial information while ensuring that the network can handle regions of various scales and shapes consistently across the pipeline, which is critical for accurate object detection and classification, see Figure 3.9
- Fast R-CNN as detection network identifies objects after ROI Pooling. It classifies objects by determining what object is in each region (e.g., a car, a person, a dog), then refines the bounding box by adjusting the boundaries of the box around the object to ensure it fits perfectly.

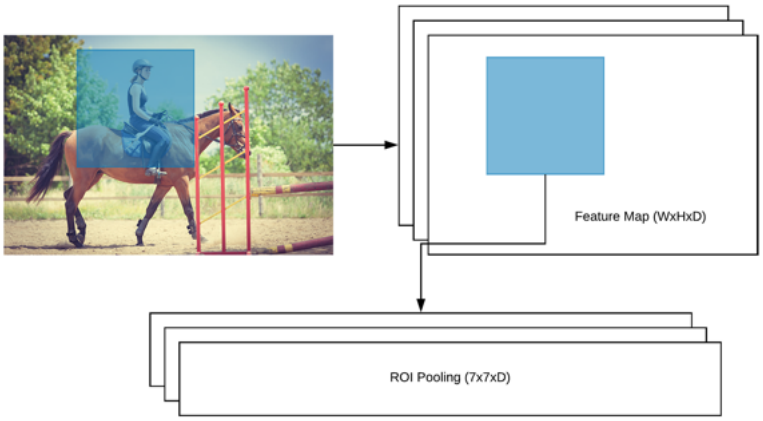

Figure 3.9: The goal of the ROI pooling module is to take all N proposal locations and then extract the corresponding ROI features from the convolutional feature map. The ROI Pooling module then resizes the dimensions of the extracted features for the ROI down to 7×7×D (where D is the depth of the feature map). This fixed size prepares the feature for the two upcoming FC layers in the next module, adapted from Martinez (2023).

The complete overview of Faster R-CNN architecture is presented in Figure 3.10

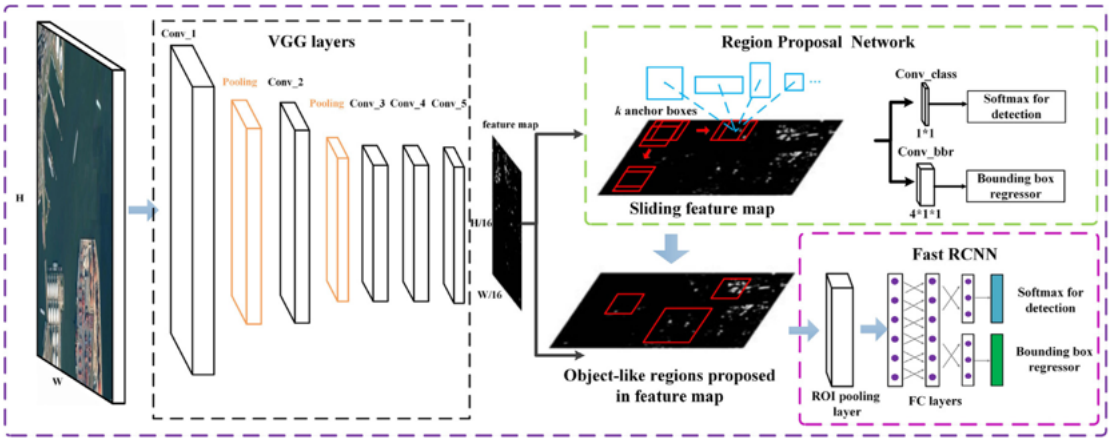

Figure 3.10: Faster R-CNN Architecture, adapted from Deng et al. (2018).

Faster R-CNN in Detectron2

Detectron2 was developed by Wu et al. (2019) as a part of Facebook AI Research. Detectron2 is an open-source platform for object detection and segmentation, built on PyTorch. It provides a modular framework, enabling easy customization of

models like Faster R-CNN. Detectron2's flexibility allows users to modify components such as the backbone network, RPN, and detection head. This makes it ideal for adapting Faster R-CNN to specific tasks or datasets.

Detectron2 supports powerful backbone networks including ResNet50, ResNet101, and ResNeXt101, often paired with FPN for improved multi-scale object detection. The RPN can be configured to adjust anchor boxes and proposals, while the detection head handles object classification and bounding box refinement.

Detectron2's support for distributed training and mixed precision training enhances efficiency, enabling faster training across multiple GPUs. It also offers pre-trained models through its Model Zoo, allowing users to fine-tune Faster R-CNN for specific applications without starting from scratch.

Advanced features like FPN further improved detection accuracy, making Detectron2 a powerful platform for both research and production (Saxena et al. 2022).

The architecture of Detectron 2 is illustrated in Figure 3.11.

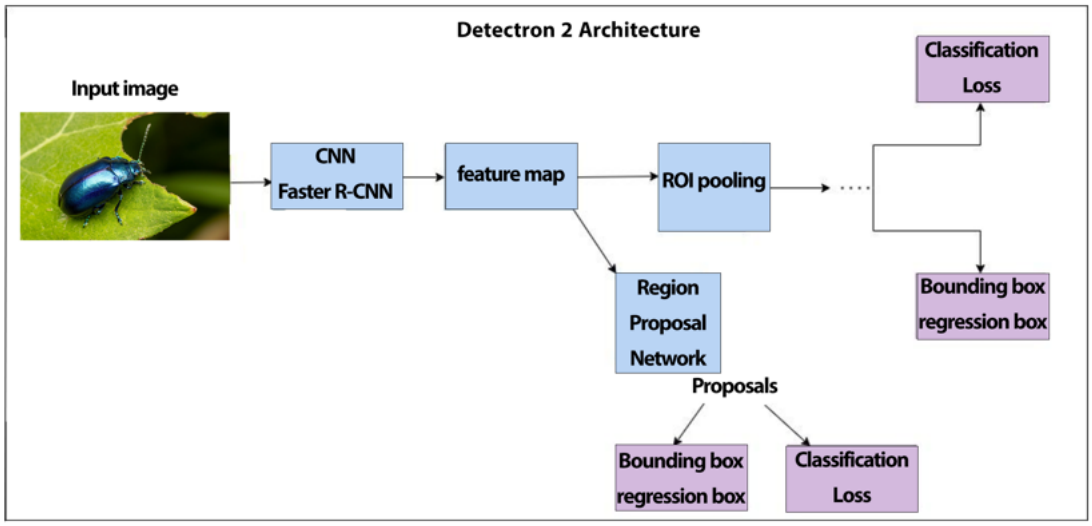

Figure 3.11: The architecture of Detectron 2 for Faster R-CNN is made up of a convolution layer that captures features of the input images and feeds them to a regional proposal network that offers probable regions of interest for class objects and a detection network that creates bounding boxes around them, adapted from Saxena et al. (2022).

3.2.2 RetinaNet by Lin, Dollar, et al. (2017)

RetinaNet is a one-stage object detection model introduced by Lin, Dollar, et al. (2017), which differs from two-stage detectors like Faster R-CNN that first generates region proposals and then classifies them. In contrast, RetinaNet directly predicts both class probabilities and bounding box coordinates in a single forward pass, making it faster while maintaining high accuracy.

A key innovation in RetinaNet is the Focal Loss function, designed to address the class imbalance problem common in object detection tasks. Focal Loss down-weights the loss contribution from well-classified examples, focusing more on hard, misclassified ones. This enables RetinaNet to handle the imbalance between foreground (objects) and background regions more effectively.

Figure 3.12 illustrates the RetinaNet architecture.

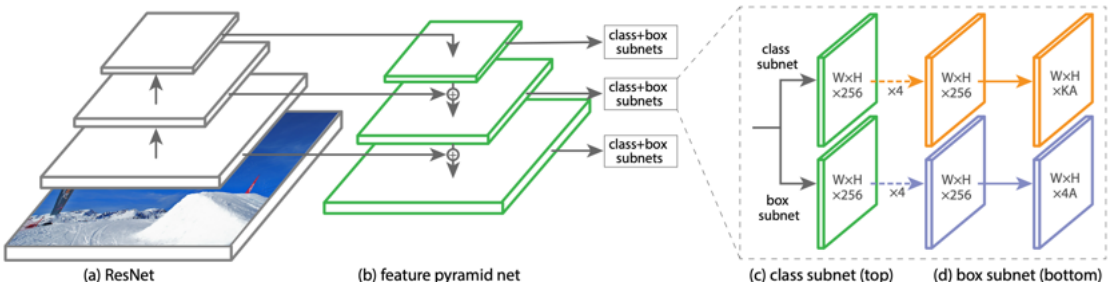

Figure 3.12: RetinaNet uses a Feature Pyramid Network FPN on top of a ResNet backbone to create a multi-scale feature pyramid. It attaches two subnetworks: one for classifying anchor boxes and another for bounding box regression. This simple design enables the model to focus on the novel Focal Loss function, which closes the accuracy gap between one-stage and two-stage detectors, adapted from Lin, Dollar, et al. (2017).

Core components of RetinaNet

RetinaNet's structure is built around a backbone network, a FPN, and two parallel subnetworks for classification and regression. Below is a breakdown of these components:

- Backbone network: The backbone network, typically a ResNet architecture (e.g., ResNet50 or ResNet101), is responsible for extracting feature maps

from the input image. These feature maps are then processed by the subsequent layers to detect objects. The backbone generates a rich set of multi-level features, which are further refined by the FPN.

- Feature pyramid network: Feature pyramid network (FPN) is layered on top of the backbone to build a multi-scale feature representation, which is crucial for detecting objects of various sizes. The FPN combines semantically strong, low-resolution features with semantically weaker, high-resolution features, allowing RetinaNet to detect both large and small objects effectively, see Figure 3.13.

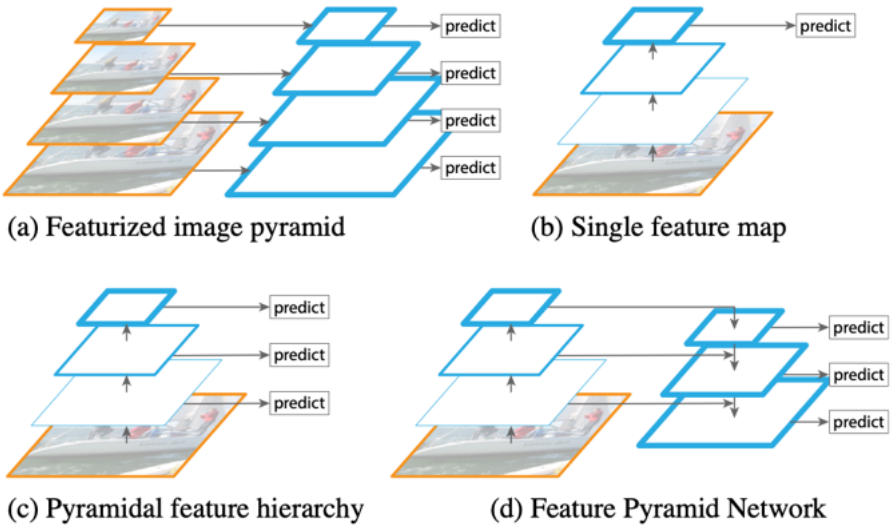

Figure 3.13: (a) Traditional methods use image pyramids to generate features at different scales, which is slow. (b) Modern detectors often use a single-scale feature map for faster detection. (c) FPN reuses the pyramidal feature hierarchy within the ConvNet to achieve a featurized image pyramid. (d) RetinaNet's FPN is both fast and accurate by utilizing these features, adapted from Lin, Dollar, et al. (2017).

- Anchors: Similar to Faster R-CNN, RetinaNet uses anchor boxes to cover different object scales and aspect ratios. These predefined anchors are spread across the image, and the model evaluates each anchor for potential object presence. Each anchor is assigned a class label and bounding box coordinates, which the model then adjusts to fit detected objects, see Figure 3.8.

- Classification and regression subnets: RetinaNet employs two parallel subnetworks:
  - Classification subnet: This subnetwork predicts the probability that an object exists at each anchor and assigns a class label (e.g., "person," "car").
  - Regression subnet: This subnetwork refines the bounding box coordinates by predicting the offset values relative to each anchor, thereby adjusting the anchors to fit the detected objects better.

  The two subnetworks allow RetinaNet to to classify objects and fine-tune the bounding box simultaneously, making it efficient for real-time object detection tasks, see Figure 3.12.

- Focal Loss:

  A major challenge in object detection is the class imbalance between foreground (objects) and background (non-objects). Standard loss functions like Cross Entropy tend to be dominated by a large number of easy, well-classified background examples. To solve this, RetinaNet introduces Focal Loss, which reduces the loss for well-classified examples and focuses more on harder-to-classify examples.

  The Focal Loss function modifies the standard Cross Entropy loss by introducing a scaling factor $(1 - p_t)^\gamma$, where (p_t) is the predicted probability for the correct class. When $(\gamma > 0)$, this factor reduces the loss contribution for well-classified examples, thereby putting more emphasis on hard, misclassified examples. Figure 3.14 illustrates the impact of Focal Loss.

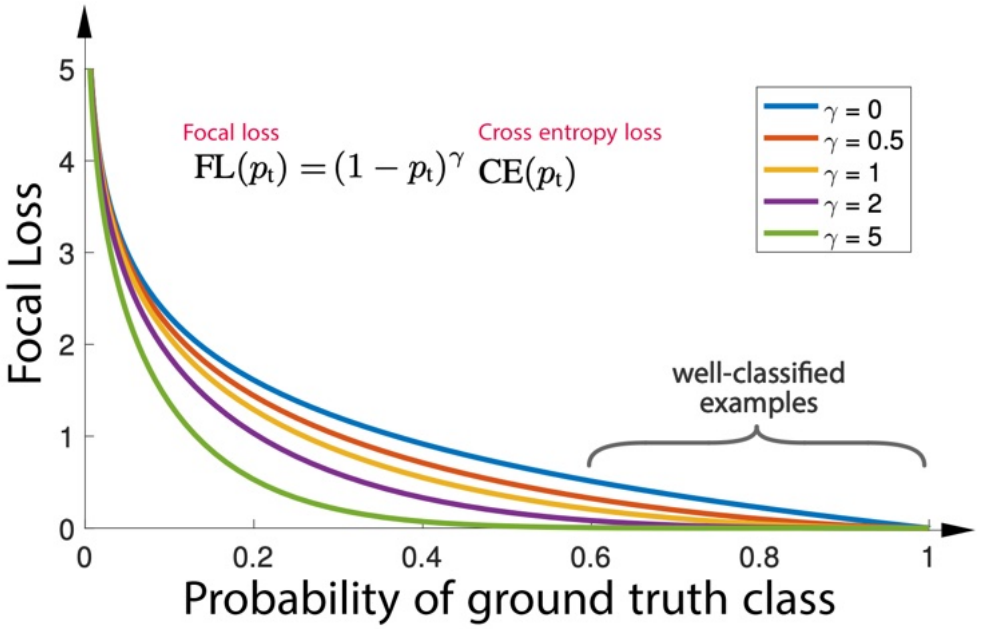

Figure 3.14: Focal Loss reduces the relative loss for well-classified examples, focusing training on hard examples. By adjusting the scaling factor $(1-p_t)^\gamma$, RetinaNet can balance the effect of class imbalance, leading to improved accuracy, adapted from Lin, Dollar, et al. (2017).

RetinaNet in Detectron2

Detectron2 (Wu et al. 2019)'s modular design allows researchers to easily implement and modify RetinaNet. The flexibility of Detectron2 enables users to swap out components like the backbone network, the FPN, or the classification and regression subnetworks. This allows RetinaNet to be adapted to specific use cases, enhancing its performance for various object detection tasks, see Figure 3.12.

3.2.3 YOLO

The YOLO (You Only Look Once) family of models represents a breakthrough in real-time object detection. Introduced by Redmon et al. (2016), YOLO transformed object detection by framing it as a single, unified task where a single neural network predicts both bounding boxes and class probabilities directly from an input image in one pass. This end-to-end approach provides YOLO with its core strength, and high speed without compromising accuracy, making it ideal for real-time applications.

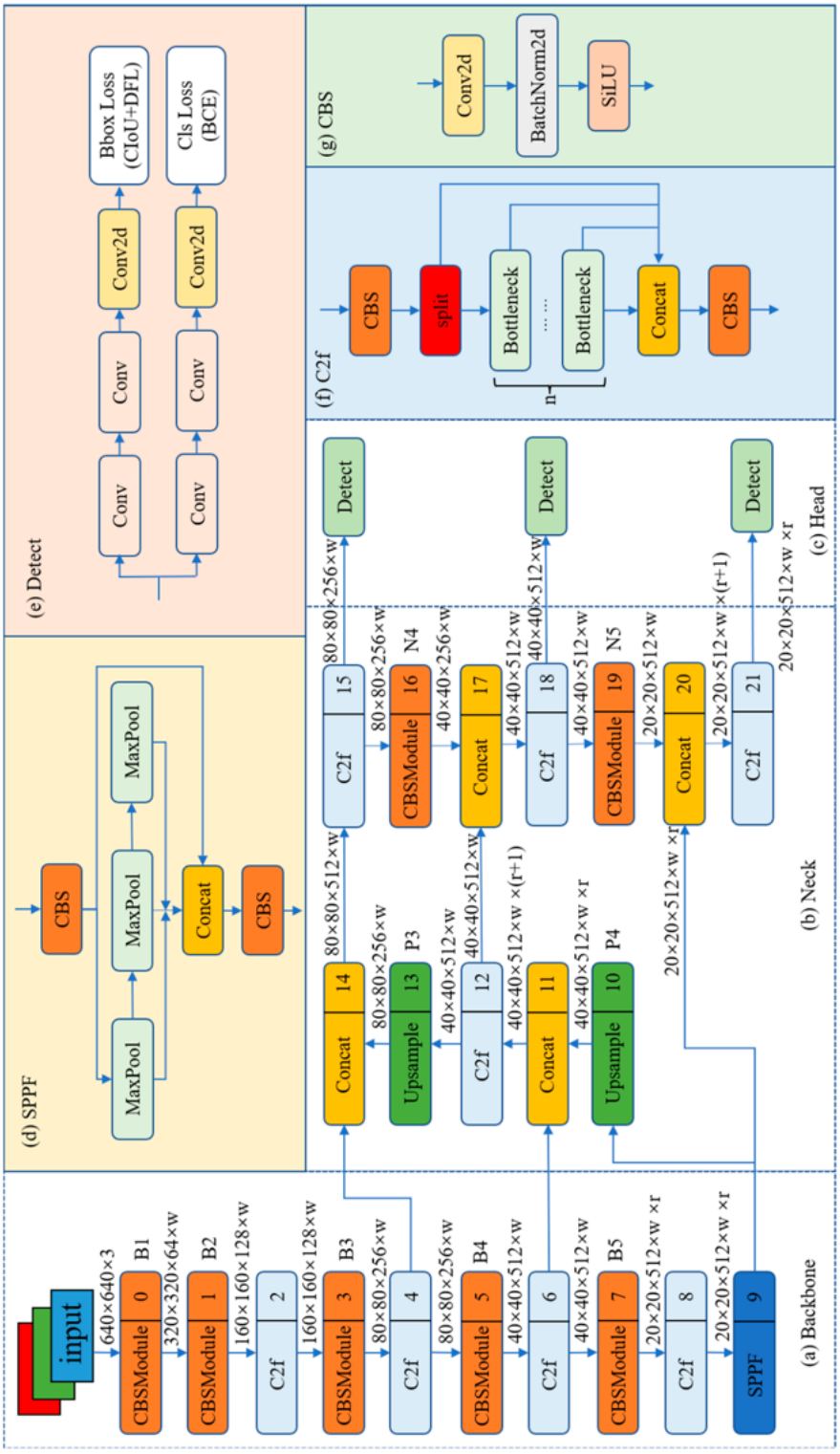

Figure 3.15: The network structure of YOLOv8. The w (width) and r (ratio) in this figure used to represent the size of the feature map. The size of the model can be controlled by setting the values of w and r to meet the needs of different application scenarios, adopted from Wang et al. (2023).

YOLO's main advantage lies in its efficiency. It treats object detection as a regression problem, simultaneously predicting bounding box coordinates and their corresponding class probabilities. Over the years, YOLO has evolved through various versions, each improving upon speed, accuracy, and feature set, making it one of the most widely used object detection models (Terven, Córdova-Esparza, and Romero-González 2023).

YOLOv8

YOLOv8 is the latest iteration of the YOLO family, developed by Ultralytics (Jocher, Chaurasia, and Qiu 2023). It builds on the strengths of earlier versions while introducing several enhancements, such as improved accuracy, speed, and ease of use. Figure3.15 provides an overview of YOLOv8's architecture.

The following is the breakdown of YOLOv8's characteristics by Wang et al. (2023)

- Backbone and neck architecture:

  YOLOv8 introduces a refined backbone architecture designed for both speed and accuracy. The CSPDarknet53 backbone, an enhanced version of the original Darknet, uses Cross-Stage Partial connections to improve gradient flow and reduce computational overhead. This architecture is further strengthened by the Cross-Stage Partial Bottleneck with Two Convolutions (C2f) module, which enhances the network's ability to capture and integrate features from various layers.

  Additionally, YOLOv8 includes a Spatial Pyramid Pooling Fast (SPPF) layer that pools features into a fixed-size map, speeding up computations and enabling the model to handle varying input sizes more efficiently.

  The model also employs decoupled heads, which independently process objectness, classification, and regression tasks. This separation allows each branch to specialize in its respective task, improving overall precision, see Figure 3.16.

- Bounding boxes and anchor boxes:

  In YOLOv8, a bounding box is a rectangle enclosing a detected object in an image, defined by its center coordinates (x, y), width, height, and a confidence score indicating the likelihood that the box contains an object, see Figure 3.8.

YOLOv8 uses an anchor-free approach with a decoupled head, which processes objectness, classification, and regression tasks independently. This anchor-free design eliminates the need for predefined anchor boxes, allowing for more flexible and accurate bounding box predictions.

The model uses a sigmoid function (Section 2.3.2) for the objectness score, representing the probability that a bounding box contains an object, and a softmax function (Section 2.3.2) for class probabilities, representing the likelihood that the object belongs to a specific class.

- Non-Maximum suppression:

NMS is a post-processing technique used to eliminate redundant bounding boxes, ensuring that each object is represented by a single, most confident box. NMS improves detection accuracy by removing duplicate boxes, making YOLOv8's predictions more precise, see Figure 3.17.

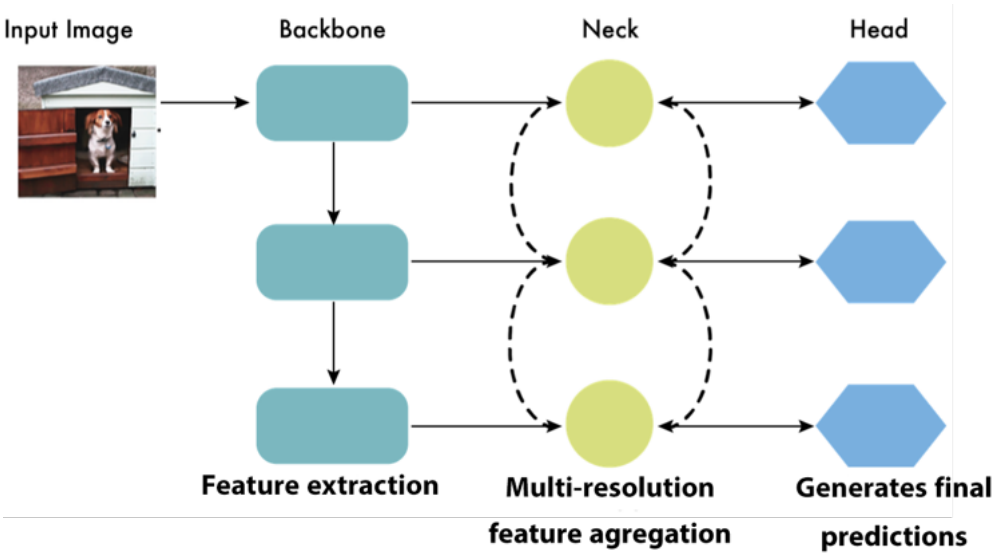

Figure 3.16: The backbone extracts vital features from the image, while the neck refines these features for object detection. The decoupled heads handle objectness, classification, and bounding box regression tasks independently, adopted from Terven, Córdova-Esparza, and Romero-González (2023).

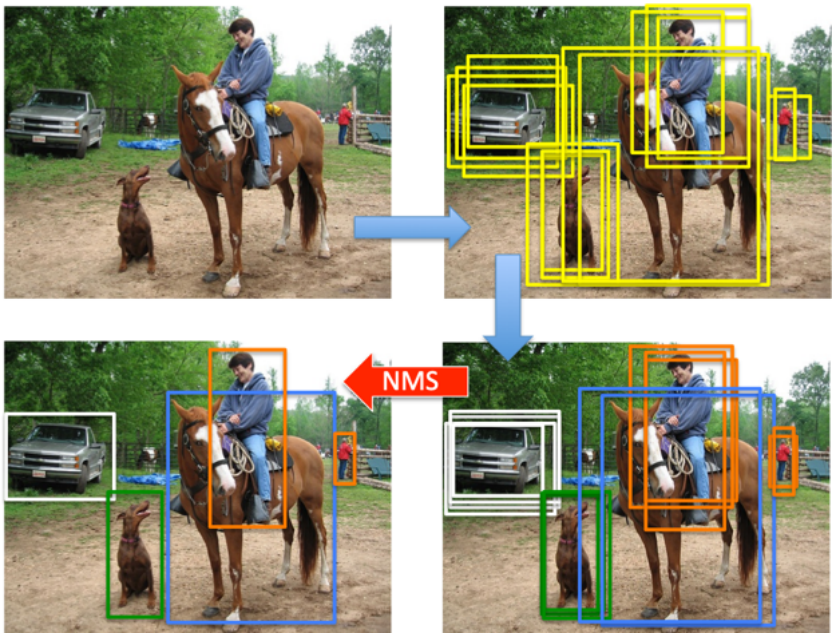

Figure 3.17: NMS eliminates redundant bounding boxes, ensuring each object is represented by one confident box, adapted from Bodla et al. (2017).

- CIoU loss function:

  YOLOv8 utilizes the CIOU loss function to enhance object detection precision. The CIOU loss consists of three components:

  - Intersection over union : Intersection Over Union (IOU) measures the overlap between the predicted and ground-truth boxes.
  - Center distance: Accounts for the squared distance between the centers of the predicted and ground-truth boxes.
  - Aspect ratio consistency: Ensures that the aspect ratios of the predicted and ground-truth boxes are consistent.

  This sophisticated loss function improves bounding box regression by evaluating overlap, center distance, and scale differences between predicted and actual boxes. This loss function is defined as Equation 3.2 and shown as Figure 3.18 (Koay et al. 2021).

$$\text{IoU} = \frac{\text{area}(B_p \cap B_{\text{gt}})}{\text{area}(B_p \cup B_{\text{gt}})} \quad \text{(Koay et al. 2021)} \tag{3.1}$$

  where:

- B_p is the predicted bounding box
- B_{gt} is the ground-truth bounding box
- $\cap$ represents the intersection of the two boxes
- $\cup$ represents the union of the two boxes

$$L_{\text{CIoU}} = 1 - \text{IoU} + \frac{\rho^2(B, B_{\text{gt}})}{c^2} + \alpha v \quad \text{(Koay et al. 2021)} \tag{3.2}$$

where:

- IoU is the intersection over union
- $\rho^2(B, B_{\text{gt}})$ is the squared distance between the center points of the predicted box B and the ground-truth box B_{gt}
- c is the diagonal length of the smallest enclosing box covering both B and B_{gt}
- α is a weighting factor, and v is the aspect ratio term

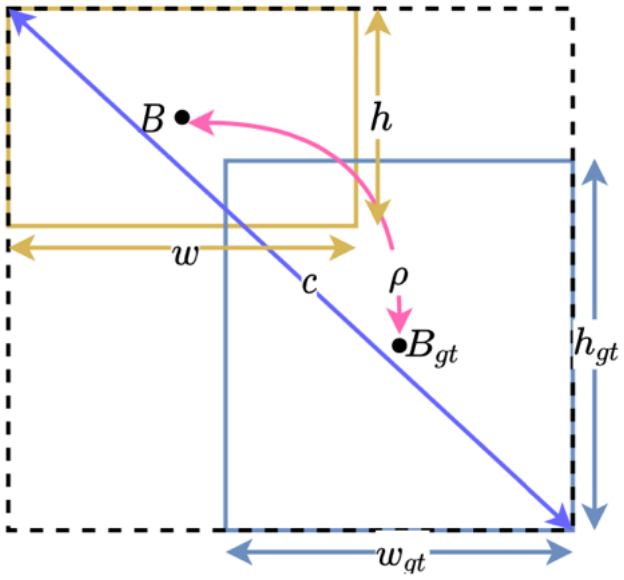

Figure 3.18: CIOU (Koay et al. 2021).

Training Techniques

YOLOv8 leverages advanced training techniques such as hyperparameter optimization, mixed precision training, and data augmentation. These techniques

contribute to faster convergence during training and help the model generalize better to unseen data (Terven, Córdova-Esparza, and Romero-González 2023).

Ultralytics and YOLOv8

Ultralytics by Jocher, Chaurasia, and Qiu (2023) is the company behind YOLO series, and with YOLOv8, they have continued their commitment to innovation in the field of computer vision. Ultralytics has not only developed this model but also offers an ecosystem of tools and services that enhance the usability and deployment of its models.

3.2.4 Summary of Selected Models

In this study, various state-of-the-art DL models are employed for insect detection across different species. The models selected include both single-stage and two-stage object detectors, each with its unique architectural features. YOLOv8, RetinaNet (ResNet 50 and 101), and Faster R-CNN (ResNet 50, 101, and ResNeXt 101) have been chosen due to their proven performance in object detection tasks. These models are evaluated not only for their ability to detect insects of varying sizes but also for their computational efficiency and accuracy.

Table 3.1 summarizes the key properties and architectures of the selected models, which are analyzed and compared in this chapter and include:

- Model type: Defines whether the model is single-stage (like YOLO and RetinaNet) or two-stage (like Faster R-CNN).
- Backbone Architecture: The feature extraction backbone used by each model.
- Anchor Type: Defines whether the model uses anchors or is anchor-free (like YOLOv8n).
- Head Type: Explains how object detection and bounding box regression tasks are handled (e.g., decoupled head for YOLOv8, ROI Pooling for Faster R-CNN).
- Feature Pyramid: Describes the use of glsfpn for multi-scale feature extraction, which is key for detecting objects of different sizes.
- Loss Function: The loss function is used for object detection and bounding box regression, which is critical for optimizing model performance.

3.2.5 Data Preprocessing

The third phase of our methodology focuses on data preprocessing, a critical step in ensuring that our dataset is properly prepared for effective model training. This phase involves several key processes aimed at transforming raw annotations into formats that are compatible with the selected object detection models:

- Conversion of LabelBox JSON annotations: Annotations provided by LabelBox are converted into the appropriate formats for the different models. Specifically, we convert them to the COCO format for RetinaNet and Faster R-CNN, and the YOLO format for YOLOv8.
- Image segmentation: Each image in the dataset is divided into smaller sub-images. This step reduces the complexity of individual images, making it easier for the models to detect smaller objects and improving overall detection accuracy.
- Label Adjustment: The labels corresponding to each image are adjusted to reflect the objects present in the newly segmented sub-images, ensuring consistency and accuracy across the entire dataset.
- Training Set Construction: A portion of the dataset is allocated for training, ensuring the models are exposed to diverse variations in the data during the learning process, which is crucial for improving generalization and detection performance.

3.2.6 Implementation of the Experiment

The fourth phase involves the implementation of our CNN models, where both the software and hardware are configured and optimized to ensure seamless and efficient execution. This phase includes the following components:

- **Hardware setup**:
  - The models are trained on a system equipped with a Quadro P1000 GPU with 4096MiB of memory, utilizing CUDA for acceleration.
  - CUDA 11.7 is configured to ensure compatibility with the DL frameworks, enabling efficient use of GPU resources.
- **Software stack**:

- Python 3.11.8 is used as the programming environment, ensuring compatibility with the latest ML and DL libraries.
- The following major libraries are used for the model training and evaluation:
  * Ultralytics YOLOv8.2.31: The YOLOv8n framework is employed for real-time object detection, utilizing torch 2.0.0+cu117 for training on GPU.
  * Detectron2: Detectron2 is used for training and testing RetinaNet and Faster R-CNN models, providing state-of-the-art tools for object detection. The Detectron2 configuration is set up using model zoo configurations from the library.
  * TensorFlow and Keras: These libraries are used for backend processing and loss functions, ensuring compatibility with both the YOLOv8 and Detectron2 frameworks.

3.2.7 Evaluation

This phase outlines the evaluation criteria and metrics used to assess the performance of the object detection models developed in this study. Given the challenges of detecting insects in complex natural environments, a comprehensive set of evaluation metrics has been selected to ensure a thorough and nuanced assessment of model performance.

Evaluation Metrics:

To capture different aspects of model performance, we focus on several core metrics:

- Confusion Matrix for Object Classes: A confusion matrix will be generated to visualize the model's performance across different object classes, such as different insect species or categories of objects. This matrix will highlight where the model may struggle with misclassification, providing insight into how often one class is confused with another, especially in cases where insects are confused with the background (Sa'doun et al. 2021).

  For each ground truth label, the Intersection over Union (IoU) with the corresponding prediction labels is calculated. In our study, we consider the highest IoU between 50% and 95% for matching predictions. The ground

truth label is assigned to the prediction label with the highest IoU, ensuring that the best possible match is made between the predicted and actual objects.

Below is an explanation of the confusion matrix used in our study. Figure 3.19 provides a visual representation of the matrix:

- Diagonal Values (Green Cells): Represent successful detection and correct classification of objects. High values along the diagonal indicate that the model is accurately identifying and classifying objects.
- Grey Cells: Indicate cases where a ground truth label has a maximum IoU of 0, meaning that the object was not detected by the model. This highlights potential gaps in the model's ability to detect certain objects.
- Orange Cells: Represent false positives—cases where a prediction label has a maximum IoU of 0, meaning it was incorrectly labeled as a target object when it should have been classified as a non-object or a different class.
- White Cells: Represent cases where the model successfully detected an object but misclassified it. These highlight areas where the model struggles to differentiate between similar classes, suggesting the need for additional training or data refinement.

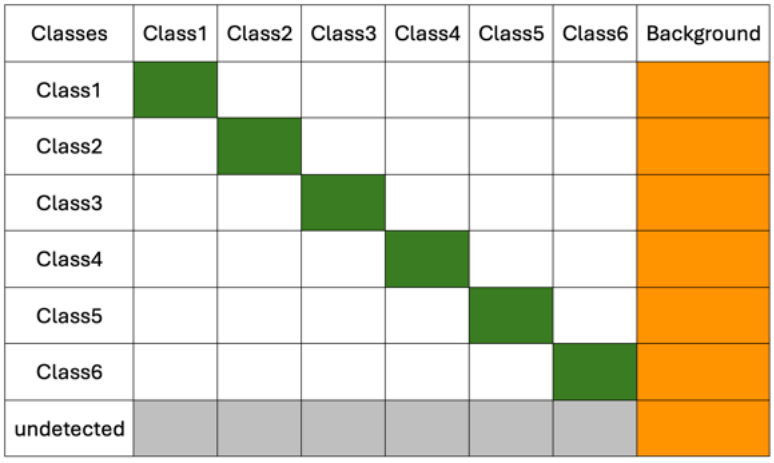

Figure 3.19: Example of confusion matrix.

The confusion matrix is computed by comparing the predicted class with the actual class for each detection (Saxena et al. 2022). By analyzing the

confusion matrix, we gain valuable insights into the model's classification performance, identifying strengths and areas for improvement. For each class:

True Positives (TP) = Correctly predicted objects for class

False Positives (FP) = Objects wrongly predicted as a class

False Negatives (FN) = Objects that were not detected by the model

Precision, recall, and F1 scores are then derived from the confusion matrix:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad \text{(Saxena et al. 2022)} \tag{3.3}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad \text{(Saxena et al. 2022)} \tag{3.4}$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad \text{(Saxena et al. 2022)} \tag{3.5}$$

- Mean Average Precision by Wang (2022): The COCO-style mAP will be employed, which calculates the average precision across multiple IOU thresholds ranging from 0.50 to 0.95 in increments of 0.05. This metric provides a holistic view of models performance across different precision levels and offers insight into how well the model handles varying degrees of overlap between predicted and ground-truth bounding boxes (Lin, Maire, et al. 2014). mAP two main thresholds are:
  - mAP@50 refers to calculating Average Precision at a fixed IOU threshold of 0.50, meaning an object is considered correctly detected if the overlap between the predicted and ground-truth bounding boxes is at least 50%.
  - mAP@50:95 calculates Average Precision (AP) across several IOU thresholds ranging from 0.50 to 0.95 in steps of 0.05, providing a more comprehensive performance measure by testing the model's robustness across stricter conditions.

    The computation of mAP@50:95 involves the following steps:

    * Precision-Recall curve: Plot a precision-recall curve based on the model's predictions for each object class, see Figure 3.20
    * IoU Thresholds: IOU thresholds range from 0.50 to 0.95 in 0.05

steps.

* AP: Computate the average precision, which is the area under the precision-recall curve, for each IOU threshold.
* mAP: Calculate the mean of all AP values across IOU thresholds (from 0.50 to 0.95), see Equation 3.7

For a dataset that contains k classes of objects, the mAP of a detector at an IOU threshold (t) is defined in Equation 3.6.

$$\text{mAP@t} = \frac{1}{k}\sum_{k=1}^{K}\text{AP@t} \quad \text{(Wang 2022)} \tag{3.6}$$

$$\text{mAP}_{50:95} = \frac{1}{10}\sum_{i=0}^{9}\text{mAP}_{0.50+0.05i} \quad \text{(Wang 2022)} \tag{3.7}$$

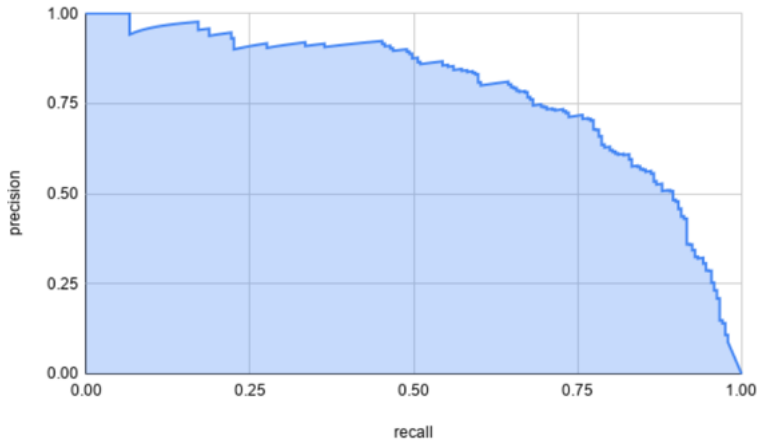

Figure 3.20: An Example of precision-recall curve of Faster R-CNN on object detection. The shaded area under the curve is equal to the AP, which ranges from 0 to 1, adopted from Wang (2022).

- F1 Score: The F1 Score, which is the harmonic mean of precision and recall, will be used as a unified metric to balance these two aspects. This is especially useful for models where a trade-off between precision (minimizing false positives) and recall (detecting all relevant objects) is necessary (Yang 1999). Components of F1 score (Equation 3.5):
  - Precision: Measures the proportion of correctly identified objects out of

all objects identified by the model, see Equation 3.3. A high precision score indicates the model's ability to minimize false positives, which is particularly important in noisy environments where background clutter could lead to incorrect detections (Everingham et al. 2010).

– Recall: Measures the proportion of actual objects correctly identified by the model, see Equation 3.4 High recall indicates the model's ability to detect the most relevant objects in the dataset, even in the presence of background noise (Everingham et al. 2010).

Due to the class imbalance in our dataset, where some species (such as *Diptera* with 330 instances) are more represented than others (like *Hemiptera* with only 20 instances), we have recalculated the **(mAP@50:95)** metrics using a **(mAP@50:95)** approach. This recalibration ensures that the performance evaluation accurately represents all classes, regardless of their frequency in the dataset.

The weighting is determined by the number of instances per class. The formula for **Weighted(mAP@50:95)** is as follows:

$$\text{Weighted mAP} = \sum_{i=1}^{n} w_i \times \text{mAP}_i \quad \text{adapted from (Phan and Yamamoto 2020)} \tag{3.8}$$

Where:

- w_i represents the weight for class i, calculated as the proportion of instances of class i relative to the total number of instances across all classes.
- mAP_i denotes the mAP score for class i.

This method will be used to calculate the F1 score similar to Equation 3.8. It helps to reduce the bias caused by the more common classes and provides a thorough evaluation of the model's performance across all species. This ensures that the accuracy of detecting rare species, like *Chelicerata* and *Lepidoptera*, is given the same importance as other species.

This set of evaluation metrics offers a comprehensive assessment of the models' capabilities, addressing aspects such as localization accuracy, classification precision, and performance across object scales.

Table 3.1: Summary of selected insect detection models

Model	**Type**	**Backbone Architecture**	**Anchor Type**	**Head Type**	**Feature Pyramid**	**Loss Function**
YOLOv8n	Single-stage	CSPDarknet53	Anchor-free	Decoupled Head	SPPF	CIoU Loss
RetinaNet (ResNet 50)	Single-stage	ResNet 50	Anchors	Decoupled Head	FPN	Focal Loss
RetinaNet (ResNet 101)	Single-stage	ResNet 101	Anchors	Decoupled Head	FPN	Focal Loss
Faster R-CNN (ResNet 50)	Two-stage	ResNet 50	Anchors	ROI Pooling + Fully Connected	FPN	Cross-entropy + BBox Regression
Faster R-CNN (ResNet 101)	Two-stage	ResNet 101	Anchors	ROI Pooling + Fully Connected	FPN	Cross-entropy + BBox Regression
Faster R-CNN (ResNeXt 101)	Two-stage	ResNeXt 101	Anchors	ROI Pooling + Fully Connected	FPN	Cross-entropy + BBox Regression

CHAPTER

4 Implementation

In this chapter, we present the two key steps involved in the implementation of our object detection system for insect detection. The first step focuses on the Preprocessing Phase, where we discuss the necessary adjustments and modifications made to the dataset to ensure it is appropriately structured for training. This includes converting annotation formats, segmenting images, and adjusting labels to align with the requirements of the selected CNN models.

The second step covers the Training Phase, where we describe the software and hardware setup utilized to train the models. This section includes details on the installed libraries, configured environments, and hardware optimizations performed to support the computational demands of deep learning. We also present sample image results to illustrate the output of the trained models. Finally, we explain how the performance of each CNN model is numerically evaluated using relevant metrics, ensuring a comprehensive assessment of their effectiveness in detecting insects in noisy backgrounds.

4.1 Data Preprocessing

After removing blurry or inadequate images of insects, our final dataset for the experiment consisted of 1303 high-resolution images (4608 × 3456 pixels, 4608 × 2592 pixels). We used 1108 images for training and 195 images for validation. The distribution of the 6 different species targeted in data acquisition is displayed in Table 4.1, revealing a highly unbalanced dataset, resulting in a low chance of detecting smaller species such as Hemiptera, see Table 4.1 and Figure 4.1 for the statistical description of our insect dataset.

Table 4.1: Distribution and size of insect species in the dataset

Specie	**Instances**	**Percentage Share (%)**	**Real size (mm)**
Lepidoptera	134	4.95	50-55
Coleoptera	182	6.73	25-32
Diptera	1836	67.91	12-45
Hymenoptera	251	9.28	20-35
Chelicerata	216	7.99	20-30
Hemiptera	85	3.14	14-19
Total	2704	100.0	

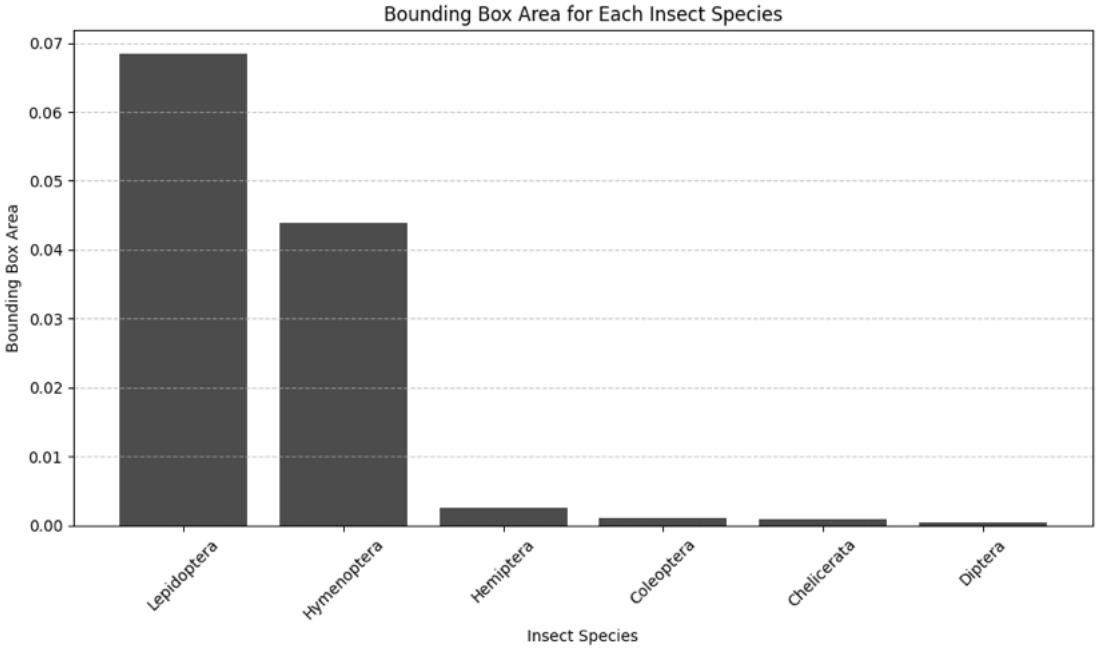

Figure 4.1: Size ratio of bounding boxes for different species

Figure 4.1 illustrates the variation in size and bounding box shapes among the different species, which poses a challenge for scaling. The diversity of species is further highlighted in Figure 4.2, encompassing Chelicerata, Coleoptera, Diptera, Hemiptera, Hymenoptera, and Lepidoptera. Figure 4.2 provides an overview of all species classes.

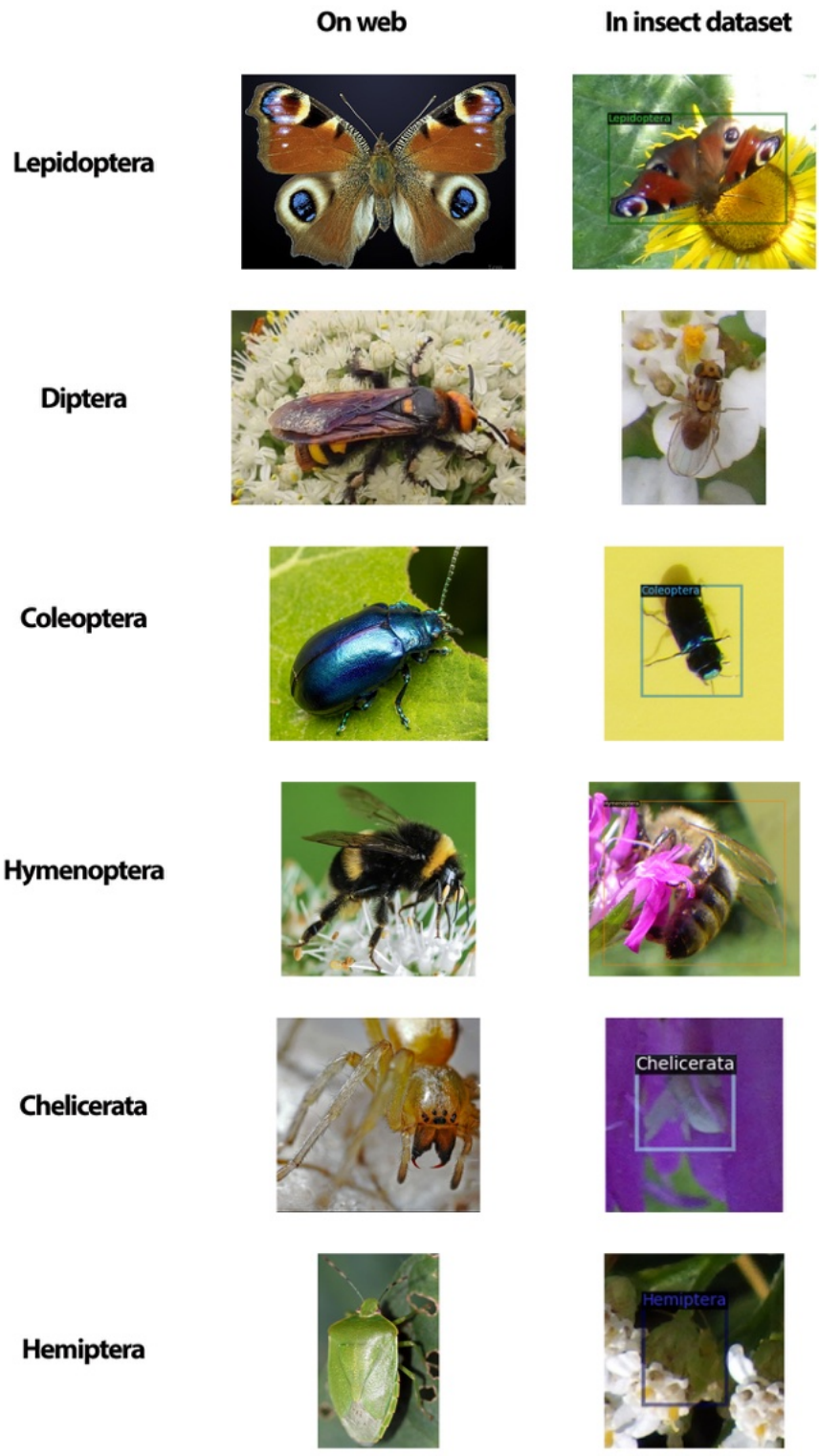

Figure 4.2: All species in the insect dataset are represented, with the first column of images sourced from NatureNet (1995) for enhanced visualization.

Data Preprocessing Methods

We decided to take distinct data preprocessing methods to have a better idea of the effectiveness of our models

- First approach: Cropping with a 70% threshold around the bounding box

  In this approach, we focused on reducing background noise by cropping the images tightly around the bounding boxes of the insects. Specifically, we applied a 70% threshold around each bounding box, which allowed us to maintain a buffer zone around the insect while minimizing the inclusion of extraneous background elements. This method was designed to help the model focus more precisely on the target object, thereby improving detection precision.

- Second approach: Standardized cropping and augmentation

  Here, we implemented a more standardized cropping method, where each image was cropped around the bounding box but resized to a uniform dimension of 1200 $\times$ 1200 pixels. This consistent image size ensures that the model processes all inputs on a similar scale, potentially improving the learning efficiency. Additionally, to address the class imbalance, we applied data augmentation techniques, generating variations of each image in four different orientations. This augmentation strategy not only increases the dataset size but also helps the model generalize better by exposing it to a wider variety of image perspectives and conditions.

After completing these preprocessing steps, we built our dataset, dividing it into training and validation. Since our data was cropped around the bounding boxes, an approach that creates an idealized dataset focused on the objects of interest, we wanted to ensure that our test set better reflected real-world conditions; Therefore, we used the original, unaltered images for the validation set as the test set. This approach allowed us to evaluate the model's performance not only under the optimized conditions of the training but also in scenarios that more closely resemble the complexity and variability of real-world images.

4.2 Training Process of CNN Models

The detailed configuration for implementing the selected CNN models is provided in Section 3.2.6. Additionally, we conducted two distinct training sessions, each aimed at addressing different levels of detection complexity.

- General Insect-level training: This focused on distinguishing insects from non-insect objects, providing a broader categorization.
- Species-level training: This involved classifying insects into six specific species: Chelicerata, Coleoptera, Diptera, Hemiptera, Hymenoptera, and Lepidoptera.

4.3 Inference Process of CNN Models

After completing the training phase for both general insect-level detection and species-level classification, the models are tested using the inference phase to evaluate their applicability. The goal of inference is to determine how well the models can generalize when presented with unseen data that was not used during training.

Inference Process

- Input data: During inference, new images from the validation set are passed into the trained CNN models. These images are preprocessed the same as in the training phase, but no labels are provided, as the model will predict the classes.
- Forward Pass: The trained model processes the input images through its convolutional layers and other learned parameters such as weights. During this phase, no backpropagation or weight updates occur, as the model is simply applying its learned features to make predictions.
- Detection and Classification: For the General Insect-Level model, the output is a binary prediction indicating whether an object is an insect or a non-insect object. For the Species-Level model, the output includes multi-class predictions, classifying detected insects into one of the six species: Chelicerata, Coleoptera, Diptera, Hemiptera, Hymenoptera, and Lepidoptera. Bounding Box Predictions: The models generate bounding boxes around detected insects, marking their position in the images. For each bounding box, the model provides, a predicted class label, and the confidence score indicating the likelihood that the object within the box is of the predicted class.

4.3.1 YOLOv8 Training and Inference

For the training process, we utilized the Ultralytics YOLOv8 repository (Jocher, Chaurasia, and Qiu 2023). The training was conducted on insect images across two classification levels: individual insects and species. To standardize the input

data, all images in configuration related to training were resized to a resolution of 800 × 800 pixels, striking a balance between preserving detailed features for accurate detection and managing computational efficiency.

As explained in Section 4.1, to enhance the model's robustness and generalization capabilities, we employed two distinct preprocessing approaches. The first approach involved cropping with a 70% threshold around the bounding boxes to reduce background noise while retaining a buffer zone around the target insects. The second approach utilized standardized cropping, where each image was resized to a uniform dimension of 1200 × 1200 pixels, combined with data augmentation techniques to address the class imbalance and improve the model's ability to generalize across different orientations and perspectives.

The training process involves determining the optimal number of epochs using the elbow method. As shown in Figure 4.9, the optimal number of epochs for YOLOv8 training appears to be in the range of 150 to 200 epochs. Beyond this range, further training may not yield significantly. The number of epochs was adjusted based on the detection complexity at each level.

Throughout the training process, the model aimed to minimize YOLOv8's loss functions, including box loss, class loss with a focus on mAP across different IOU thresholds (mAP@0.5:0.95).

Considering the challenges of detecting and classifying various insect species, we employed two distinct approaches in our training strategy for species level detection:

- NON-trained weights: training from scratch on insect-level weights. In this approach, we initially trained the model on the general insect-level dataset, focusing on detecting insects without differentiating between species. After completing this phase, we transferred the learned weights and fine-tuned the model on the species-level classification task. This approach leverages the initial training phase to develop a strong foundational understanding of general insect features, which can then be specialized for species classification.
- Trained weights: training solely on pre-trained weights from the repository. We also trained the model directly on the species-level classification task using pre-trained weights provided by the Detectron2 repository. This approach allows us to evaluate the effectiveness of the repository's pre-trained weights in directly classifying specific insect species without prior training on general insect detection.

The main point of this technique is to check each of the selected CNN models' robustness for diffrent conditions regarding the learning weights. In the inference process, the output generated by YOLOv8 results in two key components. The first is a set of labeled images with bounding boxes indicating detected objects. An example of YOLOv8's detection results at the insect level and species level are shown in Figures 4.3 and 4.4. The second component is an Excel sheet containing detailed information about each epoch, including (mAP@50:95) and the confidence score of how well each species is detected during inference on the validation set. This confidence value quantifies the model's certainty in its predictions, providing a measure of reliability for each detected object.

For YOLOv8, this comprehensive evaluation considers not only the mAP scores but also precision-recall curves and confusion matrices, providing insights into the model's ability to accurately detect and classify insects and their species.

Figure 4.3: Sample of YOLOv8 results for insect-level detection, illustrating the model's ability to accurately identify insects in images.

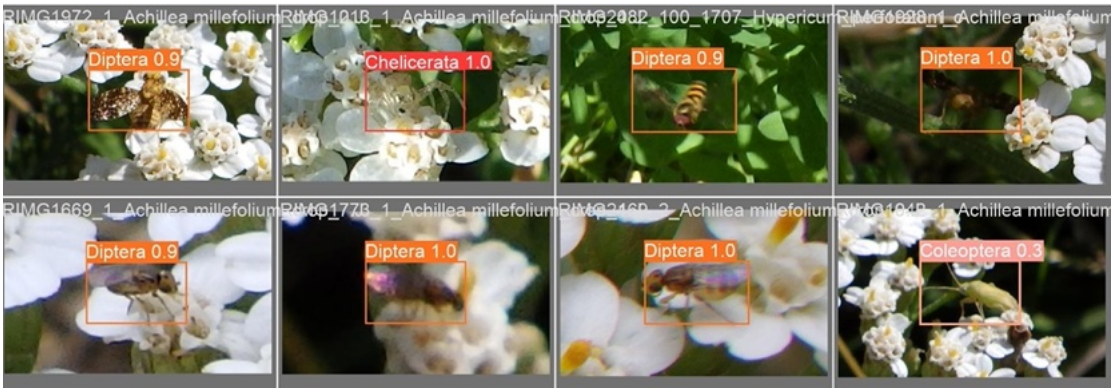

Figure 4.4: Sample of YOLOv8 results for species-level detection, demonstrating the model's capacity to classify each category among others.

4.3.2 RetinaNet Training and Inference

The RetinaNet model was trained using the well-established Detectron2 repository by Wu et al. (2019), utilizing different ResNet backbones (ResNet-50 and ResNet-101). For each detection level, including both species and general insect detection, the model is trained for 3000 epochs. Figure 4.10 illustrates the optimized number of epochs for training, based on the loss function. The training was performed using the predefined configurations from the repository, with input images resized to 800 $\times$ 800 pixels. The only adjustment made was modifying the number of classes for insects and species at each level.

The choice of ResNet backbone plays a critical role in the model's performance. ResNet-50, being lighter with fewer layers, tends to train faster and requires less computational power, making it suitable for scenarios with limited resources. On the other hand, ResNet-101, with its deeper architecture, provides more capacity for learning complex features, which can be beneficial for distinguishing between subtle differences in species-level detection. It must be considered boosting confidence for challenging detections can enhance mAP, especially in scenarios with imbalanced classes, but this does not necessarily translate to an overall improvement in precision. For example, mAP@0.5 (IoU=0.5) might show significant gains due to improved handling of small or hard-to-detect objects. Still, mAP@0.75 or higher thresholds might not always reflect similar improvements if the boosted confidence leads to overfitting on specific hard examples.

Additionally, the model's emphasis on harder examples can increase false positives, particularly in complex backgrounds where non-insect objects, such as shadows or

plant structures resemble insects. This could introduce noise into the predictions, slightly reducing the mAP score, particularly affecting the (mAP@50:95) values, which averages precision across a range of IOU thresholds.

The inference phase is as mentioned for YOLO happens for trained weights and non-trained weights. By utilizing ResNet-50 and ResNet-101 backbones in these situations, we aimed to understand the trade-offs between model capacity and training efficiency. The comparison between these approaches and backbones offers insights into whether a two-stage training process or direct fine-tuning from pre-trained weights yields better performance in terms of mAP and overall precision.

An example of RetinaNet's detection results at the insect level and next to it, species level are shown in Figures 4.5 and 4.6.

Figure 4.5: Sample of RetinaNet results for insect-level detection, illustrating the model's ability to accurately identify insects in images.

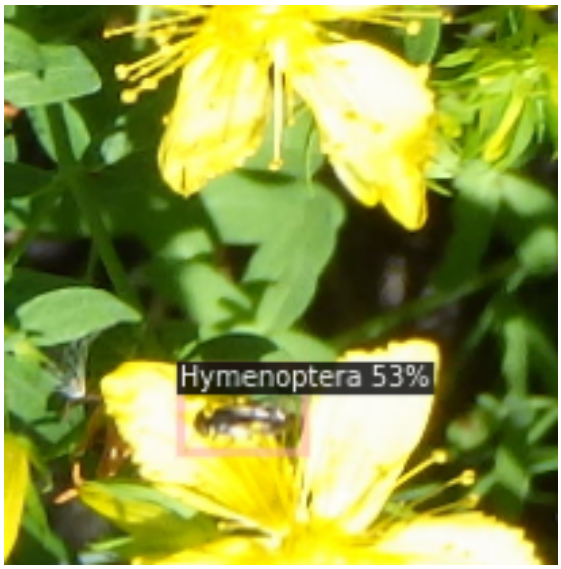

Figure 4.6: Sample of RetinaNet results for species-level detection, demonstrating the model's capacity to classify each category among others.

4.3.3 Faster R-CNN Training and Inference

The Faster R-CNN model the same as RetinaNet, is also trained using the well-established Detectron2 by Wu et al. (2019) repository with different ResNet backbones (ResNet-50, ResNet-101, and ResNeXt-101) For each classification level, including both species and general insect detection, the model was trained over 3000 epochs, see Figure 4.11 for the optimized number of epochs for training based on the loss function. The training utilized the predefined configurations from the repository, with input images resized to 800 × 800 pixels. The only modifications made were adjusting the class counts to match our dataset at both the insect level and species level.

In our implementation, we experimented with several backbone networks to enhance the feature extraction capabilities of Faster R-CNN:

- ResNet-50: A relatively lightweight model with 50 layers, offering a good balance between performance and computational efficiency.
- ResNet-101: A deeper version with 101 layers, providing greater capacity for learning complex features, which can be beneficial for more challenging detection tasks.

- ResNeXt-101: An extension of ResNet-101, incorporating a split-transform-merge strategy to further, improve model performance by enhancing feature representation and processing efficiency.

The choice of backbone significantly impacts the model's ability to capture intricate details, especially when dealing with diverse and complex datasets such as the one used in this study. The training and inference phase is also the same as the two other models for Faster R-CNN. We evaluated the output of the Faster R-CNN model using key metrics like mAP across various IOU thresholds and confusion matrices. These evaluations provide valuable insights into the model's strengths and areas for improvement, especially in handling class imbalance and complex background scenarios. By integrating these advanced techniques and tools, our implementation of Faster R-CNN offers a robust solution for object detection, capable of handling the diverse challenges presented by our dataset. An example of Faster R-CNN detection results at the insect level and species level are shown in Figures 4.7 and 4.8.

Figure 4.7: Sample of Faster R-CNN results for insect-level detection, illustrating the model's ability to accurately identify insects in images.

Figure 4.8: Sample of Faster R-CNN results for species-level detection, demonstrating the model's capacity to classify each category among others.

For a more comprehensive comparison, the training phase and inference phase (also known as Validation) loss functions for YOLO, RetinaNet, and Faster R-CNN are illustrated in the following Figure 4.9 to Figure 4.11.

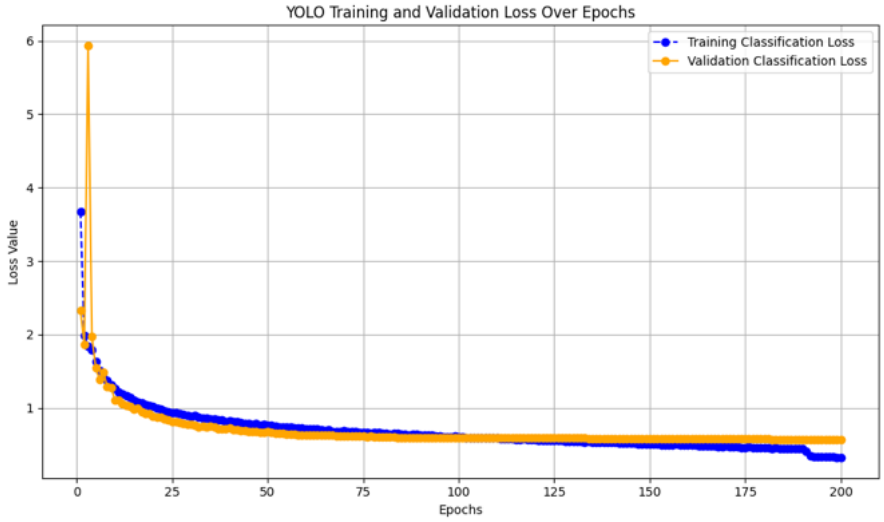

Figure 4.9: YOLO loss function over epochs, indicating that the model's learning plateaus after 150 to 200 epochs

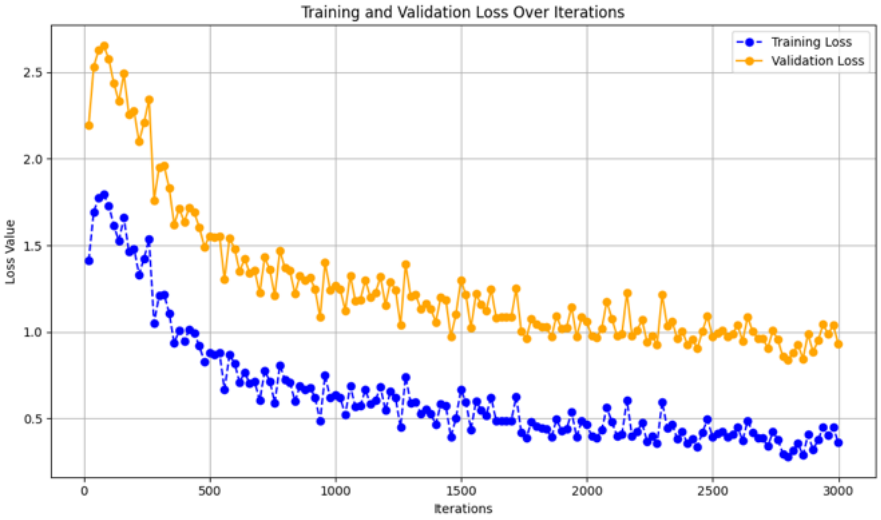

Figure 4.10: RetinaNet Loss function over epochs, indicating that the model's learning plateaus are 3000 epochs.

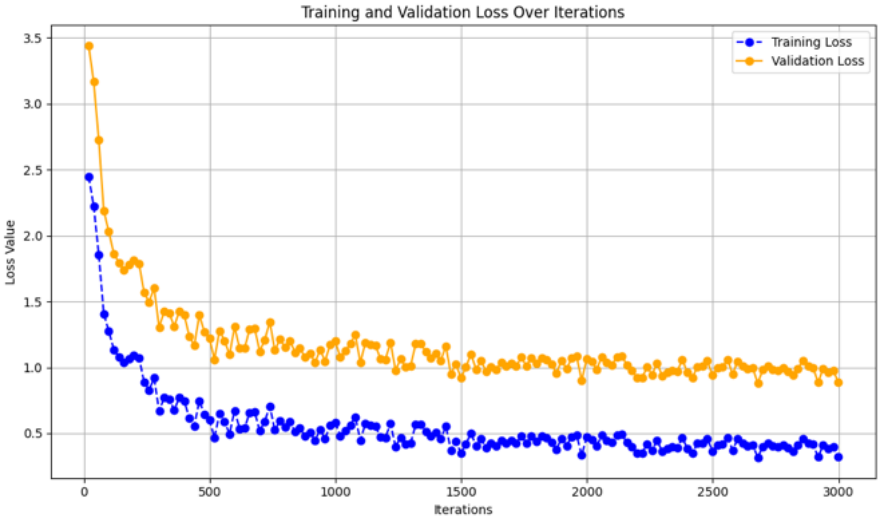

Figure 4.11: Faster R-CNN Loss function over epochs, indicating that the model's learning plateaus are 3000 epochs.

CHAPTER 5 Results

In this chapter, we evaluate the performance of various CNN in detecting and classifying insects at both the insect level and species level. To ensure robust evaluation, we employ cross-validation, where each predicted label is compared to the corresponding ground truth label, and the results are summarized in confusion matrices.

5.1 Count Performance Assessment

The first assessment focuses on the insect count performance. Insect count is defined as the number of predicted labels that correspond to labeled insects in the validation set. This assessment is performed across all levels (insect and species level) to gain insights into how the counting performance is influenced by the number of classes.

It is important to note that the number of detection labels generated by the models does not necessarily match the number of ground truth labels in the validation set. A ground truth label may be undetected, and additional labels may be generated that do not correspond to a target insect. Therefore, the results are assessed in terms of Insect Labels, Undetected Insects, and Non-Insect Detected instances. Figure 5.1 illustrates the percentage of predicted labels based on the ground truth labels in insect levels. It shows "FasterRCNN_Approach1_ResNet101", "Retinanet_Approach1_ResNet50" and "YOLO_Approach1" outperform other methods regarding predicting correct labels in insect detection for the validation part of the process. Figure 5.2 presents the percentage of predicted labels based on the ground truth labels for all models and methods used. The illustration indicates that YOLO_Approach1 is better at detecting insects at the species level compared to the other approaches.

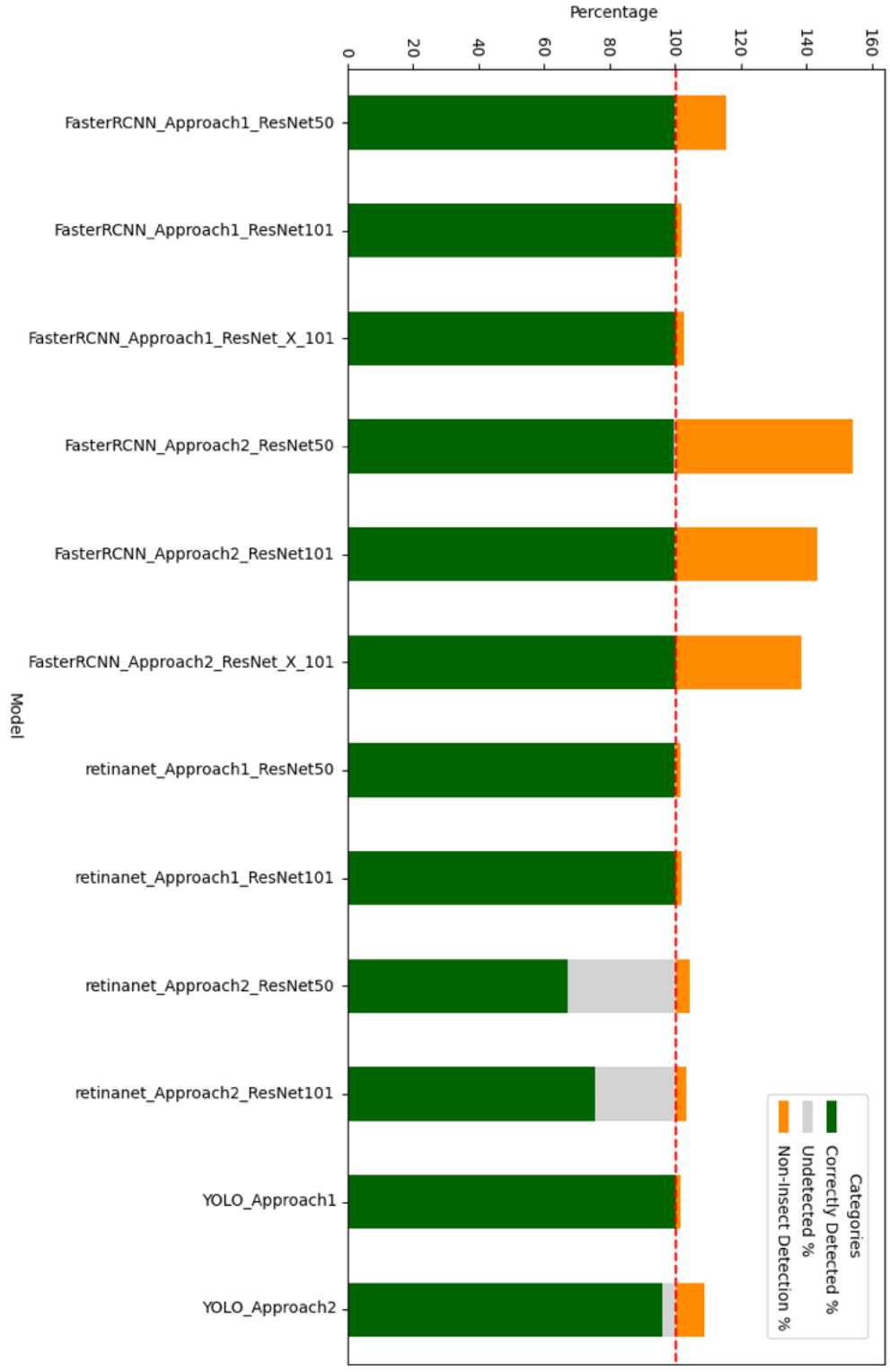

Figure 5.1: Count performance on insect level detection for all used methods for all three CNN architectures.

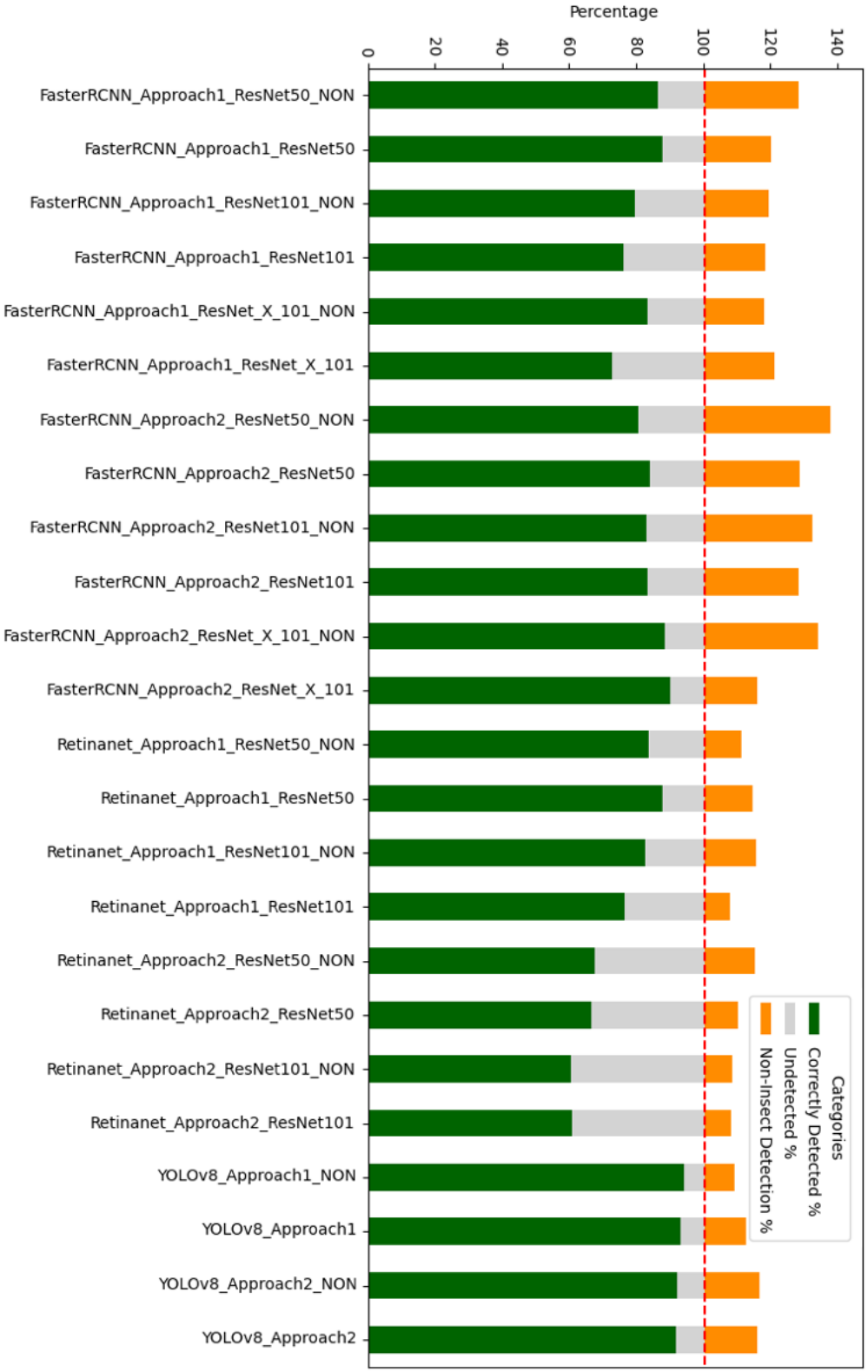

Figure 5.2: Count performance on Species level detection for all used methods for all three CNN architectures, including ("NON" as training on the NON-trained weights on insect detection model, trained weights on insect detection models, and training on no preprocessing data.)

5.2 Quantitative Analysis of Detection Metrics

As previously mentioned in Section 3.2.7, we used specific metrics including mAP and F1 Score to evaluate model performance. Table 5.1 presents results for insect-level detection. It shows that YOLO_Approach1 and FasterRCNN_Approach1_ResNeXt_101 outperform other models in detecting insects.

Table 5.1: Detection performance metrics of insect level for top 10 models

Model	**(mAP@50:95)**	**F1 Score**
YOLO_Approach1	0.93177	0.994
FasterRCNN_Approach1_ResNeXt_101	0.87300	0.989
Retinanet_Approach1_ResNet101	0.84200	0.992
Retinanet_Approach1_ResNet50	0.84100	0.991
FasterRCNN_Approach1_ResNet101	0.78900	0.993
FasterRCNN_Approach1_ResNet50	0.69200	0.917
YOLO_Approach2	0.61201	0.934
Retinanet_Approach2_ResNet101	0.41700	0.844
FasterRCNN_Approach2_ResNeXt_101	0.57400	0.762
FasterRCNN_Approach2_ResNet101	0.51000	0.724

Table 5.2 and Table 5.3 present results for species-level detection in the top 10 models' performances. Table 5.2 shows YOLOv8_Approach1_NON performed better across all species detection and Table 5.3 shows that YOLOv8_Approach2 outperforms other models in correctly detecting species based on (mAP@50:95) and F1 score.

Confusion Matrix on Species Levels

Based on evaluation metrics during the inference that is presented in Table 5.2, we consider the best configuration of each CNN architecutes for the corresponding confusion matrix to check how well species are distinguished from each other, see Figures 5.3, 5.4, and 5.5.

Table 5.2: Detection performance metrics of each species for top 10 models

Model	**Chelicerata mAP**	**Coleoptera mAP**	**Diptera mAP**	**Hemiptera mAP**	**Hymenoptera mAP**	**Lepidoptera mAP**	**Overall mAP**
YOLOv8_Approach1_NON	0.939	0.867	0.912	0.656	0.819	0.864	0.843
YOLOv8_Approach1	0.931	0.788	0.910	0.707	0.776	0.889	0.833
Retinanet_Approach1_ResNet50	0.65	0.57	0.848	0.21	0.458	0.677	0.58
Retinanet_Approach1_ResNet101	0.599	0.361	0.800	0.082	0.272	0.540	0.44
YOLOv8_Approach2_NON	0.582	0.542	0.629	0.478	0.442	0.534	0.534
YOLOv8_Approach2	0.589	0.555	0.605	0.511	0.430	0.525	0.536
Retinanet_Approach1_ResNet50_NON	0.53	0.49	0.796	0.458	0.456	0.677	0.493
FasterRCNN_Approach1_ResNeXt_101	0.520	0.338	0.804	0.236	0.387	0.304	0.391
FasterRCNN_Approach1_ResNet101	0.326	0.199	0.761	0.078	0.224	0.202	0.304
FasterRCNN_Approach1_ResNeXt_101_NON	0.376	0.276	0.713	0.0213	0.317	0.193	0.299

Table 5.3: Detection performance metrics at species level for top ten models

Model	Weighted mAP	Overall mAP	Weighted F1
YOLOv8_Approach2_NON	0.7124	0.8430	0.7404
YOLOv8_Approach2	0.6990	0.8330	0.7310
Retinanet_Approach2_ResNet101	0.4798	0.3790	0.6091
Retinanet_Approach1_ResNet101	0.4783	0.3760	0.6076
Retinanet_Approach2_ResNet50	0.4701	0.2570	0.6094
Retinanet_Approach1_ResNet50	0.4700	0.2570	0.6093
YOLOv8_Approach1_NON	0.4613	0.8430	0.5988
YOLOv8_Approach1	0.4603	0.8330	0.5983
FasterRCNN_Approach2_ResNeXt_101	0.3373	0.3910	0.5461
FasterRCNN_Approach2_ResNet101	0.3195	0.2990	0.5397

Figure 5.3: YOLO Approach 2 confusion matrix on species level

Figure 5.4: Retinanet Approach 2 ResNet101 confusion matrix on species level

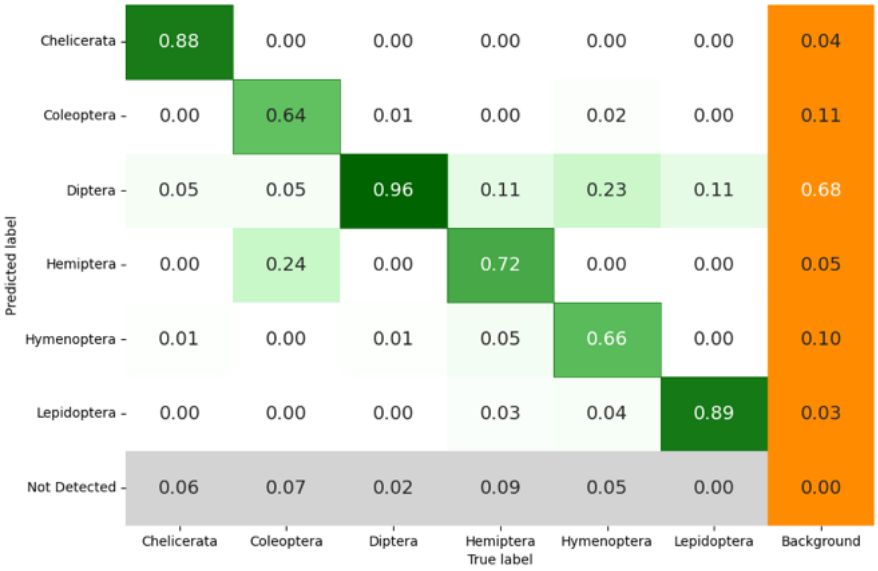

Figure 5.5: Faster R-CNN Approach 2 ResNeXt 101 confusion matrix on species level

Figure 5.3 to Figure 5.5 show across all species detection, YOLOv8 outperforms the other models.

5.3 Test on Original Images

After completing the initial validation phase, in which all models were evaluated on a preprocessed dataset, we conducted a more challenging inference to further assess the robustness of these models. For this phase, we used the validation set as the test set but removed all preprocessing steps, such as cropping and resizing, to test the models on the original, unprocessed images. This approach was designed to simulate real-world conditions, where preprocessed images may not always be feasible or practical.

The goal of this phase was to determine how well the top ten models generalize when applied to original images. Although the validation results indicated high performance on preprocessed images, it was crucial to test whether the models could maintain their effectiveness on unprocessed images, which are more representative of real-world scenarios.

The results from this unprocessed images provided valuable insights into each model's strengths and limitations. It allowed us to identify which models could still deliver reliable performance without the advantages of preprocessing, offering a more realistic measure of their practical applicability. Based on these findings, we selected our top ten models at each classification level for this evaluation, helping us understand their true generalization capability in real-world settings.

5.3.1 Insect Level Detection Models' Performance

After evaluating the models on the original images (raw images) for the test phase, Table 5.4 shows that YOLO and RetinaNet outperform Faster R-CNN at this stage of insect detection. Although YOLO is not the most robust model, it still delivers reliable performance in accurately detecting insects. Figure 5.6 provides a visual representation of the predicted instances on the test set for each model. While Faster R-CNN successfully detects insects, its precision is compromised by misclassifications of background elements as insects, leading to reduced accuracy. After testing our models for insect detection on our original images, Table 5.4 illustrates that YOLO and Retina outperform the other models at this level. While YOLO is not the most robust model, it still demonstrates a reliable performance in detecting insects. Figure 5.6 visualizes the predicted instances on the test set for each model. Although Faster R-CNN effectively identifies insects, its precision is reduced due to misclassifications of background features as insects.

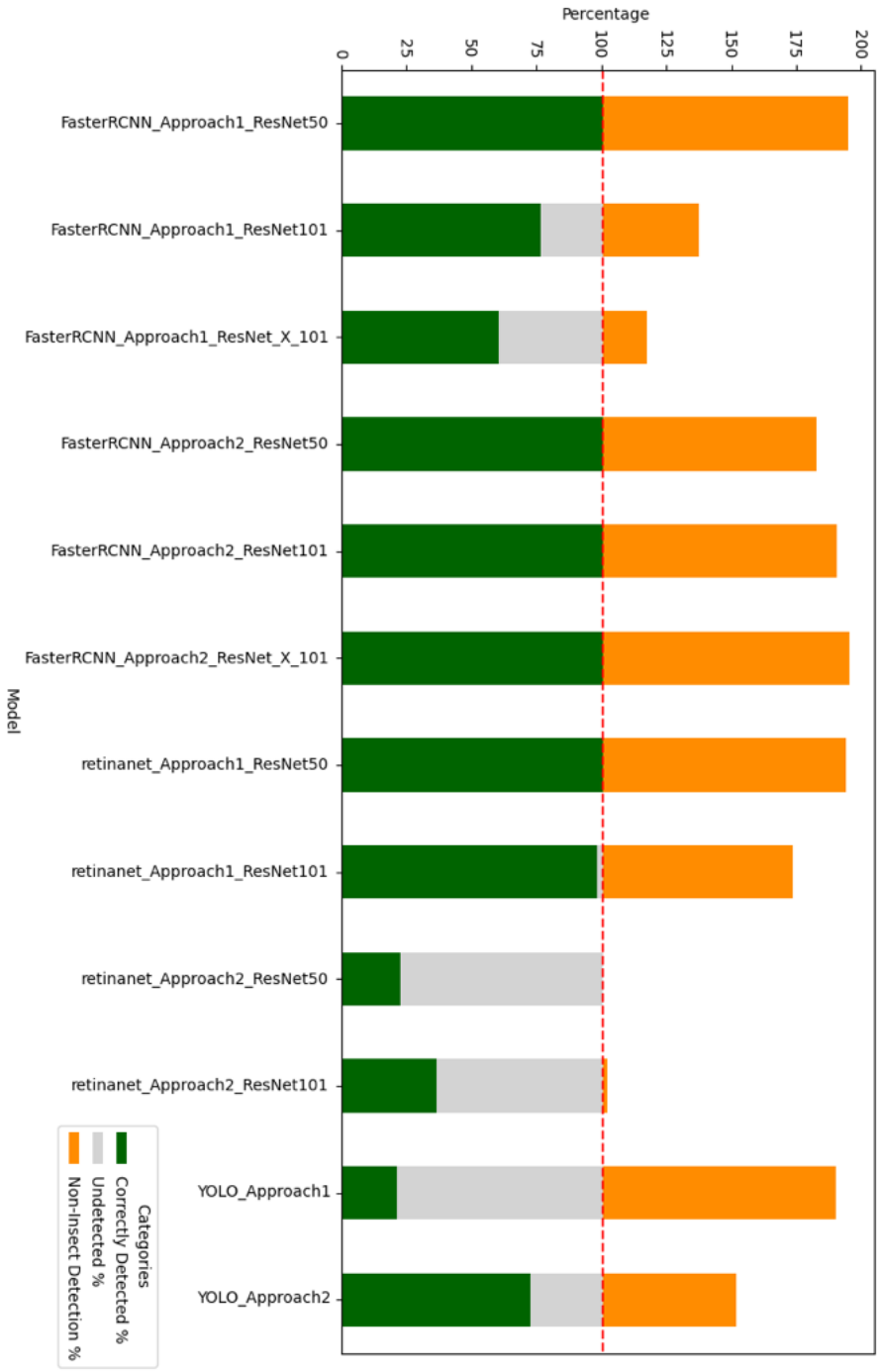

Figure 5.6: Insect level: Top 10 models confusion matrix for original images

5.3.2 Species Level Detection Models' Performance

We also tested our models at the species level to assess how well they can detect each of the species. The number of correct predicted instances in Figure 5.7 indicates that Faster R-CNN configurations have the potential to outperform the

Table 5.4: Detection performance metrics for insect level on original images

Model	**mAP(@50:95)**	**F1**
YOLO_Approach2	0.3600	0.5180
Retinanet_Approach2_ResNet101	0.1100	0.5240
Retinanet_Approach2_ResNet50	0.1000	0.3656
FasterRCNN_Approach2_ResNet101	0.1090	0.1809
FasterRCNN_Approach2_ResNet50	0.0910	0.2965
FasterRCNN_Approach2_ResNeXt_101	0.0770	0.0937
FasterRCNN_Approach1_ResNet50	0.0010	0.1028
FasterRCNN_Approach1_ResNet101	0.0000	0.6481
FasterRCNN_Approach1_ResNeXt_101	0.0000	0.6627
YOLO_Approach1	0.0000	0.0412

others. However, the significant number of misclassified background features negatively impacts its mAP.

Table 5.5 shows the robustness of each model in detecting species on the roriginal data set. In addition, Table 5.6 demonstrates the overall effectiveness of each model in this regard.

5.3.3 Model Performance on Insect Population Density

Given that the YOLO_Approach_2 model has demonstrated strong performance on the test set, we further investigate its robustness in scenarios involving high insect population densities, both at the insect level and species level. Figures 5.8 and 5.9 illustrates that the model maintains its efficacy under these conditions, successfully detecting insects and differentiating between species even in high-density conditions.

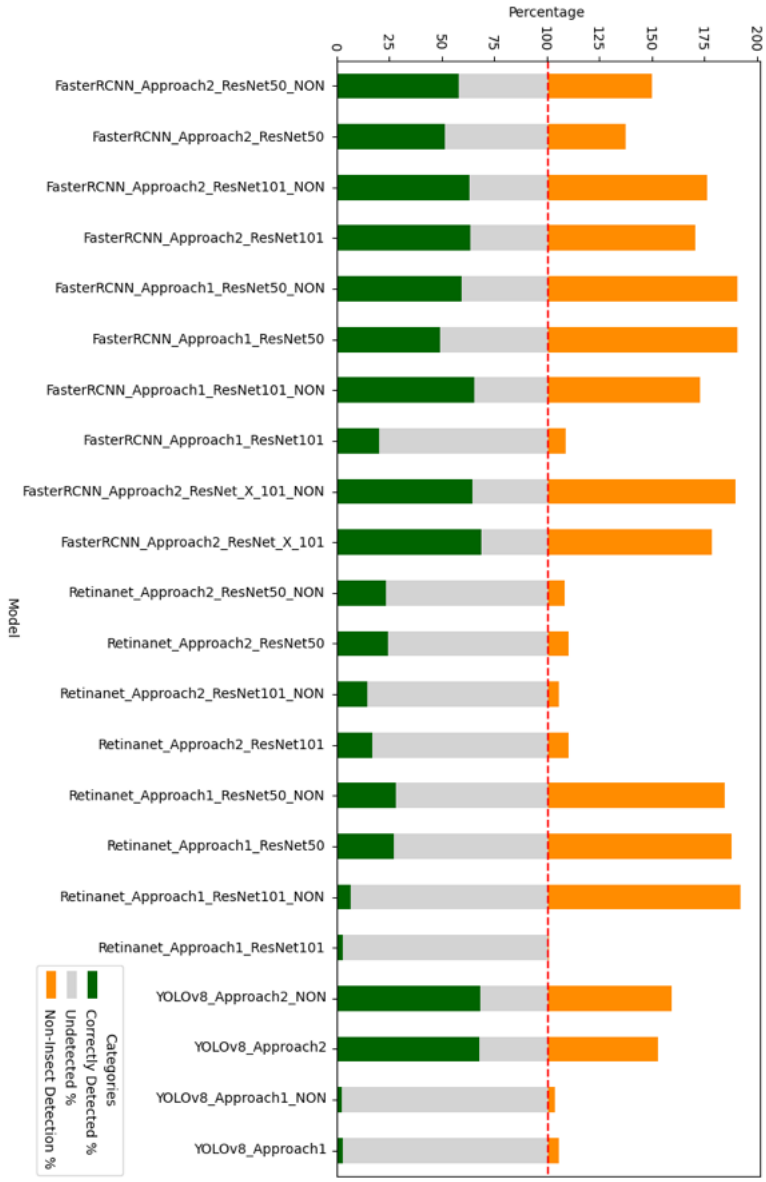

Figure 5.7: Species level: Top 10 models confusion matrix for orignal images

Table 5.5: Detection performance metrics for each species on original images

Model	**Chelicerata mAP**	**Coleoptera mAP**	**Diptera mAP**	**Hemiptera mAP**	**Hymenoptera mAP**	**Lepidoptera mAP**	**Overall mAP**
YOLOv8_Approach2	0.511	0.378	0.357	0.355	0.376	0.296	0.379
YOLOv8_Approach2_NON	0.517	0.454	0.395	0.325	0.380	0.258	0.379
Retinanet_Approach2_ResNet50	0.090	0.180	0.080	0.040	0.060	0.210	0.129
Retinanet_Approach2_ResNet50_NON	0.090	0.180	0.080	0.040	0.060	0.240	0.115
FasterRCNN_Approach2_ResNet50	0.130	0.200	0.100	0.000	0.190	0.210	0.142
FasterRCNN_Approach2_ResNet50_NON	0.050	0.180	0.080	0.000	0.220	0.220	0.129
FasterRCNN_Approach2_ResNeXt_101	0.110	0.013	0.080	0.060	0.250	0.010	0.103
YOLOv8_Approach1	0.000	0.000	0.000	0.000	0.000	0.000	0.010
FasterRCNN_Approach2_ResNet101_NON	0.060	0.170	0.080	0.000	0.200	0.060	0.100
FasterRCNN_Approach2_ResNet101	0.050	0.130	0.100	0.002	0.150	0.019	0.079

Table 5.6: Detection performance metrics of species on original images

Model	Weighted (mAP@50:95)	Overall mAP
YOLOv8_Approach2	0.34524	0.379
YOLOv8_Approach2_NON	0.34718	0.379
Retinanet_Approach2_ResNet50	0.12803	0.129
Retinanet_Approach2_ResNet50_NON	0.11691	0.115
FasterRCNN_Approach2_ResNet50	0.13027	0.142
FasterRCNN_Approach2_ResNet50_NON	0.12028	0.129
FasterRCNN_Approach2_ResNeXt_101	0.05675	0.103
YOLOv8_Approach1	0.01580	0.010
FasterRCNN_Approach2_ResNet101_NON	0.07585	0.100
FasterRCNN_Approach2_ResNet101	0.05227	0.079

Figure 5.8: Insect level detection on high-density population

.

Figure 5.9: Species level detection on high-density population

.

5.4 Different View and Image Conditions Analysis

The performance of object detection models can vary significantly depending on the viewpoint and quality of the images used for inference. Among the models

tested, YOLOv8_Approach2 consistently outperforms others across different conditions, demonstrating its robustness and versatility. Therefore, we test it in other scenarios. Figures 5.10 and 5.11 shows its performance under general conditions where any type of scenario may occur during image capture.

Figure 5.10: insect level detection under no specific conditions

.

Figure 5.11: Species level detection under no specific conditions

.

Potential scenarios that could impact the detection performance:

- Nadir view: Nadir view refers to images captured from directly above the insects. This angle usually provides the most complete view of the subject, making it potentially easier for models to detect and classify insects accurately. However, the uniformity of this perspective might also reduce the visibility of certain distinguishing features.

- Oblique view: When insects are observed from an oblique angle, different aspects of the insect's body may be revealed. This view can introduce additional challenges for detection models due to possible distortions or less visible features.

- Blurry images are common in fieldwork and can significantly challenge detection models. This subsection evaluates how image blurriness impacts model Precision, focusing on how well the models maintain detection performance when details are obscured.

- Clear images: Clear images, where details are sharp and well-defined, provide an ideal scenario for model performance. The results in this subsection serve as a control group, offering insight into each model's optimal performance under ideal conditions.

- Images with occlusions: In many natural settings, insects are partially hidden behind foliage, flowers, or other objects, making detection more difficult. This subsection evaluates the models' abilities to identify insects when they are not fully visible, analyzing how occlusions impact detection rates.

Table 5.7 and 5.8illustrates the impact of the captured images from various scenarios on YOLOv8_Approach2's detection performance at both insect and species levels.

Table 5.7: YOLOv8 detection performance under different potential scenarios at the insect level

Model	**(mAP@50:95)**	**F1 Score**
Clear_Images	0.799	0.924
Nadir_View	0.790	0.932
Blurry_Images	0.761	0.939
Oblique_View	0.775	0.918
Images_with_Occlusions	0.715	0.897
Original_Images	0.612	0.934

Table 5.8: YOLOv8 detection performance under different potential scenarios at the species level

Model	**Chelicerata mAP**	**Coleoptera mAP**	**Diptera mAP**	**Hemiptera mAP**	**Hymenoptera mAP**	**Lepidoptera mAP**	**Overall mAP**
Clear_Images	0.833	0.712	0.765	0.365	0.749	0.166	0.598
Blurry_Images	0.758	0.568	0.726	0.655	0.768	0.246	0.620
Nadir_View	0.687	0.630	0.757	0.352	0.759	0.747	0.655
Oblique_View	0.795	0.687	0.753	0.604	0.708	0.204	0.625
Original_Images	0.589	0.555	0.605	0.511	0.430	0.525	0.536
Images_with_Occlusions	0.716	0.771	0.680	0.688	0.673	0.271	0.633

CHAPTER 6 Discussion

In this chapter, we explore the findings of this project by addressing the eight key questions which were stated in Section 1.2. We leverage the insights gained from the results to examine the characteristics of the insect dataset, the behavior of the CNN models, and the overall evaluation of the outcomes.

6.1 Precision and Robustness of CNN Models

- Question1: How do these CNN models perform in terms of precision when detecting and classifying insect species in images?

  Our original dataset contains multiple insects per image, which introduces noise and complexity during model training. To address this, we employed two distinct preprocessing approaches:

  - Cropping images around the bounding boxes with a 70% threshold.
  - Splitting each image to 1200x1200 pixel images around bounding boxes and recalculating the bounding boxes accordingly.

  Table 5.2 shows that the first approach yields better results on the validation set for all three CNN models. However, when testing all selected models on the raw dataset with larger images, the precision reduces significantly for Approach_1, incompare to Approach_2. This decline is attributed to the difficulty models face in transferring learned details from smaller, lower-quality objects to larger images, a challenge noted by similar studies in object detection (Lin, Dollar, et al. 2017).

  Despite these challenges, YOLOv8 outperformed both Faster R-CNN and RetinaNet in terms of Weighted (mAP@50:95), particularly in handling small insects like Chelicerata, a tiny nearly transparent spider, measuring 20 mm to 30 mm in length, often mistaken for flowers, see Figure 4.2. Table 5.1 shows YOLO_Approach2 achieved a Weighted (mAP@50:95) of 0.61201%

at the insect level on the validation set, closely aligning with the scale of the training set. Table 5.3 also presents YOLO_Approach2 gained a Weighted (mAP@50:95) of 0.6990% at the species level, ensuring a fair evaluation across different species categories.

Even with the more challenging test set, which involved larger images and varying scales, YOLOv8 maintains its performance advantage, albeit with a reduction in precision. The Weighted (mAP@50:95) on the test set was 34.524% (Table 5.6), reflecting the model's robustness across diverse conditions. This suggests that YOLO's configuration is effective in scenarios with scale consistency while also showing resilience in more complex environments (Terven, Córdova-Esparza, and Romero-González 2023).

- Question2: What is the robustness of these CNN models in processing insect images from camera traps in natural habitats?

  Robustness in processing insect images is crucial, especially for large-scale monitoring using camera traps in natural habitats. Contrary to common perceptions regarding inference speed, while YOLOv8 maintains its advantage of faster inference compared to Faster R-CNN and RetinaNet, its training stage is often more computationally demanding and longer. This is partly due to its advanced architectural modifications aimed at optimizing detection accuracy. YOLOv8's real-time inference capability makes it highly suitable for live-stream processing and large-scale monitoring (Girshick 2015, Ren et al. 2016).

  YOLOv8's ability to detect and classify small objects like Chelicerata and Diptera demonstrates its strength in distinguishing fine details, which is crucial for accurate identification in visually complex environments. This is particularly valuable in ecological monitoring, where detecting visually ambiguous targets is essential (Li, Zhu, and Li 2021, Buehler et al. 2019).

  As shown in Table 5.5 and Table 5.6, YOLOv8 consistently performed better across various species, minimizing false positives even in challenging conditions. Despite its computational demands, YOLOv8's precision and advanced feature learning make it a powerful tool for insect detection, especially when the focus is on small species.

  While it excels in precision, its robustness may be a limiting factor depending on the task's scale and complexity. All models show robustness in detecting

insects with sufficient instances, such as Diptera, which comprises three-quarters of our dataset. However, their performance declines with less represented species, where detection capabilities decrease significantly. Faster R-CNN struggled with background misclassifications, reducing its Weighted (mAP@50:95). Similarly, RetinaNet exhibited weaker performance in noisy environments and when dealing with smaller objects.

6.2 Handling Environmental and Image Quality Challenges

- Question1: How well do CNN architectures detect insects when there are multiple insects in one image?

  Our original dataset contains images with multiple insects, but for training purposes, we crop the images around the bounding boxes to reduce noise with two distinct appraoches. During testing, we find that all three CNN models (YOLOv8, Faster R-CNN, and RetinaNet) effectively detect insects, even when multiple insects are present. However, the primary challenge lies in background noise, which can lead to false positives. This issue is particularly noticeable in Faster R-CNN, which tends to misclassify background features as insects.

- Question2: How do different scales of images in the dataset affect the performance of CNN models?

  The testing phase introduced larger images and varying scales, which posed challenges to the models' robustness. Despite these challenges, YOLOv8 outperformed both Faster R-CNN and RetinaNet, maintaining a performance advantage. The weighted (mAP@50:95) on the test set is 34.524 (Table 5.5), which highlights the model's limitations under diverse conditions such as detecting insects in large scale images. It is important to note that the lack of sufficient data is a key factor affecting the performance of all three architectures in this context.

- Question3: How effective are CNN models in detecting insects in images with partial occlusion caused by natural elements?

  YOLOv8's ability to handle small objects like Chelicerata (Figure 4.2), a tiny and nearly transparent spider and also Diptera with tiny features often mistaken for flowers, underscores its strength in learning and distinguishing

fine details. This capability is particularly valuable given the challenges of detecting such minute and visually ambiguous targets in natural environments. Even with partial occlusions, YOLOv8 consistently demonstrated robustness across various species, minimizing misclassification even in challenging conditions (Tables 5.7 and 5.8).

- Question4: How do CNN models perform when background pixels are not clear or when images contain significant noise?

  In scenarios where images are blurry or contain significant noise, the model's performance, while reduced, remains relatively robust. Table 5.7 demonstrates that YOLOv8 performs well in both nadir and oblique views. The nadir view, which captures insects from directly above, resulted in better detection performance, as the model could utilize more complete features of the insects. However, blurriness posed significant challenges, leading to a decrease in precision.

  Occlusions, where parts of the insects are hidden by other objects, further complicate detection. This issue is especially pronounced for larger insects like Lepidoptera, as their larger surface area increases the likelihood of being partially obscured, leading to a more significant drop in detection performance (see Table 5.8).

6.3 Strengths and Limitations of the Experiment

In this research, we successfully conducted insect detection and species classification using high-resolution images with noisy backgrounds, which posed significant challenges. Our approach demonstrated strong resilience, particularly with YOLOv8, which outperformed other models such as Faster R-CNN and RetinaNet. Previous studies, such as those by Li, Gu, et al. (2019) and Butera et al. (2021), focused on pest insect detection across different species using residual networks but did not explore the use of YOLOv8 with the CSP Darknet53 backbone. In contrast, our experiment utilized YOLOv8, which proved highly effective in object detection and demonstrated superior performance under various conditions.

One of the key strengths of YOLOv8 is its continuous updates and improvements by Ultralytics Jocher, Chaurasia, and Qiu (2023), ensuring that the model remains state-of-the-art. In comparison, repositories like Detectron2 Wu et al. (2019), for Faster R-CNN and RetinaNet, have seen fewer updates, potentially limiting their adaptability to new challenges.

Despite these strengths, the experiment has notable limitations. One significant challenge was the imbalance in species distribution within the dataset, which could skew the results and limit the model's ability to generalize across underrepresented species. Additionally, some species displayed variations in shape and size, likely due to subspecies differences or different life stages, which were not consistently labeled. This variability in appearance introduced complexity to model training and evaluation, potentially impacting the overall performance of the models.

CHAPTER 7 Summary and Conclusion

This study provides a thorough assessment of various CNN models, including YOLOv8, Faster R-CNN, and RetinaNet, for insect detection and species classification in high-resolution images obtained from natural habitats. The experimental outcomes shed light on the strengths and limitations of each model, offering valuable insights into their performance across diverse scenarios.

7.1 Key Observations

- Precision in detection and classification: YOLOv8 consistently outperformes Faster R-CNN and RetinaNet in terms of precision, particularly in the detection of small and complex objects such as Chelicerata and Diptera. The model showcases robust performance on the validation set, achieving mAP of 0.61201 at the insect level and a weighted mAP of 0.6990 at the species level. Even when subjected to the more challenging test set, YOLOv8 maintains its advantage, albeit with a reduced mAP of 34.524, indicating its resilience across varied conditions.

- robustness in processing: While YOLOv8 excelled in precision, it was observed to be computationally intensive and slower than its counterparts, making it less suitable for real-time analysis. On the other hand, Faster R-CNN and RetinaNet, though less accurate, presented a more balanced trade-off between precision and processing time, which could be advantageous in scenarios requiring efficient processing.

- Handling of environmental challenges: The study delved into the performance of these models under diverse environmental and image quality challenges. YOLOv8 exhibited resilience in detecting insects even in noisy backgrounds, blurry images, and when insects were partially occluded. However, larger insects like Lepidoptera were more prone to detection failures under occlusions, suggesting an avenue for potential improvement.

- Species imbalance and variability: The imbalance in species representation within the dataset posed a notable challenge, particularly for models like Faster R-CNN and RetinaNet, which grappled with smaller datasets and imbalanced data. Furthermore, the variability in species appearance, possibly attributed to different life stages or subspecies, further complicated the detection task, emphasizing the necessity for more consistent labeling and diverse data collection.

7.2 Summary

The findings from this study underscore the varying strengths and limitations of CNN models in insect detection and species classification. YOLOv8 demonstrated superior precision, particularly for detecting smaller species, and maintained robustness across different environmental conditions. However, its computational complexity and slower performance limit its application in real-time monitoring tasks. On the other hand, Faster R-CNN and RetinaNet, while less accurate, offer a more efficient solution for large-scale monitoring, making them better suited for environments where speed is a priority.

Despite the overall success of YOLOv8, the experiment revealed challenges in species imbalance and variability, which impacted the models' ability to generalize across underrepresented species. Future research should focus on improving data diversity and labeling consistency, particularly for smaller species like Chelicerata and Hemiptera, to further enhance the accuracy and generalization capabilities of insect detection models.

In conclusion, while YOLOv8 proves to be a powerful tool for precise insect detection, there is a need for further optimization to improve its processing efficiency. Faster R-CNN and RetinaNet offer viable alternatives in scenarios where computational resources are constrained, but they require more robust strategies for handling small species and noisy data.

CHAPTER 8 Future work

To overcome the limitations identified in this study and to further advance the field of insect detection in natural habitats, several key areas should be addressed in future work:

- Expert-Guided Data Collection and Labeling:

  Future efforts should prioritize data collection under the supervision of entomologists and other relevant experts to ensure the accurate labeling of insect species and subspecies. This is particularly important for addressing the issue of variability in species appearance due to different life stages or the presence of subspecies. Accurate and consistent labeling will enhance the quality of the training data, leading to better model performance and more reliable results.

- Expanding Data Collection to Diverse Environments:

  Increasing the diversity of data by capturing images from various locations and habitats is crucial. This will enable models to better generalize across different backgrounds, lighting conditions, and environmental factors. By training on a more diverse dataset, models can become more robust in detecting insects across a wide range of natural settings, improving their applicability in real-world ecological monitoring.

- Exploration of Alternative CNN Architectures and Backbones:

  While this study focused on YOLOv8, Faster R-CNN, and RetinaNet, future research should explore the use of other CNN architectures and backbones, such as EfficientNet or Vision Transformers. These architectures may offer different strengths in terms of Precision, efficiency, and robustness, potentially leading to improved performance in insect detection tasks. Experimenting with these alternative models could provide insights into how different architectures handle the challenges posed by high-resolution, noisy

images in natural habitats.

- Advanced Data Augmentation and Synthetic Data Generation:

  Incorporating advanced data augmentation techniques, such as synthetic data generation, could help address the issue of species imbalance and enhance model robustness. By generating synthetic images that mimic various environmental conditions and insect appearances, models can be trained on a wider variety of scenarios, improving their ability to detect insects in less common or more challenging situations.

- Integration of Multimodal Data:

  Future work could also explore the integration of multimodal data, such as combining image data with environmental variables (e.g., temperature, humidity) or using sensor data from camera traps. This approach could provide additional context that may improve the model's ability to accurately detect and classify insects, particularly in complex or variable environments.

- Real-Time Implementation and Field Testing: To move towards practical applications, future research should also focus on implementing these models in real-time systems and testing them in field conditions. This will help evaluate the models' performance in dynamic, real-world scenarios, and identify any further adjustments needed for effective deployment in ecological monitoring.

References

Abhishek, Jain (Jan. 2022). *All About Convolutions, Kernels and Features in CNN. Medium*. Accessed: 2024-07-28. URL: https://medium.com/abhishekjainindore24/all-about-convolutions-kernels-features-in-cnn-c656616390a1.

Ansel, Jason, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala (Apr. 2024). „PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation“. In: *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM. DOI: 10.1145/3620665.3640366. URL: https://pytorch.org/assets/pytorch2-2.pdf.

Bodla, Navaneeth, Bharat Singh, Rama Chellappa, and Larry S Davis (2017). „Soft-NMS–improving object detection with one line of code“. In: *Proceedings of the IEEE international conference on computer vision*, pp. 5561–5569.

Buehler, Patrick, Bill Carroll, Ashish Bhatia, Vivek Gupta, and Derek E Lee (2019). „An automated program to find animals and crop photographs for individual recognition“. In: *Ecological informatics* 50, pp. 191–196.

Butera, Luca, Alberto Ferrante, Mauro Jermini, Mauro Prevostini, and Cesare Alippi (2021). „Precise agriculture: effective deep learning strategies to detect pest insects“. In: *IEEE/CAA Journal of Automatica Sinica* 9.2, pp. 246–258.

Carrasco-Escobar, Gabriel, Marta Moreno, Kimberly Fornace, Manuela Herrera-Varela, Edgar Manrique, and Jan E Conn (2022). „The use of drones for mosquito surveillance and control“. In: *Parasites & vectors* 15.1, p. 473.

Chen, Ke, Chen Change Loy, Shaogang Gong, and Tony Xiang (2012). „Feature mining for localised crowd counting.“ In: *Bmvc*. Vol. 1. 2, p. 3.

Chollet, François et al. (2015). *Keras*. https://keras.io. Accessed: 2024-09-05.

Cohen, Jeremy (n.d.). *Think Autonomous - About*. Accessed: 2024-09-02. URL: https://www.thinkautonomous.ai/about/.

Date, Prasanna (Dec. 2019). „Combinatorial Neural Network Training Algorithm for Neuromorphic Computing“. PhD thesis. DOI: 10.13140/RG.2.2.27337.90726.

DeepMind, Google team (2014). *AlphaGo*. https://deepmind.google/technologies/alphago/. Accessed: 2024-09-03.

Deng, Zhipeng, Hao Sun, Shilin Zhou, Juanping Zhao, Lin Lei, and Huanxin Zou (May 2018). „Multi-scale object detection in remote sensing imagery with convolutional neural networks“. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 145. DOI: 10.1016/j.isprsjprs.2018.04.003.

Dixit, Prashant (2021). *Max Pooling, Why use it and its advantages*. https://medium.com/geekculture/max-pooling-why-use-it-and-its-advantages-5807a0190459. Accessed: 2024-09-05.

EcoTech (2024). *EcoVac Insect Vacuum*. Accessed: 2024-09-03. URL: https://www.ecotech.de/en/product/ecovac_insect_vacuum.

Everingham, Mark, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman (2010). „The pascal visual object classes (voc) challenge“. In: *International journal of computer vision* 88, pp. 303–338.

Feng, Junxi, Xiaohai He, Qizhi Teng, Chao Ren, Honggang Chen, and Yang Li (Sept. 2019). „Reconstruction of porous media from extremely limited information using conditional generative adversarial networks“. In: *Physical Review E* 100. DOI: 10.1103/PhysRevE.100.033308.

Ghosh, Anirudha, A. Sufian, Farhana Sultana, Amlan Chakrabarti, and Debashis De (Jan. 2020). „Fundamental Concepts of Convolutional Neural Network“. In: pp. 519–567. ISBN: 978-3-030-32643-2. DOI: 10.1007/978-3-030-32644-9_36.

Girshick, Ross (2015). „Fast r-cnn“. In: *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep learning*. MIT press.

Gopika, P, CS Krishnendu, M Hari Chandana, S Ananthakrishnan, V Sowmya, EA Gopalakrishnan, and KP Soman (2020). „Single-layer convolution neural network for cardiac disease classification using electrocardiogram signals“. In: *Deep learning for data analytics*. Elsevier, pp. 21–35.

Häuser, CL and K Riede (2015). „Field methods for inventorying insects“. In: *Descriptive taxonomy: the foundation of biodiversity research. Cambridge University Press, Cambridge* 8, pp. 190–213.

Holthouse, Mark, Diane Alston, Lori Spears, and Erin Petrizzo (July 2017). „Brown Marmorated Stink Bug [Halyomorpha halys (Stal)]“. In.

Howard, Andrew G, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam (2017). „Mobilenets: Efficient convolutional neural networks for mobile vision applications“. In: *arXiv preprint arXiv:1704.04861*.

Jocher, Glenn, Ayush Chaurasia, and Jing Qiu (Jan. 2023). *Ultralytics YOLO*. Version 8.0.0. URL: https://github.com/ultralytics/ultralytics.

Kingma, Diederik P and Jimmy Ba (2014). „Adam: A method for stochastic optimization“. In: *arXiv preprint arXiv:1412.6980*.

Koay, Hong Vin, Joon Huang Chuah, Chee Onn Chow, Yang-Lang Chang, and Keh Yong (Oct. 2021). „YOLO-RTUAV: Towards Real-Time Vehicle Detection through Aerial Images with Low-Cost Edge Devices“. In: *Remote Sensing* 13. DOI: 10.3390/rs13214196.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). „Imagenet classification with deep convolutional neural networks“. In: *Advances in neural information processing systems* 25.

Kumar, R (2024). *Supervised, Unsupervised, and Semi-supervised Learning*. https://www.enjoyalgorithms.com/blogs/supervised-unsupervised-and-semisupervised-learning. Accessed: 21-Jul-2024.

Kumar, Satya Prakash, A Subeesh, Bikram Jyoti, and CR Mehta (2023). „Applications of drones in smart agriculture“. In: *Smart Agriculture for Developing Nations: Status, Perspectives and Challenges*. Springer, pp. 33–48.

Labelbox (2024). *Labelbox: Data Labeling Platform*. Accessed: 2024-09-05. URL: https://app.labelbox.com/home.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). „Deep learning“. In: *nature* 521.7553, pp. 436–444.

LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998). „Gradient-based learning applied to document recognition“. In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Li, Jing, Jinan Gu, Zedong Huang, and Jia Wen (2019). „Application research of improved YOLO V3 algorithm in PCB electronic component detection“. In: *Applied Sciences* 9.18, p. 3750.

Li, Kunpeng, Junsheng Zhu, and Nianqiang Li (2021). „Insect detection and counting based on YOLOv3 model“. In: *2021 IEEE 4th International Conference on Electronics Technology (ICET)*. IEEE, pp. 1229–1233.

Li, Z., W. Nash, S. Brien, Y. Qiu, Rajeev Gupta, and Nick Birbilis (Feb. 2022). *cardiGAN: A Generative Adversarial Network Model for Design and Discovery of Multi Principal Element Alloys*.

Lin, Tsung-Yi, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick (2014). „Microsoft COCO: Common Objects in Context“. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars. Cham: Springer International Publishing, pp. 740–755. ISBN: 978-3-319-10602-1.

Lin, Tsungyi, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie (July 2017). „Feature Pyramid Networks for Object Detection“. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Long, Liangqu and Xiangming Zeng (2022). „Keras Advanced API“. In: *Beginning Deep Learning with TensorFlow: Work with Keras, MNIST Data Sets, and Advanced Neural Networks*. Berkeley, CA: Apress, pp. 283–314. ISBN: 978-1-4842-7915-1. DOI: 10.1007/978-1-4842-7915-1_8. URL: https://doi.org/10.1007/978-1-4842-7915-1_8.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

Martinez, Hector (2023). „Faster R-CNNs“. In: *PyImageSearch*. Ed. by Puneet Chugh, Aritra Roy Gosthipaty, Susan Huot, Kseniia Kidriavsteva, and Ritwik Raha. URL: https://pyimg.co/iaxmq.

Mitchell, Tom M (1997). *Machine learning*. Vol. 1. 9. McGraw-hill New York.

Montgomery, Graham A, Michael W Belitz, Rob P Guralnick, and Morgan W Tingley (2021). „Standards and best practices for monitoring and benchmarking insects“. In: *Frontiers in ecology and evolution* 8, p. 579193.

NatureNet (1995). *NatureNet*. Accessed: 2024-09-05. URL: https://naturenet.org/about-us/.

O'Connor, Rory S, William E Kunin, Michael PD Garratt, Simon G Potts, Helen E Roy, Christopher Andrews, Catherine M Jones, Jodey M Peyton, Joanna Savage, Martin C Harvey, et al. (2019). „Monitoring insect pollinators and flower visitation: The effectiveness and feasibility of different survey methods“. In: *Methods in Ecology and Evolution* 10.12, pp. 2129–2140.

O'shea, Keiron and Ryan Nash (2015). „An introduction to convolutional neural networks“. In: *arXiv preprint arXiv:1511.08458*.

Paoletti, Mercedes E, Juan Mario Haut, Javier Plaza, and Antonio Plaza (2018). „A new deep convolutional neural network for fast hyperspectral image classification“. In: *ISPRS journal of photogrammetry and remote sensing* 145, pp. 120–147.

Pavlíček, Josef, Jan Jarolímek, P Pavlíčková, S Dvořák, J Pavlík, and P Hanzlík (2018). „Automated wildlife recognition“. In: *AGRIS on-line Papers in Economics and Informatics* 10.1, pp. 51–60.

Peng, Junjie, Elizabeth Jury, Pierre Dönnes, and Coziana Ciurtin (Sept. 2021). „Machine Learning Techniques for Personalised Medicine Approaches in Immune-Mediated Chronic Inflammatory Diseases: Applications and Challenges“. In: *Frontiers in Pharmacology* 12. DOI: 10.3389/fphar.2021.720694.

Phan, Trong Huy and Kazuma Yamamoto (2020). „Resolving Class Imbalance in Object Detection with Weighted Cross Entropy Losses“. In: *CoRR* abs/2006.01413. arXiv: 2006.01413. URL: https://arxiv.org/abs/2006.01413.

Ramalingam, Balakrishnan, Rajesh Elara Mohan, Sathian Pookkuttath, Braulio Félix Gómez, Charan Satya Chandra Sairam Borusu, Tey Wee Teng, and Yokhesh Krishnasamy Tamilselvam (2020). „Remote Insects Trap Monitoring System Using Deep Learning Framework and IoT“. In: *Sensors* 20.18. ISSN: 1424-8220. URL: https://www.mdpi.com/1424-8220/20/18/5280.

Ranjbar, Sajad, Fereidoon Moghadas Nejad, Hamzeh Zakeri, and Amir H Gandomi (2020). „Computational intelligence for modeling of asphalt pavement surface distress“. In: *New Materials in Civil Engineering*. Elsevier, pp. 79–116.

Redmon, Joseph, Santosh Divvala, Ross Girshick, and Ali Farhadi (2016). „You only look once: Unified, real-time object detection“. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788.

Ren, Shaoqing, Kaiming He, Ross Girshick, and Jian Sun (2016). „Faster R-CNN: Towards real-time object detection with region proposal networks“. In: *IEEE transactions on pattern analysis and machine intelligence* 39.6, pp. 1137–1149.

Russell, Stuart J and Peter Norvig (2016). *Artificial intelligence: a modern approach*. Pearson.

Sa'doun, Mohammad Mustafa, Christopher D Lippitt, Gernot Paulus, and Karl–Heinrich Anders (2021). „A Comparison of Convolutional Neural Network Architectures for Automated Detection and Identification of Waterfowl in Complex Environments“. In: *GI_ Forum 2021,* 9, pp. 152–166.

Saadat, Md and Muhammad Shuaib (Dec. 2020). „Advancements in Deep Learning Theory and Applications: Perspective in 2020 and beyond“. In: ISBN: 978-1-83962-878-8. DOI: 10.5772/intechopen.92271.

Es-sabery, Fatima, Abdellatif Hair, Junaid Qadir, Beatriz Sainz de Abajo, Begona Garcia-Zapirain, and Isabel De la Torre Díez (Jan. 2021). „Sentence-Level Classification Using Parallel Fuzzy Deep Learning Classifier“. In: *IEEE Access* PP, pp. 1–1. DOI: 10.1109/ACCESS.2021.3053917.

Saxena, Saloni, Sneh Thorat, Prachi Jain, Rupal Mohanty, and Trupti Baraskar (Aug. 2022). „Analysis of object detection techniques for bird species identification“. In: *Journal of Physics: Conference Series* 2325, p. 012054. DOI: 10.1088/1742-6596/2325/1/012054.

Schaffhauser, Lukas (2021). „Automated Recognition of Monkeys in Natural Habitats using Convolutional Neural Networks (CNN)“. PhD thesis. Department of Geography, University of New Mexico.

Shahriar, Nafiz (July 2021). *What is Convolutional Neural Network (CNN) - Deep Learning. Medium.* Accessed: 2024-07-28. URL: https://nafizshahriar.medium.com/what-is-convolutional-neural-network-cnn-deep-learning-b3921bdd82d5.

Sharma, Sagar, Simone Sharma, and Anidhya Athaiya (2017). „Activation functions in neural networks“. In: *Towards Data Sci* 6.12, pp. 310–316.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Terven, Juan, Diana-Margarita Córdova-Esparza, and Julio-Alejandro Romero-González (2023). „A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas“. In: *Machine Learning and Knowledge Extraction* 5.4, pp. 1680–1716.

Valan, Miroslav, Karoly Makonyi, Atsuto Maki, Dominik Vondráček, and Fredrik Ronquist (2019). „Automated taxonomic identification of insects with expert-level accuracy using effective feature transfer from convolutional networks“. In: *Systematic Biology* 68.6, pp. 876–895.

Vyavhare, Suhas, M. Way, Allen Knutson, Stephen Biles, and Pearson A. (Sept. 2015). *Managing soybean insects in Texas*. DOI: 10.13140/RG.2.1.3328.3044.

Wang, Beinan (2022). „A parallel implementation of computing mean average precision“. In: *arXiv preprint arXiv:2206.09504*.

Wang, Gang, Yanfei Chen, Pei An, Hanyu Hong, Jinghu Hu, and Tiange Huang (2023). „UAV-YOLOv8: A small-object-detection model based on improved YOLOv8 for UAV aerial photography scenarios“. In: *Sensors* 23.16, p. 7190.

Wu, Jianxin (2017). „Introduction to convolutional neural networks“. In: *National Key Lab for Novel Software Technology. Nanjing University. China* 5.23, p. 495.

Wu, Yuxin, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick (2019). *Detectron2*. https://github.com/facebookresearch/detectron2.

Yamashita, Rikiya, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi (2018). „Convolutional neural networks: an overview and application in radiology“. In: *Insights into imaging* 9, pp. 611–629.

Yang, Yiming (1999). „An evaluation of statistical approaches to text categorization“. In: *Information retrieval* 1.1, pp. 69–90.

Zeiler, Matthew D and Rob Fergus (2014). „Visualizing and understanding convolutional networks“. In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*. Springer, pp. 818–833.

Zhu, Wenbo, Yan Ma, Yizhong Zhou, Michael Benton, and Jose Romagnoli (2018). „Deep learning based soft sensor and its application on a pyrolysis reactor for compositions predictions of gas phase components“. In: *Computer Aided Chemical Engineering*. Vol. 44. Elsevier, pp. 2245–2250.

List of Figures

List of Tables

Glossary

COCO Common Objects in Context format is a standard format for storing and sharing annotations for images and videos.. 59, 62

CUDA CUDA is a parallel computing platform and programming model developed by Nvidia that focuses on general computing on GPU. CUDA speeds up various computations helping developers unlock the GPU full potential.. 59

Darknet53 DarkNet-53 was developed by Joseph Redmon, the creator of the popular YOLO system. The backbone of DarkNet-53 is a series of convolutional layers that are used to analyze the input image. These layers are designed to recognize patterns and shapes that are indicative of specific types of objects.. 33, 54, 100

JSON JavaScript Object Notation is a standard text-based format for representing structured data based on JavaScript object syntax. It is commonly used for transmitting data in web applications (e.g., sending some data from the server to the client, so it can be displayed on a web page, or vice versa).. 59

LabelImg LabelImg is part of the Label Studio community. The popular image annotation tool created by Tzutalin is no longer actively being developed, but can be checked out on Label Studio, the open source data labeling tool for images, text, hypertext, audio, video and time-series data.. 33

Acronyms

AI Artificial Intelligence. 1, 16, 17, 20

ANN Artificial Neural Network. 11, 27, 28

AP Average Precision. 62, 63

C2f Cross-Stage Partial Bottleneck with Two Convolutions. 54

CIOU Complete Intersection Over Union. 33, 56, 57, 116

CNN Convolutional Neural Network. 1, 2, 3, 4, 6, 9, 11, 15, 16, 17, 27, 28, 29, 30, 32, 33, 34, 37, 43, 67, 70, 71, 73, 81, 82, 83, 84, 97, 98, 99, 100, 103, 104, 105

CPU Central Processing Unit. 31

CSP Cross-Stage Partial. 33, 54, 100

CUAS Carinthia University of Applied Science. 1, 4

DL Deep Learning. 16, 20, 21, 25, 31, 32, 58, 59, 60

FPN Feature Pyramid Network. 2, 48, 49, 50, 52

GPU Graphics Processing Unit. 31, 32, 60, 119

IOU Intersection Over Union. 56, 62, 63, 72, 75, 77

KNN K-Nearest Neighbour. 17

mAP mean Average Precision. 4, 6, 33, 34, 43, 62, 63, 72, 73, 74, 75, 77, 84, 90, 97, 98, 99, 103

ML Machine Leraning. 1, 9, 15, 16, 17, 19, 20, 41, 60, 115

NMS Non-Maximum Suppression. 2, 55

R-CNN Region-based Convolutional Neural Networks. 2, 3, 4, 8, 33, 34, 43, 76, 77, 78, 79, 88, 89, 97, 99, 100, 103, 104, 105, 116

ReLU Rectified Linear Unit. 21, 22

ResNet Residual Network. 34, 43, 74, 75, 76

ResNeXt Extended Residual Network. 43

ROI regions of interest. 46

RPN Region Proposal Network. 44, 45, 46, 48

SGD Stochastic Gradient Descent. 26

SPPF Spatial Pyramid Pooling Fast. 54

SSD Single Shot MultiBox Detector. 32

SVM Support Vector Machine. 17

TFX TensorFlow Extended. 31

UAS Unmanned Aerial Systems. 15, 16

YOLO You Only Look Once. iii, 2, 3, 4, 8, 32, 33, 34, 35, 41, 43, 52, 53, 54, 55, 56, 57, 58, 59, 60, 71, 72, 73, 74, 75, 78, 87, 88, 97, 98, 99, 100, 103, 104, 105, 116, 119