RIGA TECHNICAL UNIVERSITY

Faculty of Computer Science, Information Technology and Energy

Institute of Applied Computer Systems

Bekassyl Adenov

Academic Bachelor Level Study Program “Computer System”

Student ID No 221ADB130

TREE NURSERIES WEED RGB'S CAMERA RAW DATA PRE-PROCESSING AND PREPAIRING A DATASET FOR DEEP LEARNING TASKS

BACHELOR THESIS

Scientific adviser Dr.sc.ing., Leading researcher

Andrejs Zujevs

RIGA 2025

RIGA TECHNICAL UNIVERSITY

FACULTY OF COMPUTER SCIENCE, INFORMATION TECHNOLOGY AND ENERGY

INSTITUTE OF APPLIED COMPUTER SYSTEMS

Work Performance and Assessment Sheet of the Bachelor Thesis

The author of the graduation thesis:

Student Bekassyl Adenov ______________________________

(signature, date)

The graduation thesis has been approved for the defence:

Scientific adviser:

Dr.sc.ing. Leading Researcher Andrejs Zujevs __________________________

(signature, date)

ABSTRACT

Keywords: Weeds, Tree nurseries, Deep Learning, Data pre-processing, Dataset creation, Weed detection.

Weeds pose significant challenges to the efficient operation of tree nurseries, impacting forest sector productivity. To address this, the current study focuses on pre-processing RGB camera raw data recorded from Latvian tree nurseries and creating a high-quality dataset tailored for deep learning tasks. This work aims to enable automated weed detection using deep learning technologies, which can support the planning and implementation of effective weed control strategies.

This thesis involves researching existing online datasets of weeds, understanding principles of data pre-processing and producing a new set by labelling images of several weed species. PyTorch is used to train DL models and load prepared datasets for training and testing purposes. The findings established the groundwork for creating sophisticated weed identification models to add toward modernized monitoring systems within tree nurseries. Future work will focus on integrating and extending the dataset with real-time weed control technology.

The volume of the thesis report: 77 pages, 16888 words 37 figures, 3 tables, 44 references.

ANOTĀCIJA

Atslēgvārdi: Nezāles, kokaudzētavas, Dziļā mācīšanās, Datu pirmapstrāde, Datu kopas izveide, Nezāļu noteikšana.

Nezāles rada būtiskus izaicinājumus kokaudzētavu efektīvai darbībai, negatīvi ietekmējot mežsaimniecības nozares produktivitāti. Lai risinātu šo problēmu, šajā pētījumā uzmanība tiek pievērsta RGB kameras izejas datu pirmapstrādei, kas iegūti no Latvijas kokaudzētavām, un augstas kvalitātes datu kopas izveidei, kas pielāgota dziļās mācīšanās uzdevumiem. Šī darba mērķis ir veicināt automatizētu nezāļu noteikšanu, izmantojot dziļās mācīšanās tehnoloģijas, kas var palīdzēt plānot un īstenot efektīvas nezāļu apkarošanas stratēģijas.

Šajā bakalaura darbā tiek pētītas esošās tiešsaistē pieejamās nezāļu datu kopas, izprasti datu pirmapstrādes principi, kā arī izveidota jauna datu kopa, marķējot attēlus ar dažādu nezāļu sugu attēlojumiem. PyTorch ietvars tiek izmantots dziļās mācīšanās modeļu apmācībai un sagatavoto datu kopu ielādei apmācības un testēšanas nolūkos. Iegūtie rezultāti veido pamatu sarežģītu nezāļu identificēšanas modeļu izstrādei, veicinot modernizētu monitoringa sistēmu ieviešanu kokaudzētavās. Nākotnes darbi tiks vērsti uz šīs datu kopas integrēšanu un paplašināšanu, izmantojot reāllaika nezāļu apkarošanas tehnoloģijas.

Bakalaura darba apjoms: 77 lappuses, 16888 vārdi, 37 attēls, 3 tabulas, 44 izmantotie avoti.

TABLE OF CONTENTS

TABLE OF FIGURES

INTRODUCTION

Tree nurseries serve as the foundational stage of sustainable forestry management in Latvia, a country where the forest sector plays a critical role in both economic development and environmental balance. One of the primary challenges in tree nurseries is the management of weeds, which compete with young saplings for vital resources like water, sunlight, and nutrients. Manual weeding methods, though effective, are time-consuming, labor-intensive, and inconsistent in outcomes. As the need for precision and automation in agriculture increases, the use of deep learning technologies has emerged as a promising solution for automating weed detection and supporting effective control strategies. However, the successful application of such technologies depends heavily on the availability of high-quality, domain-specific datasets—something notably lacking for tree nurseries, particularly in Latvian environmental conditions. This highlights the topicality and necessity of the present research: to develop a pre-processed, annotated dataset from raw RGB camera data for the purpose of training deep learning models in weed detection.

The main aim of this thesis is to pre-process and annotate the RGB camera's raw data from tree nurseries and create a new dataset for weed detection by using deep learning technology. To achieve this goal, several tasks were undertaken:

1) Datasets. Research and describe online available datasets of weeds. Determine and describe in details already available datasets (also available in online) aimed at weed and particularly weeds growing in tree nurseries.
2) Data pre-processing. Describe and investigate the methods, hardware, general principles, and steps for data pre-processing tasks to create a dataset. The description should include how raw data is obtained, pre-processed, annotated, and finally published.
3) Deep learning. Describe the ML technology, and main concepts, and prepare a list of famous examples. Describe differences between the classical and deep learning approaches. Generally describe working principles of CNN neural networks, data preprocessing and annotation principles used within deep learning.
4) New dataset. Make different weeds' visual data annotation by using online annotation tools for the previously recorded visual raw data from

tree nurseries. Develop a software which is dedicated for data feeding deep learning within the PyTorch library.

5) Augmented data. Describe and develop different the data augmentation methods within the Pytorch's data loader, test them and validate by using the new dataset.
6) Conclusions. Describe future development for the new dataset and its usage.

This thesis is structured into several key sections. Chapter 1 presents theoretical foundations of machine learning and deep learning, with a focus on Convolutional Neural Networks (CNNs) and the role of datasets. Chapter 2 reviews existing datasets and related work, highlighting their features and limitations in the context of weed detection. Chapter 3 outlines the data collection, annotation, cleaning, and augmentation pipeline. Chapter 4 covers the practical implementation of the dataset loader and integration with augmentation techniques in PyTorch. Chapter 5 will cover the results achieved during the work. Chapter 6 will contain conclusion and future work. The Appendix includes technical documentation for label extraction and conversion to YOLOv1 format.

By addressing a research gap in agroforestry technology, this work contributes to the modernization of tree nursery management through AI-powered solutions and provides a scalable dataset model for future research in the field.

1. FUNDAMENTALS OF DEEP LEARNING AND IMAGE DATA

1.1 Machine Learning

Machine Learning (ML) has transformed artificial intelligence (AI) by enabling systems to learn patterns from data, make decisions with minimal human supervision, and adapt different scenarios. ML methodologies are traditionally classified into three major categories: supervised learning, unsupervised learning, and reinforcement learning.

In supervised learning, models are trained on labeled datasets, where each input is paired with the correct output. This approach is commonly utilized in areas like image recognition and natural language processing. For example, in healthcare, algorithms are trained on annotated X-ray images linked to specific diagnoses. The goal of supervised learning is to develop an accurate mapping from inputs to outputs, enabling models to make reliable predictions on new, unseen data (Mitchell, 1997). Convolutional Neural Networks (CNNs) are a notable application in supervised learning, particularly for facial recognition technologies employed in security and social media. As illustrated in Figure 1.1, the model receives input features and their corresponding labels to learn mappings that can be generalized to unseen data.

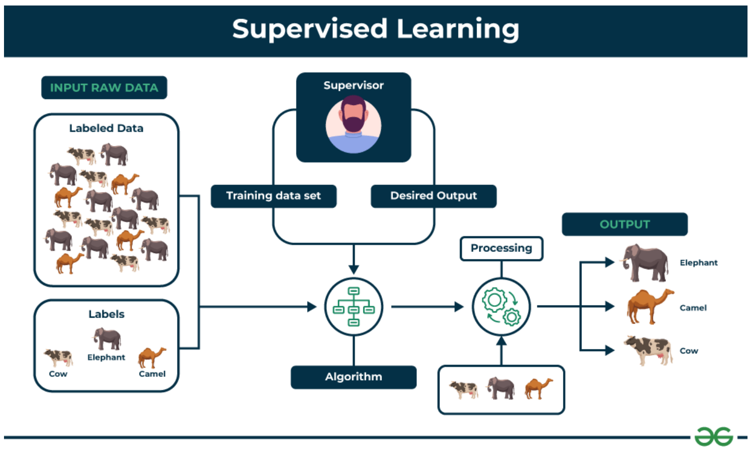


Figure 1.1 Supervised learning visual representation (adopted from GeeksforGeeks)

Regression and classification are two fundamental types of supervised learning problems. Regression is used when the target output is a continuous value, such as predicting house prices or temperature. Common metrics used for evaluating regression models include Mean Squared Error (MSE) and R-squared. On the other hand, classification is used when the target output is a discrete class label, such as determining whether an email is spam or not. Metrics for classification problems often include accuracy, precision, recall, and F1-score.

Conversely, unsupervised learning deals with unlabeled data, aiming to uncover hidden patterns or groupings without predefined outcomes. This method is particularly effective for tasks like clustering and dimensionality reduction. For instance, businesses use unsupervised learning to segment customers based on purchasing behavior, enabling targeted marketing strategies. Techniques like clustering and Principal Component Analysis (PCA) are prevalent in this domain (Alpaydin, 2020).

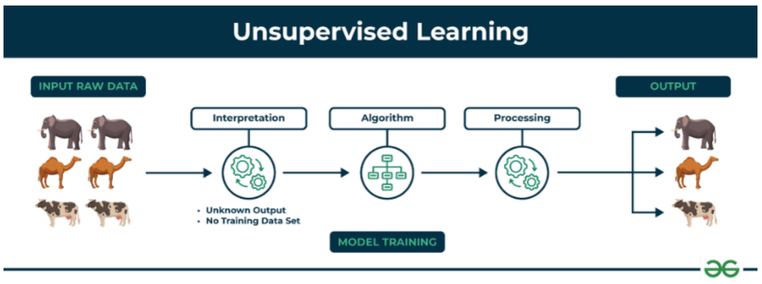


Figure 1.2 Unsupervised Learning visual representation (adopted from GeeksforGeeks)

Another major path in ML is reinforcement learning, in which an agent interacts with its environment and learns to choose a sequence of decisions by receiving feedback in the form of rewards or penalties. A technique popularised in behavioural psychology, this approach has demonstrated impressive results as seen in robotics and strategic game playing, among its many other successes. One notable example is AlphaGo created by DeepMind, which defeated human champions in the complex game of Go (Sutton & Barto, 2018).

Possessing a good ML model construction starts with data preprocessing cleaning normalizing and converting raw data into representable forms appropriate for analysis. This measure is critical because the quality of input directly affects the model performance (Han et al., 2011). Essential techniques include normalization, missing values, and feature selection to ensure optimal learning processes.

The next step after preprocessing is model selection where the algorithms are selected according to the problems. Some problems can be solved using linear regression, while others, such as image classification, can be addressed through deep learning architectures (Hochreiter & Schmidhuber, 1997), such as CNN (Convolusional Neural Network); RNN (Recurrent Neural Networks).

The training and validation are then organized to optimize the parameters of the model and evaluate its performance. K-fold cross-validation helps prevent overfitting, where the model memorizes training data yet is ineffective on novel examples (Goodfellow et al., 2016). The regularization techniques and dropout in neural networks are general techniques to improve model generalization. More advances, such as GANs, have made a huge impact by allowing very realistic, artificial data to be generated. GANs are based on the idea of a competition between two networks, referred to as a generator and a discriminator — which iteratively improve their performance (Goodfellow et al., 2014).

A further significant development is the emergence of Large Language Models (LLMs), like GPT-3, that can produce plausible, context-sensitive text across many tasks, including translation and creative writing (Brown et al., 2020).

As ML systems win favor amidst industries, ethical dimensions as bias, fairness and transparency become pivotal. For example, biased datasets can lead to unfair treatment in applications such as hiring or healthcare, highlighting the importance of robust ethical frameworks in deploying ML (Obermeyer et al., 2019).

To summarize, Machine Learning is an ever-unfolding field that progresses at an incredible speed, propelled by technology and the availability of data. The rapidadvancements in AI should be of concern to both researchers and practicioners alike and deserves focus both by people working in the technical domains right up until ethics.

1.2 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) represent a specialized class of deep neural networks designed to process data with grid-like structures, such as images. Their architecture is inspired by the biological visual cortex and is particularly effective for tasks like image classification, object detection, and semantic segmentation (Goodfellow et al., 2016).

CNN consists of three principal types of layers: convolutional layers, pooling layers, and fully connected layers. As illustrated in Figure 1.3, this layered design is tailored for tasks such as object detection. Convolutional layers apply multiple learnable filters across the input image to extract critical spatial features, including edges, textures, and complex patterns. By convolving these filters over the input, the network creates feature maps that retain spatial hierarchies and local relationships, enabling the model to recognize intricate structures within visual data.

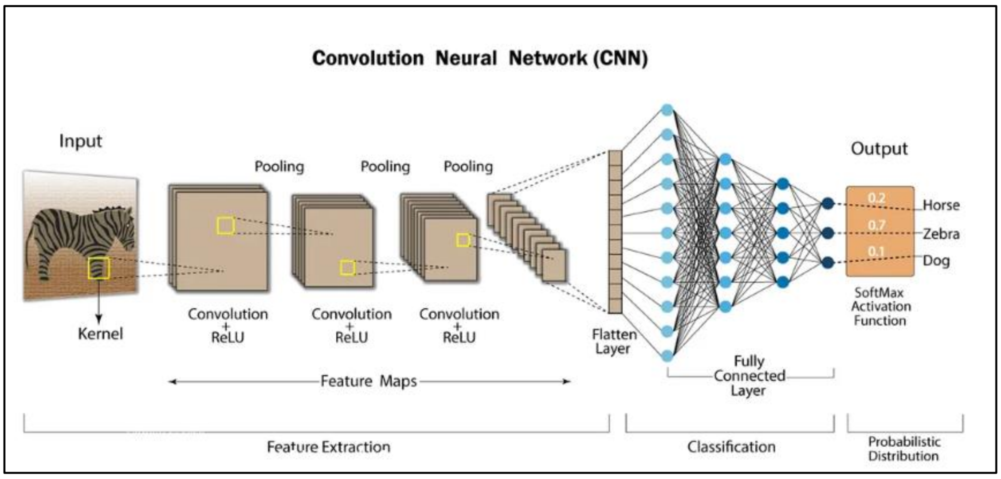


Figure 1.3 Convolutional Neural Network — CNN architecture (adopted from (Haque, 2020)

Following convolutional layers, pooling layers are employed to reduce the spatial dimensions of feature maps. This down sampling not only reduces computational demands but also introduces a degree of translation invariance, allowing the model to identify objects regardless of slight positional changes (Olsen et al., 2019). Finally, the fully connected layers interpret these high-level features to produce output predictions using functions like SoftMax.

The evolution of CNN architecture has yielded several landmark models that progressively advanced performance and efficiency. AlexNet marked a significant breakthrough by winning the ImageNet competition in 2012, introducing techniques like ReLU activation and dropout regularization. VGGNet (Simonyan & Zisserman, 2015) demonstrated the benefits of deeper networks with uniform convolutional layers, while ResNet (He et al., 2016) addressed the vanishing gradient problem by introducing residual connections, enabling the training of ultra-deep networks.

In the context of this thesis, CNNs are leveraged to classify and localize various features within tree nurseries, including distinguishing between different types of pine crowns and identifying weeds. Given the visually complex and overlapping nature of nursery environments, CNNs are particularly advantageous, as they can learn to differentiate subtle variations in appearance between saplings and weeds, even under challenging conditions like dense vegetation and variable lighting.

Moreover, the strategy of transfer learning—where pre-trained CNNs are fine-tuned on specialized datasets—proves highly effective when labeled data is limited. Pre-trained models like YOLO can be adapted to tree nursery datasets, significantly enhancing detection accuracy and reducing training time (Olsen et al., 2019). Transfer learning has been especially impactful in agricultural vision tasks, where domain-specific data is often scarce.

Thus, CNNs form a critical component of the deep learning solutions proposed in this work, enabling robust and scalable object detection tailored to the unique challenges of tree nursery environments.

1.3 Importance of Datasets in Deep Learning

Datasets are the basis of deep learning success, fundamentally influencing a model's ability to learn, and perform precisely. The quality, volume, and diversity of a dataset directly impact how effectively a deep learning model can recognize patterns and make reliable predictions. Tasks like weed detection within tree nurseries, datasets must encapsulate a wide variety of weed species captured under diverse lighting conditions, angles, and degrees of occlusion to ensure the resulting models are robust and generalizable.

High-quality datasets enable supervised learning models—such as Convolutional Neural Networks (CNNs) to learn meaningful feature representations. In contrast, low-quality or poorly annotated datasets can severely compromise model performance, resulting in biased outputs or underperforming systems (Wang et al., 2022). An ideal dataset, as depicted in Figure 1.4, contains a rich diversity of samples, consistent annotations, and accurately reflects the complexity of real-world environments.

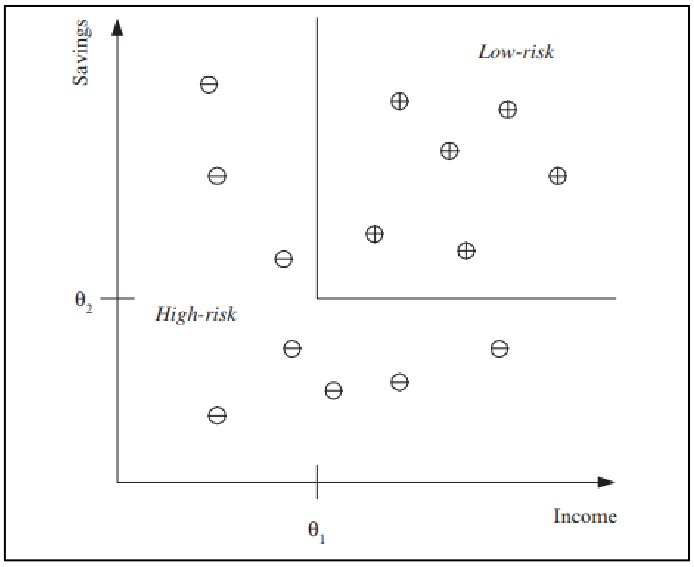


Figure 1.4 Training dataset (adopted from Alpaydin, 2020)

Consistency and accuracy of data labeling are also main aspects of the dataset. Inaccurate or inconsistent annotations can mislead models during training, leading to poor generalization. Consequently, meticulous annotation practices are essential, particularly for applications demanding high precision, such as agricultural monitoring.

Furthermore, the symbolic of a dataset determines a model’s ability to operate in different scenarios. If a dataset only covers a narrow range of conditions, the trained model may fail when encountering unexpected variations, such as different soil backgrounds, seasonal lighting changes, or varying weed growth stages.

Dataset size also plays a critical role. Larger datasets generally allow for better generalization by exposing models to a broader range of examples. However, simply increasing the dataset size without maintaining quality can introduce noise, reducing the effectiveness of learning. Therefore, a balance between quantity and quality must be achieved.

1.3.1 Dataset Creation

The process of creating a dataset is a systematic and meticulous endeavor, critical across diverse research fields such as ML, social sciences, and healthcare informatics. It begins with clearly defining the research objectives, which then guide the identification of appropriate data sources. Data can be collected directly through primary methods like surveys and experiments, or indirectly by utilizing secondary sources such as existing databases. For instance, in healthcare research, electronic health records have become a vital source of rich, structured data, offering vast opportunities for improving patient outcomes (Raghupathi & Raghupathi, 2014).

Following the determination of data sources, the next crucial step is data collection. This phase must be carefully planned to select appropriate methodologies. Researchers may employ qualitative approaches such as interviews and focus groups or quantitative methods, like surveys and automated web scraping. Employing validated instruments during collection is critical to ensuring the reliability and validity of the gathered information (DeVellis, 2016).

In the context of image datasets, data collection often involves capturing new imagery using cameras, drones, or specialized imaging devices, or scraping images from online repositories. Methods like web scraping must be approached cautiously, ensuring compliance with copyright regulations (Shrivastava et al., 2016). Collaboration with institutions or communities, or the use of crowdsourcing platforms, also provides an avenue for gathering diverse and representative visual data.

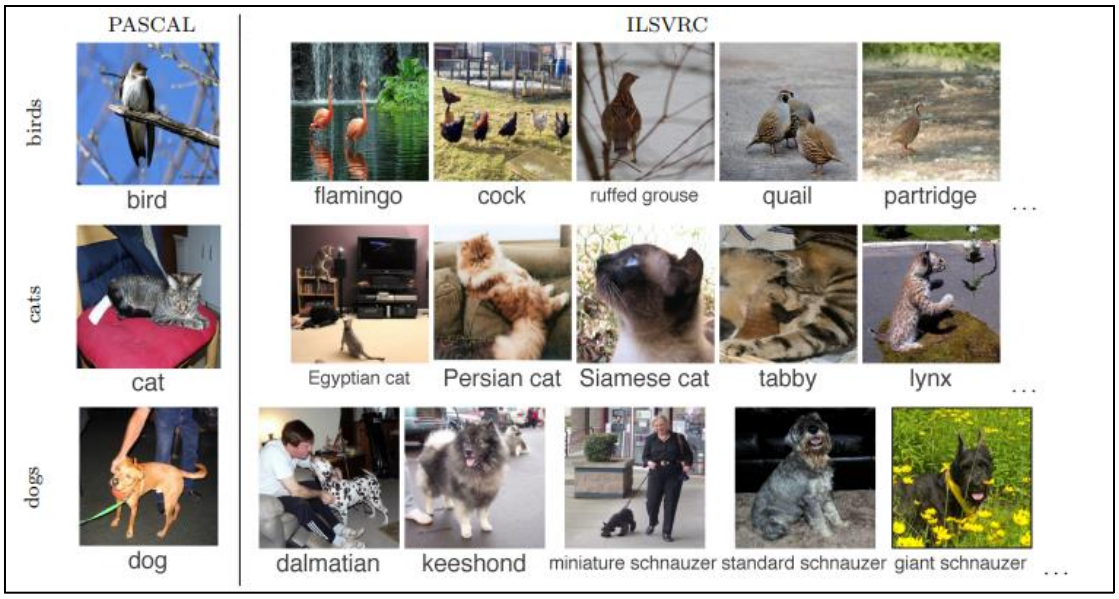


Figure 1.5 ILSVRC dataset which contains different classification of animals (adopted from Russakovsky et al., 2015)

Once collected, datasets must undergo thorough cleaning and preprocessing. This step involves removing duplicate entries, correcting errors, handling missing values, and standardizing formats to prepare the data for subsequent analysis. Proper cleaning ensures that models trained on the data achieve higher accuracy and reliability (Zadeh et al., 2020). Without this stage, any subsequent modeling or analysis risks being undermined by inconsistencies or biases inherent in the raw data.

For supervised learning tasks annotation is key. In image datasets, this involves labeling objects, regions, or features within the images. Accurate annotation is paramount, as the quality of labeled data directly affects model training outcomes (Russakovsky et al., 2015). Poor or inconsistent annotations can severely hinder a model's performance, underscoring the need for stringent quality control protocols during labeling (Paszke et al., 2019).

Following cleaning and annotation, the dataset must be partitioned into training, validation, and test sets. This division supports effective model evaluation and prevents overfitting. Properly structured splits, often following an 80/10/10 or 70/15/15 ratio, allow researchers to measure model performance reliably and generalize findings beyond the initial dataset (Kohavi, 1995).

Additionally, comprehensive documentation and the creation of detailed metadata are crucial. Recording methods of data collection, definitions of variables, and preprocessing steps not only enhance dataset usability but also ensure research reproducibility and transparency (Gurtoo & Elisseeff, 2003).

In conclusion, dataset creation is a deliberate and multi-faceted process encompassing careful planning, systematic collection, thorough cleaning, precise annotation, strategic splitting, and meticulous documentation. Each stage contributes to building datasets that are not only fit for immediate research needs but also serve as valuable resources for future scientific inquiries.

1.4. Dataset Splitting (Train/Validation/Test)

Splitting a dataset into distinct subsets is a fundamental step in the machine learning pipeline, crucial for evaluating a model's ability to generalize to unseen data. Without proper dataset partitioning, there is a substantial risk that a model will simply memorize the training examples—a phenomenon known as overfitting—rather than learning generalizable patterns.

Typically, a dataset is divided into three main subsets: the training set, the validation set, and the test set (Kohavi, 1995).

- The training set is used to fit the model by adjusting its parameters through optimization algorithms like stochastic gradient descent.
- The validation set assists in hyperparameter tuning and model selection. It enables practitioners to monitor performance during training and adjust model configurations, such as learning rates or dropout rates, to improve generalization.
- The test set provides an unbiased final assessment of the model's predictive capabilities after training is complete, offering a realistic estimate of performance in real-world applications (Russakovsky et al., 2015).

A common rule of thumb is to allocate 70% of the data for training, 15% for validation, and 15% for testing. However, these ratios can be adjusted depending on the overall size of the dataset. In cases where data is scarce—as was the situation in this thesis—alternative strategies like k-fold cross-validation are employed. In k-fold cross-validation, the dataset is partitioned into *k* subsets, and the model is trained and validated *k* times, each time using a different subset for validation and the remaining ones for training. This approach provides a more reliable estimate of model performance, particularly when working with limited data.

When dealing with image datasets, particular caution must be exercised to prevent data leakage, where information from the test set unintentionally influences training. For example, if images taken under slightly varying lighting conditions from the same scene appear in both the training and testing sets, it could artificially inflate model performance. Ensuring that similar frames are kept exclusively within one subset is crucial for obtaining honest evaluations (Xia et al., 2018).

Moreover, proper splitting fosters reproducibility. Clearly defined splits enable other researchers to replicate experiments and compare results consistently. Public datasets like ImageNet and COCO have standardized splits to ensure fair benchmarking across different models and research efforts (Russakovsky et al., 2015).

In this thesis, special care was taken during dataset partitioning. Images were divided based on the scene and capture date, ensuring that visually similar frames were confined to the same subset. This method reduces overfitting and better simulates how the model would perform when deployed in real-world nursery environments, where visual conditions can vary significantly.

1.5 Model Architectures Overview: Faster R-CNN and YOLO

Deep learning-based object detection models are essential for tasks requiring both object localization and classification. Two major approaches dominate the field: one-stage detectors such as YOLO (You Only Look Once), and two-stage detectors such as Faster R-CNN (Region-based Convolutional Neural Networks). In this project, both approaches were explored to understand their performance characteristics for weed detection in forestry nursery environments.

1.5.1 YOLOv8 Architecture

The YOLO family, starting with YOLOv1 (Redmon et al., 2016), introduced a new paradigm in object detection by treating detection as a direct regression problem. Instead of generating region proposals separately, YOLO models predict bounding boxes and class probabilities simultaneously from a single convolutional feature map.

YOLOv8, developed by Ultralytics (Jocher et al., 2023), represents one of the latest and most efficient iterations in the YOLO family. Key architectural innovations in YOLOv8 include:

- Backbone: A highly optimized Convolutional Neural Network (CNN) for extracting spatial features from input images. YOLOv8 typically uses CSP (Cross Stage Partial Networks) or similar lightweight designs to balance accuracy and inference speed.
- Neck: Feature Pyramid Network (FPN) or Path Aggregation Network (PANet) modules are used to merge feature maps from different scales, improving detection of both large and small objects.
- Head: Decoupled classification and regression heads, allowing the model to specialize separately in object classification and bounding box localization.
- Anchor-Free Detection: Unlike earlier YOLO versions that relied on predefined anchor boxes, YOLOv8 uses anchor-free approaches, predicting object centers and box dimensions dynamically, simplifying training and reducing hyperparameter sensitivity.
- Data Augmentation: Techniques such as Mosaic augmentation, MixUp, and random affine transformations are embedded into the training process, substantially improving generalization.

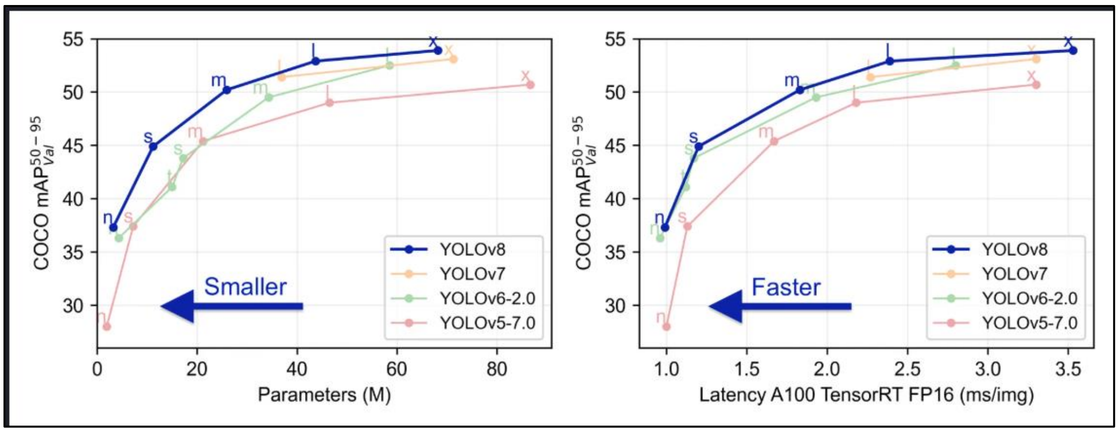


Figure 1.6 YOLOv8 compared to other YOLO models (adopted from Jocher et. al., 2023)

YOLOv8 models are optimized for real-time applications, achieving inference speeds exceeding 100 frames per second on modern GPUs, while maintaining competitive detection accuracy.
In the context of forestry nursery weed detection, such performance enables real-time drone scouting or robotic intervention scenarios.

1.5.2 Faster R-CNN Architecture

Faster R-CNN (Ren et al., 2015) is a two-stage detector that builds upon the earlier R-CNN and Fast R-CNN architectures.
Its design can be divided into three main components:

- Feature Extractor: A deep CNN (e.g., ResNet-50 or ResNet-101) generates a rich feature map from the input image.
- Region Proposal Network (RPN): Instead of relying on external algorithms like Selective Search, Faster R-CNN introduces a fully convolutional RPN that proposes candidate object regions directly from feature maps.
- Detection Head: Each proposed region is pooled into a fixed-size feature vector using Region of Interest (RoI) Align or RoI Pooling and then passed through two parallel branches: one for classifying the object, and another for refining the bounding box coordinates.

Faster R-CNN uses anchor boxes at multiple scales and aspect ratios to generate proposals, allowing it to detect objects of various sizes and shapes. The model is trained end-to-end using a multi-task loss combining classification and regression objectives.

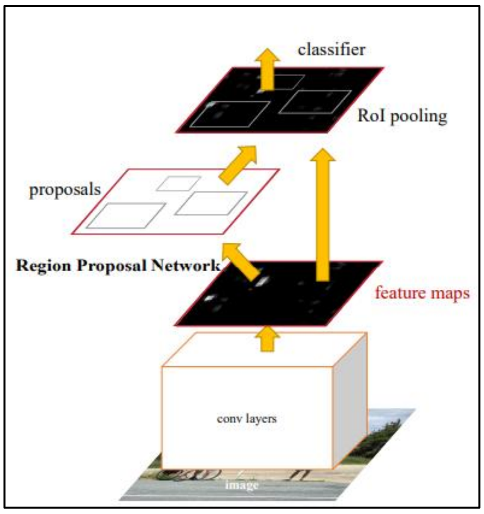


Figure 1.7 schematic demonstration of Faster RCNN (adopted from Ren et al., 2015)

While computationally more expensive than YOLOv8, Faster R-CNN often delivers higher detection accuracy, especially for small objects or in scenarios with significant occlusions — conditions commonly encountered when weeds grow close to saplings.

1.5.3 Training Processes and Loss Functions

YOLOv8 employs a compound loss function consisting of:

- Bounding box regression loss (CIoU Loss or DIoU Loss),
- Objectness score loss (binary cross-entropy),
- Classification loss (cross-entropy).

The model predicts multiple bounding boxes per grid cell and uses non-maximum suppression (NMS) to select the best candidates during inference. Faster R-CNN is trained using:

- Region proposal loss (objectless vs background),
- Bounding box regression loss for proposals,
- Final classification loss for detected objects,
- Bounding box regression refinement for final detections.

This two-step loss optimization makes Faster R-CNN highly precise, but computationally slower compared to YOLO architectures.

Table 1.1

Advantages and Disadvantages

Characteristics	**YOLOv8**	**Fast R-CNN**
Detection Type	One-stage	Two-stage
Interference speed	Very high	Moderate to low
Model complexity	Simpler	More complex
Small object detection	Good, but not optimal	Excellent
Training Complexity	Easier to configure	Requires more tuning
Best for	Real-time applications	Precision critical tasks

2. REVIEW OF EXISTING DATASETS AND RELATED WORK

2.1 Online Weed datasets

The **DeepWeeds dataset** is an essential resource for advancing weed classification methodologies, comprising 17,509 labeled images of eight nationally significant weed species native to various locations across northern Australia. This dataset is particularly valuable for developing and benchmarking deep learning models aimed at improving weed detection in agricultural settings. As noted by Olsen et al. (2019), "The dataset provides a diverse representation of weed species, enabling the training of models that can generalize well across different environments and conditions." This diversity is crucial for precision agriculture, where accurate weed identification can lead to more effective management strategies, reducing herbicide use and improving crop yields. The images in the DeepWeeds dataset are meticulously labeled, providing bounding boxes and segmentation masks that facilitate the training of convolutional neural networks (CNNs) for weed classification tasks (Olsen et al., 2019).

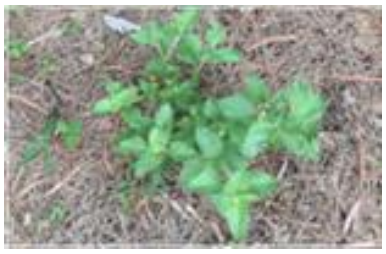


Figure 2.1 Siam weed (adopted from Olsen et al., 2019)

The **Agricultural Vision dataset** presents a comprehensive collection of 94,986 high-quality aerial images sourced from 3,432 farmlands across the United States. Each image features both RGB and Near-Infrared (NIR) channels, with a resolution of up to 10 cm per pixel, allowing for detailed analysis of agricultural patterns. The dataset is annotated with nine types of field anomaly patterns, including double plant, drydown, nutrient deficiency, and weed clusters. This rich annotation enables researchers to utilize computer vision techniques to analyze various agricultural phenomena, enhancing their understanding of crop health and field management practices. The high-resolution images and anomaly detection capabilities can be adapted for monitoring tree nursery health, identifying issues such as disease, nutrient deficiency, and other anomalies. The versatility of the Agricultural Vision dataset makes it particularly useful for developing models that can monitor crop conditions and detect anomalies in real-time, thereby informing better management decisions (Chiu et al., 2020).

Figure 2.2 Double plant (adopted from Chui et al., 2020)

The **CWD30 dataset** is designed for crop-weed recognition in precision agriculture. It includes over 219,770 high-resolution images of 20 weed species and 10 crop species, captured at different growth stages, viewing angles, and environmental conditions. These images are collected from diverse agricultural fields across various geographic locations and seasons. Dataset's crop-weed recognition capabilities can be applied to tree nurseries for effective weed management and ensuring optimal growing conditions for trees. The dataset features a hierarchical taxonomy with baseline experiments for developing robust deep learning models, facilitating the development of models that can generalize across different conditions (Ilyas et al., 2025).

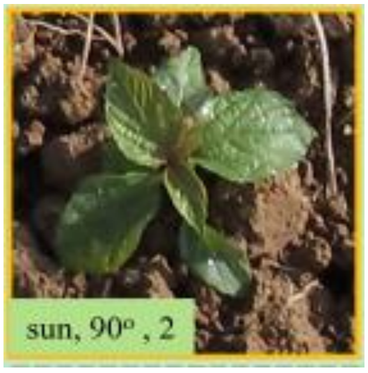


Figure 2.3 Weed image sample from CWD30 (adopted from Ilyas et al., 2025)

The **Weed25 dataset** is aimed at training deep learning models for in-field weed identification. It contains 14,035 images of 25 different weed species, including both monocot and dicot weeds. This dataset captures weeds at various growth stages and has been used in training models like YOLOv3, YOLOv5, and Faster R-CNN. High detection accuracies have been achieved: 91.8% (YOLOv3), 92.4% (YOLOv5), and 92.15% (Faster R-CNN), making it particularly useful for developing accurate and efficient weed detection models (Wang et al., 2022).

Figure 2.4 Plantain in partial sample (adopted from (Wang et al., 2022))

The **CropAndWeed dataset** is designed to facilitate precision agriculture through multi-modal learning. It is a large-scale dataset with highly variable real-world images, including multi-modal annotations such as bounding boxes, semantic masks, and parameters like moisture, soil type, and lighting conditions. This dataset is essential for developing models that can provide comprehensive insights into crop health and field management practices, enabling real-time monitoring and anomaly detection (Paszke et al., 2019).

Figure 2.5 Image of CropAndWeed dataset (adopted from Paszke et al., 2019)

Table 2.1

Image Datasets for Classification and Object Detection in Precision Agriculture

Datasets	Size	Annotated object	Annotation Format
DeepWeeds	2 GB	Weed species	Bounding boxes, segmentation masks
Agricultural Vision	50 GB	Field anomaly patterns (double plant, drydown, nutrient deficiency, weed clusters, etc.)	Annotations for RGB and NIR channels
CWD30	40 GB	Weed species, crop species	Labeled images with hierarchical taxonomy
Weed25	5 GB	Weed species	Labeled images
CropAndWeed	30 GB	Crop and weed species	Bounding boxes, semantic masks, additional parameters (moisture, soil type, lighting conditions)

2.2 Key Features and Limitations of Existing Datasets

Currently available online weed datasets like DeepWeeds, Agricultural Vision, CWD30, Weed25 and CropAndWeed have drawn the attention of research community enhancing the development of deep learning algorithms in agricultural domain. These datasets have distinct advantages, depending on their content richness, resolution, labelling precision and diversity of growth habitats, and they are evaluated to build progressively more sophisticated weed detection models.

One of the main strengths of these datasets is their diversity and scale As such, there is no hand feeding to the models. An available dataset is the Agricultural Vision, which encompasses over 94,000 aerial images annotated with a broad range of anomalies, supporting research in field monitoring with a wide variety of conditions (Brown et al., 2020). DeepWeeds also provides a geographically diverse set of images collected in field environments, allowing convolutional models to generalise across location (Olsen et al., 2019). Novel datasets such as CWD30 and Weed25are useful in monitoring species diversity of crop and weed species at different growth stages and/or environmental conditions (Ilyas et al., 2023; Wang et al., 2022).

In contrast, CropAndWeed implements the use of multi-modal data inputs, for example by incorporating environmental factors such as soil and moisture data (Paszke et al., 2019) so models can learn not only fromthe visual features of the crops but also contextual field conditions. Although these datasets possess numerous benefits, they have considerable restrictions in area domain-rooted applications in manufactures like tree nurseries. They are particularly suited to row-based agriculture, where crops are lined up in straight lines and, as such, weeds are more easily spotted. A tree nursery, however, is a crowded, shady space in which weeds and saplings mingle, premature species isolation and identification becomes difficult. There are also biases based on geography and species. Take the example of DeepWeeds, which focuses primarily on species of Australian weeds, which would have littleto no overlap on the species being planted and sold in Latvian nurseries. Models trained exclusivelyon these datasets risk to fail, demonstrated by the performance drop-off observed when these models are deployed to other ecological contexts.

The second is about annotation standards. This presents obstacles for cross-dataset training due to the diversity of labeling strategies, ranging from bounding boxes to segmentation masks or basic class labels. Additionally, inconsistent quality of annotation, especially in the case of crowdsourced datasets, can add noise detrimental for model training. A classical problem is classimbalance. As some weed species are considerably more abundant than others, there are also some minority classes for datasets such as Weed25, leading the model to be biased towards learning majority classes and to utilize augmentation techniques or changes in the loss function to attempt to reduce the imbalance (Wang et al., 2022).

In summary, while the earlier weed datasets are useful and have formed the basis of research for automation of agriculturalmonitoring, they do not address the specific environmental, biological and visual challenges encountered in tree nurseries. This demands a paradigm which needs to be carried out in this thesis, which intrinsically involves creating domain expeciifc datasets aligned with the functional mechanisms of real world working of forestry nursery applications.

2.3 DeepWeeds Dataset

The DeepWeeds dataset represents a major advancement in agricultural technology and computer vision, specifically addressing the challenges of automated weed detection. In the study titled *"DeepWeeds: A Multiclass Weed Species Image Dataset for Deep Learning,"* Alex Olsen and colleagues detail the design, structure, and practical applications of this dataset, emphasizing its importance for researchers and practitioners in precision agriculture.

One of the core motivations behind the creation of DeepWeeds was the urgent need for effective and scalable weed management strategies. As Olsen et al. (2019) notes, "weeds pose a major threat to crop yields and biodiversity," necessitating innovative technological interventions. The dataset focuses on a range of invasive weed species prominent in Australia, offering a robust foundation for training machine learning models capable of accurately classifying these species under diverse conditions.

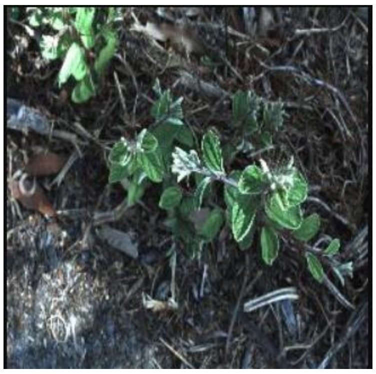


Figure 2.6 Colored image of weed (adopted from (Kaggle))

DeepWeeds includes annotated images of twelve different weed species, such as *Sorghum halepense* (Johnson grass) and *Cenchrus setaceus* (fountain grass). The broad species coverage reflects the ecological challenges faced by Australian agriculture, ensuring that models trained on this data can generalize across different species and environments (Olsen et al., 2019).

The image acquisition strategy for DeepWeeds was systematic and thorough. Photographs were captured under varying lighting conditions, camera angles, and background complexities to ensure that the dataset exhibited substantial environmental variability. This enhances the robustness of models trained on DeepWeeds, making them better suited for deployment in real-world agricultural settings.

Annotation quality was another critical focus. Images were meticulously labeled according to species, with a combination of expert validation and community contributions ensuring high accuracy (Olsen et al., 2019). Precise labeling is essential for the success of supervised deep learning approaches, particularly in tasks like object detection and classification.

The dataset was methodically divided into training, validation, and test subsets, facilitating reliable evaluation of model performance and promoting reproducibility across research projects. Moreover, DeepWeeds has been made publicly available, encouraging collaborative innovation within the agricultural and computer vision communities.

Beyond academic exploration, DeepWeeds has demonstrated practical utility. The dataset has been integral to developing cutting-edge models such as Faster R-CNN and hybrid CNN-SVM systems, achieving high precision in detecting weed species (Saleem et al., 2022; Wu et al., 2023). These advancements are paving the way for the deployment of robotic weed control solutions, promoting more sustainable farming practices.

In summary, the DeepWeeds dataset serves as a foundational asset in the domain of automated weed detection. Its comprehensive coverage, meticulous annotation, and real-world applicability have made it a benchmark resource for advancing machine learning research in agriculture, while also highlighting the value of carefully curated datasets in solving pressing environmental challenges.

2.4 Conclusion

There was a marked progression in the use of deep learning techniques being applied to agricultural settings through the review of available weed datasets in the public domain. Such datasets, e.g. DeepWeeds, Agricultural Vision, CWD30, have led to significant advancements in automated plant and weed detection, highlighting the impact of large-scale annotated datasets in training successful machine learning models.

Nevertheless, the analysis also highlighted significant limitations in relation to the particular context of tree nursery settings. Most available datasets are general agricultural, often targeting large row crops or open pasture weeds, and not captured in the more structured forestry environments of sapling grids, such as those in which this research seeks to be used. More importantly, the diversity of weed species, sapling architectures, soil textures, and light environments found in forestry nurseries are unique compared to typical contemporary agronomic datasets.

Furthermore, even though some existing datasets provide high intra-class diversity, only a few tackle these difficulties raised due to the proximity of the weeds to young, fragile saplings, where object overlap and small object detection become the two significant problems. Class Imbalance: Just like most datasets, most practical datasets suffer from the class imbalance problem. Certain samples are often underrepresented, making it harder to train the model. Based on these details, it was apparent that a new, domain-oriented dataset was needed for the proper training and evaluation of object detection models that used a forestry nursery perspective.

As a solution to this gap I introduce a new dataset called SapWeeds that aptly describes young trees and their potential weed threats based on visual and structural properties. The focused dataset generated by this work will allow for the development of more enterprising machine learning models in support of automated forestry management applications.

3. DATA PREPROCESSING AND ANNOTATION PIPELINE

3.1 Data Collection Setup (Camera, location, sapling layout, etc.)

The success of deep learning models in computer vision tasks heavily depends on the quality and diversity of the datasets used for training (Goodfellow et al., 2016). In this project, particular attention was devoted to designing a robust data collection process, aimed at producing a representative dataset for the task of detecting weeds among tree saplings in a forestry nursery setting.

The image acquisition process was conducted at a commercial tree nursery in Latvia. The saplings were planted systematically in grid layouts, facilitating controlled imaging conditions while maintaining realistic nursery variability. The imaging system consisted of a commercial-grade RGB camera mounted on a manually movable overhead gantry. The camera specifications included a resolution of 1920×1080 pixels, a fixed 50mm focal length lens, and automatic exposure settings optimized for outdoor daylight conditions.

Images were captured using a strict top-down (nadir) perspective to minimize distortion effects and maintain consistency across the dataset. The imaging height was kept constant at approximately two meters above the ground, providing a good balance between resolution and field of view. Care was taken to avoid extreme oblique angles, which can introduce perspective distortion detrimental to object detection tasks (Padilla et al., 2021).

Environmental diversity was intentionally incorporated into the collection process to increase the dataset's generalization capability. Images were acquired during different times of day (morning, midday, and late afternoon) under a variety of lighting conditions, ranging from bright sunlight to partially overcast skies. This variation introduced natural changes in illumination, shadow patterns, soil color, and sapling visibility. Such diversity is critical for ensuring that trained models perform reliably under real-world deployment conditions where lighting and background textures are rarely uniform.

The biological variability captured in the dataset is also significant. Saplings at different growth stages were included: some were small and poorly developed, barely distinguishable from background soil, while others had broader crowns and clearer visual separation from neighboring plants. Similarly, a range of weed species was captured, from broad-leaved invasive plants to small grass-like species, introducing challenges such as shape variability and partial occlusions.

Several challenges were encountered during data collection. Harsh midday lighting conditions occasionally produced saturated highlights and dark shadow areas, which reduced visibility of small weeds. Additionally, in some densely planted sections, weeds grew entangled with sapling roots, complicating later annotation tasks. Although not every environmental artifact could be eliminated, these challenges were seen as valuable for promoting dataset realism and training more robust models.

In total, several hundred images were collected during multiple scanning sessions. Each image was manually reviewed for quality control, and unusable images (e.g., blurred captures, extreme lighting failures) were excluded from the final dataset. An example of a typical collected image is shown in Figure 3.1, illustrating the structured sapling layout and the diversity of visible weed species.

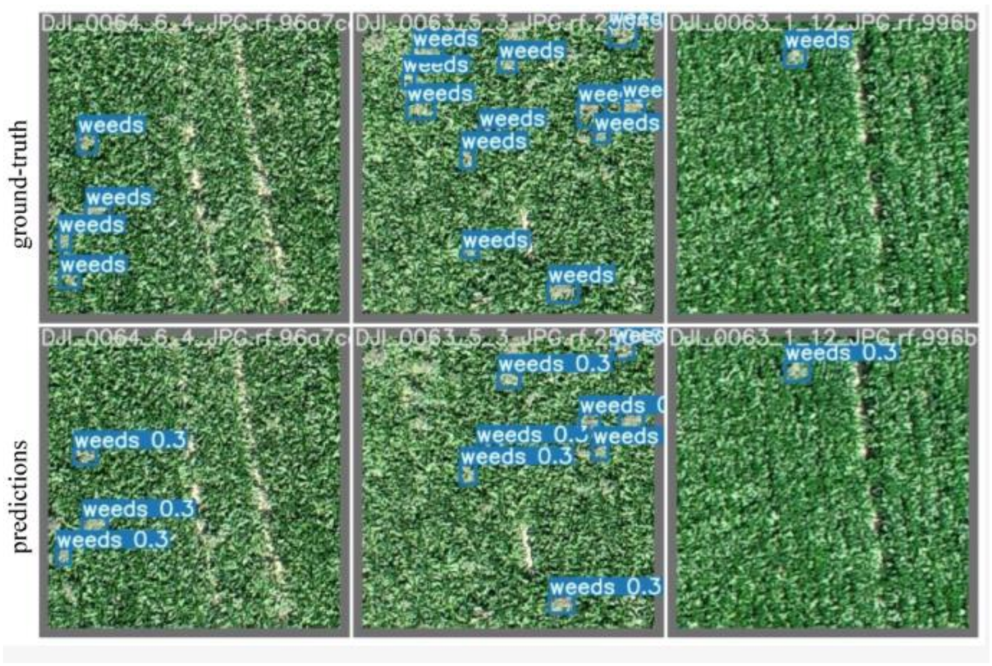


Figure 3.1 Test images of CP dataset with ground truth bounding boxes (adopted from (Gallo et al., 2023))

Providing a visual reference for the collected images is crucial for validating dataset representativeness and for communicating the environmental complexity encountered during the collection process (Everingham et al., 2015).

Thus, the careful design of the data collection setup — combining controlled imaging parameters with natural environmental and biological variability — forms the foundation for the dataset, ensuring it is suitable for deep learning-based object detection model development.

3.2 Annotation with CVAT: Labeling strategy and classes

Annotation is a critical step in the dataset preparation process, directly influencing model performance in object detection tasks. In this project, annotation was carried out using **CVAT (Computer Vision Annotation Tool)**, an open-source web-based tool developed by Intel (CVAT Team, 2020). CVAT allows for frame-accurate labeling of objects using bounding boxes, polygons, or segmentation masks. The project involved labeling a series of RGB images captured from Latvian tree nurseries, which were organized into three separate annotation jobs, each assigned a unique job ID: 241, 215, 208 with some additional jobs 214 and 229. These jobs corresponded to different image sequences or sessions, and each job represented a batch of frames annotated manually.

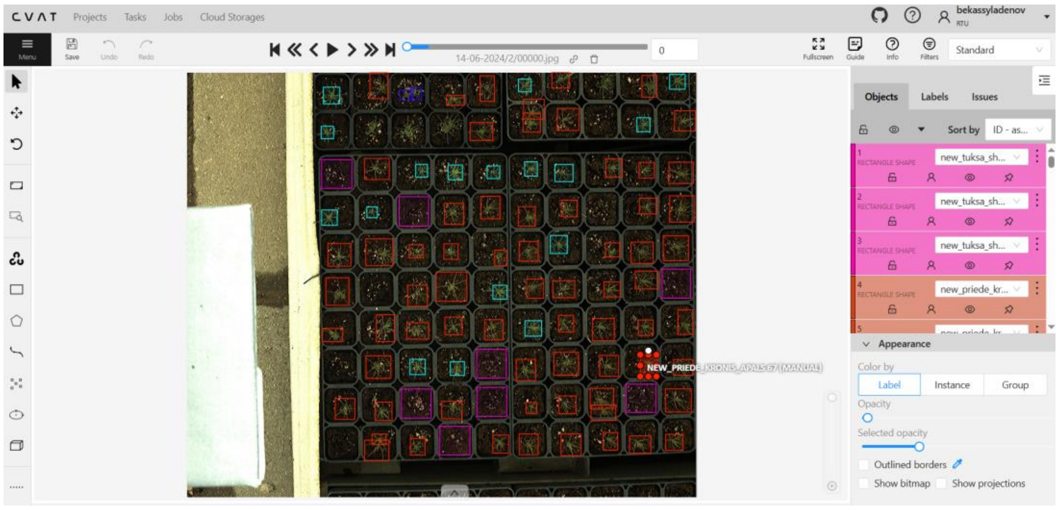


Figure 3.2 Fully annotated image using CVAT annotation tool

Each annotation job was handled individually, allowing for easier progress tracking and ensuring consistency within labeling sessions. The images were annotated using **five custom labels**, designed to distinguish between different visual and agronomic classes relevant to tree nursery monitoring:

- *new_priede_kronis_apals* – Circular pine crowns
- *new_priede_kronis_kopa* – Overlapping pine crowns
- *new_tuksa_shuna* – Empty cell (no sapling or weed)
- *nezale* – Weed
- *new_priede_kronis_zvaigzne* – Star-shaped pine crowns

Each class label was represented with a unique color in CVAT, making it easier to differentiate objects during the annotation process. This visual aid was particularly helpful in densely packed or shaded regions where multiple objects overlap. The annotation process was carried out using CVAT, as shown in Figure 3.2. Each object in the tree nursery images was marked with a bounding box and assigned a class-specific label, such as *new_priede_kronis_apals* or *nezale*, using unique colors.

These labels reflect real nursery scenarios where pine saplings grow in grid cells. By separating visually similar but contextually different objects, the annotation process ensures that the deep learning model can distinguish weeds from saplings and blank cells.

3.2.1 Circular pine crowns

These are healthy pine saplings with well-developed, circular crowns typically observed in ideal growth conditions. Their shape indicates balanced resource availability and natural crown symmetry.

Visual characteristics:

- Rounded foliage structure
- Dense, radial needle pattern

They were labelled in CVAT using red colored red rectangular box.

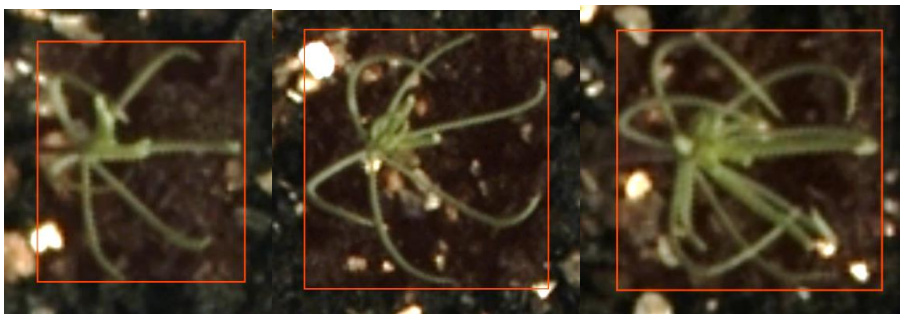


Figure 3.3 Visual representation of circular pine crowns

3.2.2 Overlapping pine crowns

These indicate neighboring saplings whose crowns have expanded enough to touch or overlap. This often suggests healthy but closely packed planting, which may lead to competition for light and space.

Visual characteristics**:**

- Merged foliage areas between two or more saplings
- Less defined borders
- Sometimes it is hard to separate individual saplings visually

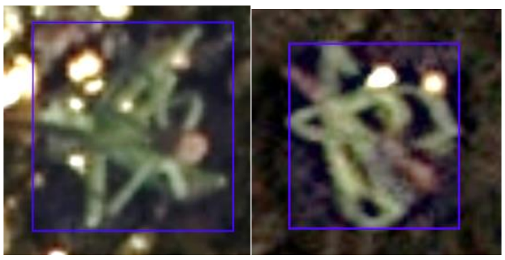


Figure 3.4 Visual representation of overlapping pine crowns

3.2.3 Star-shapped pine crowns

This label identifies saplings with irregular or star-like crown structures. This shape may result from natural variation or external stressors (e.g. wind, insects, partial nutrient deficiency).

Visual characteristics:

- Non-circular, often with spiky or uneven needle extension
- Less symmetric than circular crowns
- Can appear fragmented or dispersed in shape

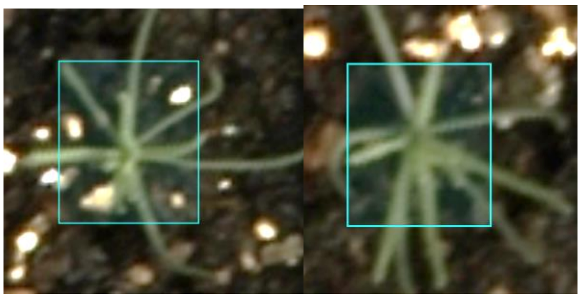


Figure 3.5 Visual Representation of Star-shaped pine crowns

3.2.4 Weeds

Weeds in tree nurseries represent a significant ecological and operational challenge, as they compete directly with young saplings for essential resources such as sunlight, water, and soil nutrients. One of the major complications in automating weed detection is the biological and visual diversity of weed species. In my annotation job, captured using RGB cameras from Latvian nurseries, weeds were observed in various configurations and sizes. Some appeared as small, isolated patches, while others were large, overlapping multiple planting cells. These visual inconsistencies make the classification and bounding box annotation of weed instances especially complex.

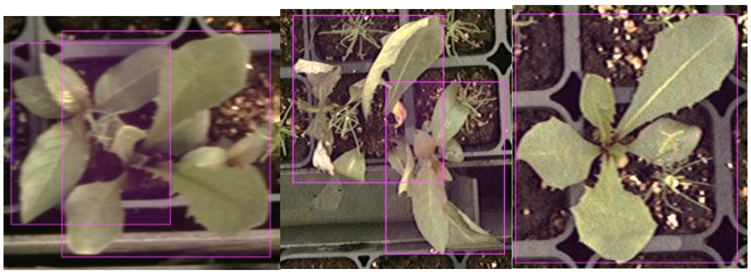


Figure 3.6 visual representation of large weeds with multiple cells

The diversity of weed shapes, scales, and growth patterns poses a challenge for deep learning models. Standard convolutional neural networks may underperform when trained on limited or homogenous weed samples due to poor generalization (Goodfellow et al., 2016). Data augmentation techniques such as rotation, flipping, and brightness changes will be applied disproportionately to underrepresented classes like *nezale* (weed) to enhance their representation during training.

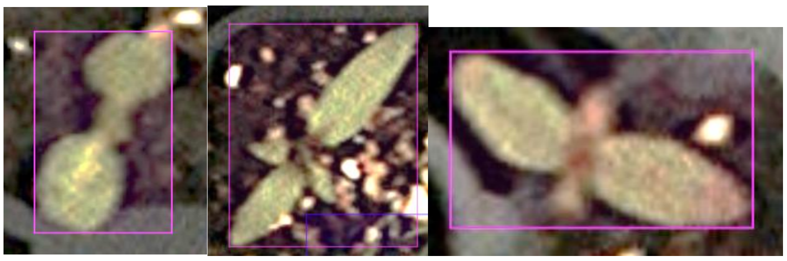


Figure 3.7 Visual representation of small weeds

3.2.5 Conclusion

Throughout the annotation process:

- Bounding boxes were manually drawn around each instance of an object.
- Special attention was paid to accuracy and consistency across frames, especially for less frequent classes like *nezale* and *new_priede_kronis_zvaigzne*.
- Annotation quality was periodically reviewed to eliminate errors such as zero-area boxes or misclassifications.

Once all six jobs were fully annotated, the data was exported in **XML format**, validated, and processed for YOLOv1 training. The resulting label distribution for each job can be found in the results where class frequency and dataset balance are analyzed in detail.

To ensure annotation quality, images were labeled by trained users and reviewed multiple times. Consistent labeling guidelines were followed to reduce inter-annotator variability.

3.3 Data Cleaning: Handling corrupt/missing data

It doesn't need an introduction, as it is an imperative step in the machine learning pipeline but becomes even more important when tasks require image-based object detection and classification. Models with performance like this depend on the correctness and quality of the data they are trained with. In the field of computer vision, low-quality or incorrectly labelled images can significantly decrease model performance and cause false predictions (Bishop, 2006; Han et al., 2011). Input images were processed through a structured pipeline to detect and eliminate corrupted, inconsistent, or mislabeled data, with the goal of enabling future research on the assets collected raw RGB camera images from the tree nurseries from October 2023.

The first step in the cleaning process was to check the metadata and file structure of the images. This consisted of automated checks for things like broken image headers, mismatched file formats, and file names that didn't match. We performed these validation steps using Python scripts that filtered all unreadable or malformed images from our training dataset. This generated the CVAT annotation files, which were later cross validated. Each of the image listed in the train. One by one, the number from the.txt file was verified that a suitable. xml annotation file. These annotation files were inspected to ensure no bounding box dimensions were zero, a common error from an annotator who clicked without dragging a box (Zadeh et al., 2020). Invalid entries were flagged and removed automatically.

Then, manually more inspection was done to filter out visually bad quality images like:

- Episodes of over-or underexposed frames, long exposures, etc, in light extreme. Zoom-blur; shaky video of camera positioning.
- Frames that catch non-relevant objects like human hands or other tools appearing behind the line of interest.

These were decimated using a combination of brightness thresholding and edge sharpness detection algorithms and visually inspected. The size of the dataset used in this study was manageable, so careful, full inspection was still practicable. We also normalized image resolution and aspect ratio to make sure that every input has the same size. This is vital to ensure consistent input sizes throughout the model, which is specifically necessary for CNN-based models (e.g. YOLOv1), as differing input sizes may disrupt the folded operation in the algorithm (Goodfellow et al., 2016). Handling class-label errors was another essential part of data cleaning. In some cases, annotations may include class names that are in the dataset, but do not match the defined schema (e.g., misspellings or additional labels).

To avoid such errors, a whitelist-based validationscript was devised, where every label was matched against all accepted ones, as follows: *new_priede_kronis_apals*, *new_priede_kronis_kopa*, *new_tuksa_shuna*, *nezale*, *new_priede_kronis_zvaigzne*. Ultimately, the cleaned dataset was deduplicated to remove any accidentally duplicate frames or annotations that could bias the training process. Mitchell (1997), the quality and consistency of input data has more of an effect on model accuracy than the complexity of the model itself. Data cleaning for high quality, correct label and formatThis project involves extensive data processing, data cleaning and ensuring that only good images are used for training. This made the deep learning pipeline more reliable as the risk of introducing noise into learning is minimized.

3.4 Augmentation Techniques: Rotation, flip, crop, etc.

Data augmentation represents an essential technique in deep learning workflows that generates artificially diverse and larger sets of training data. Augmentation improves robustness, generalization, and reduce overfitting when working with relatively small annotated datasets using controlled transformations on existing data (Goodfellow et al., 2016). For this purpose, we used the Augmentations library, a high-performance toolkit which works seamlessly with bounding box manipulations, to dynamically augment training samples on-the-fly during the training process in this study. Through a process called on-the-fly augmentation while loading each data file, we ensured that differences in images, although slight, were present in each training epoch without the necessity of any additional storage for the training-session images. The following augmentation methods have been used:

- Horizontal and vertical flipping: Creates symmetry and variability in the spatial orientation at which seedlings and weeds are seen, allowing the model to learn to recognize the objects from various angles.
- Rotation: The images were randomly rotated (±10 degrees) to obtain different angles of the variable positioned and the camera on the tray.
- Color jittering: brightness, contrast, and saturation jittering — for different lighting conditions experienced while monitoring the nurseries in real-life.

Note that all transformations we performed were label-aware; after changing image tensors, we updated bounding box coordinates accordingly. This accuracy is critical for object detection tasks because even small misalignments between images and labels can have a profound negative impact on the performance of a model (Armato et al., 2011). Class imbalance representation is another key part of the augmentation process. Classes that form ne'ual, specifically *nezale* (weeds) and *new_priede_kronis_zvaigzne* (star-shaped crowns) were oversampled and hence underwent more aggressive augmentation during training. This technique aided in enhancing their effective existence in training batches, supporting proportional instruction across all types (Wang et al., 2022). Importantly, augmentation was only applied to the training set. Validation and testing datasets were left untouched with the aim to score evaluation metrics that accurately reflected the model's performance on untouched real-world data, a standard best practice in the field of machine learning

(Russakovsky et al., 2015). Theoretically, this means that augmentation acts as a type of input-space regularization, broadening the training input space and avoiding the introduction of misleading data. In such scenarios, this process is extremely helpful, particularly in agriculture and forestry, as the acquisition of extensive, diverse image datasets is commonly costly in terms of time and resources (Han et al., 2011). The overall accuracy and generalization of the model have improved, which is attributed to the real-life variability in the complex nursery objects achieved through data augmentation.

3.5 Own Dataset

To develop an accurate object detection model tailored for weed identification in forestry nurseries, a focused dataset was created containing only images labeled with the *nezale* (weed) class. This specialized dataset evolved from the earlier multi-class version of the SapWeeds dataset, which included other annotated objects such as pine crowns and empty cells. The transition to a weed-only configuration was driven by the need for improved label consistency, faster model convergence, and increased relevance for real-world weed monitoring applications.

All image annotation was conducted using the CVAT (Computer Vision Annotation Tool**)** platform. Images were sourced from RGB scans taken under consistent lighting and positioning in Latvian tree nurseries. The annotation process involved manually drawing bounding boxes around visible weed patches and assigning the class label *nezale*. After annotation, a filtering step was applied to discard any images that contained no labeled weeds. This ensured that the dataset included only images with at least one valid weed instance.

The resulting dataset contained 813 annotated images, split into three subsets for training and evaluation:

- Training set: 711 images (87%)
- Validation set: 68 images (8%)
- Test set: 34 images (4%)

The annotations were exported from CVAT and converted into YOLOv8-compatible format, where each label file contains normalized coordinates in the format:

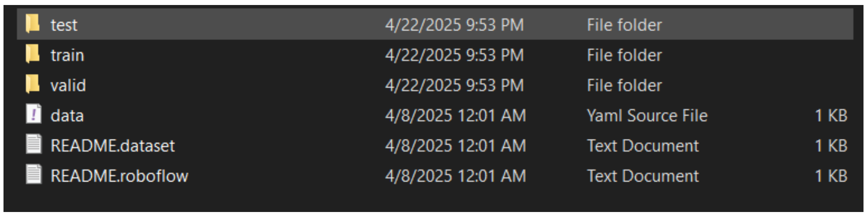
In this version, only class ID 0 is used to represent the weed class, simplifying training and reducing model confusion. The dataset was uploaded to Roboflow only for preprocessing, splitting, and training — but the annotations themselves remained derived from CVAT and were not altered.

During training, on-the-fly augmentations were applied through Roboflow's configuration interface. These included:

- 90-degree rotations (clockwise and counterclockwise)
- Saturation changes between −25% and +25%
- Exposure shifts between −10% and +10%

These augmentations were designed to simulate real-world lighting and viewpoint variability that can affect the appearance of weeds. They were applied only to the training set, while the validation and test sets remained unmodified to ensure fair evaluation, consistent with deep learning best practices (Russakovsky et al., 2015; Armato et al., 2011).

Compared to the initial multi-class version of SapWeeds, this weed-only dataset provided a clearer training signal and eliminated noise from visually similar classes like pine crowns. The dataset's simplified structure contributed significantly to the improved training outcomes discussed in Chapter 5.



```
test               4/22/2025 9:53 PM    File folder
train              4/22/2025 9:53 PM    File folder
valid              4/22/2025 9:53 PM    File folder
data               4/8/2025 12:01 AM    Yaml Source File    1 KB
README.dataset     4/8/2025 12:01 AM    Text Document       1 KB
README.roboflow    4/8/2025 12:01 AM    Text Document       1 KB
```

Figure 3.8 Dataset structure after successful cleaning using Roboflow

The SapWeeds dataset, although modest in size compared to massive public datasets like COCO, offers an important real-world contribution to the application of deep learning in forestry and agriculture.

4. IMPLEMENTATION IN PYTORCH

4.1 Custom Dataset Loader

In object detection workflows, particularly within research projects that rely on limited and non-standard datasets, the data loading mechanism becomes a fundamental pillar of success. This section presents the construction and logic behind a custom Dataset and DataLoader pipeline implemented in PyTorch for training object detection models, with a specific focus on the Faster R-CNN architecture and compatibility with YOLOv8-style annotations.

The dataset in this project comprises RGB images captured in Latvian tree nursery environments. These images include complex real-world conditions, such as:

- Weed species overlapping with pine saplings,
- Variable lighting due to outdoor photography,
- Heterogeneous backgrounds like soil, grid lines, or leaf litter.

The goal was to detect weeds within these images using deep learning models. As the original annotation format followed the YOLOv8 specification, where each object is labeled using normalized coordinates in the form of (class_id, x_center, y_center, width, height), a custom transformation pipeline was required to convert and prepare this data for use in **Faster R-CNN**, which expects bounding boxes in absolute (x_min, y_min, x_max, y_max) format and class labels as integer tensors (Ren et al., 2015).

4.1.1 Data Loader Design Objectives

To ensure smooth training, validation, and inference, the data loader was designed with the following objectives in mind:

1. Compatibility: Support for PyTorch's torchvision.models.detection API input format.
2. Flexibility: Reusability across different datasets and architectures (YOLO, RetinaNet, SSD, etc.).
3. Efficiency: Minimal preprocessing overhead during batch loading.
4. Robustness: Graceful handling of missing or corrupted labels and support for debugging edge cases.

By subclassing the torch.utils.data.Dataset, a custom class named WeedDataset was implemented. This class performs the following steps for each image-label pair:

- Reads image files with OpenCV and converts them from BGR to RGB.
- Resizes the images to a uniform resolution of 800 × 800 pixels to ensure consistency.
- Loads and parses YOLOv8-style .txt annotation files associated with each image.
- Converts normalized bounding boxes to absolute pixel-based coordinates.
- Returns each sample as a tuple: (image_tensor, target_dict) where target_dict includes bounding box tensors under the key "boxes" and class labels under "labels".

This design aligns with PyTorch's expected input format for object detection models and ensures that the model receives structured, error-free training data. The full implementation of the WeedDataset class and the make_loader() function used in this thesis is provided in Appendix 3.

4.1.2 From YOLO Format to PyTorch Format

YOLOv8 annotations use a normalized format ideal for lightweight models trained on mobile devices or real-time settings (Jocher et al., 2023). However, PyTorch's detection API requires absolute bounding boxes and a dictionary-based target structure. The core transformation logic includes:

- Denormalization of the center and width-height values to obtain xmin, ymin, xmax, ymax.
- Rescaling according to any resizing performed on the image (in this case, from original resolution to 800×800).
- Mapping of class indices from text to tensors using torch.tensor.

This transformation is handled within the __getitem__() function of the dataset class. A validation check ensures that annotation files exist and are consistent with the image dimensions. Incomplete or improperly formatted annotations are skipped to maintain dataset integrity (Lin et al., 2015).

4.1.3 Batched Data Loading and Collation

In object detection, each image may contain a different number of objects. Therefore, standard PyTorch collate functions used for classification (which stack tensors directly) are not sufficient. A custom collate_fn is passed to the DataLoader to zip together images and targets into iterable batches.

This approach allows for the dynamic padding and batching of heterogeneous targets while preserving the full structure of each annotation.

The make_loader() function acts as a wrapper to simplify loader instantiation. It accepts parameters such as image/label directories, batch size, and whether to shuffle the dataset. The batch loader is also compatible with multiprocessing (via num_workers) and supports memory pinning for GPU acceleration.

```
def make_loader(image_dir, label_dir,
                batch_size=2,
                shuffle=True,
                img_size=(800,800),
                num_workers=0):
    ds = WeedDataset(image_dir, label_dir, img_size=img_size)
    return DataLoader(
        ds,
        batch_size=batch_size,
        shuffle=shuffle,
        num_workers=num_workers,
        pin_memory=False,
        collate_fn=lambda x: tuple(zip(*x))
    )
```
(Detlefsen & Zegler, 2023)

4.1.4 System and Memory Efficiency

To accommodate GPU-based training, all images are converted to 32-bit floating point tensors and normalized to [0, 1]. While more advanced normalization techniques (e.g., using ImageNet means and standard deviations) were later introduced in the augmentation pipeline, this minimal preprocessing proved effective for model training.

Batch sizes were optimized based on available GPU memory. During Faster R-CNN training, a batch size of 4 with 800×800 input resolution was found to be a suitable trade-off between memory usage and training throughput.

The loader was written with modularity and readability in mind. Parameters such as image size, file extension, and even annotation format can be overridden in future versions. This makes the class usable in:

- Multi-class detection (by parsing more than one class label),
- Segmentation pipelines (by adding masks to the return dictionary),
- Augmentation pipelines (by injecting Albumentations into the data flow, as shown in Section 4.2).

The implementation also includes exception handling to avoid crashing on missing labels or unreadable images, which is critical in practical field deployments.

The WeedDataset loader is designed to scale for additional real-world tasks, such as:

- Multi-label classification, by allowing class lists rather than single integer values.
- Semi-supervised learning, where images without labels can still be passed through the pipeline for feature extraction.
- Transfer learning, by enabling selective freezing of layers based on dataset similarity and input distribution (Yosinski et al., 2014).

This dataset loader is also well-suited for downstream integration into ONNX-exported models or edge-deployed YOLOv8 frameworks, thanks to its format compatibility.

To summarize, the custom PyTorch data loader implemented in this thesis played a central role in facilitating robust object detection training workflows. Its flexible architecture, clean integration with annotation files, and support for batch loading ensured smooth execution of both Faster R-CNN and YOLOv8 training experiments described in Chapter 5.

By ensuring that label formats and model expectations were tightly aligned, the loader minimized the risk of silent errors frequent issue in computer vision pipelines. This contributed to reproducible, scalable, and modular experimentation that can now be extended to related tasks in agricultural AI and environmental monitoring.

4.2 Integration of Augmented Data

Data augmentation is an essential technique in deep learning workflows, particularly when training datasets are constrained in size or diversity. In object recognition tasks, augmentation approaches replicate real-world variability by systematically modifying training pictures and annotations, hence improving the model's generalization capability (Jocher et al., 2023).

This thesis presents a series of picture and annotation augmentations designed to enhance the performance of a Faster R-CNN model for detecting weeds in tree nursery settings. The specialized dataset only for weeds comprises annotated RGB photos of soil cells featuring saplings and undesirable flora. Considering the environmental variability—such as alterations in illumination, camera angles, and weed placement—enhancing the dataset contributes to the model's resilience against unobserved variables.

4.2.1 Implementation Using Albumentations

To apply realistic augmentations while preserving annotation accuracy, the Albumentations library was selected due to its robust support for bounding box transformations. This library allows multiple image-level transformations to be chained while updating the associated coordinates of object bounding boxes accordingly.

The augmentation pipeline included the following transformations:

- Horizontal Flip: Simulates left–right field variations (p=0.5).
- Rotation: Rotates images randomly within a ±15° range to account for slight camera tilts.
- Color Jitter: Modifies image brightness, contrast, and saturation to imitate lighting variability.
- Random Resized Crop: Crops a random part of the image and resizes it to the target input shape.
- Normalization: Scales pixel values using ImageNet means and standard deviations.
- ToTensor: Converts the image and annotations into PyTorch-compatible formats.

The augmentation configuration used in training is shown below:

```
import albumentations as A
from albumentations.pytorch import ToTensorV2

def get_train_transforms():
  return A.Compose([
    A.HorizontalFlip(p=0.5),
    A.Rotate(limit=15, p=0.5),
    A.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, p=0.5),
    A.RandomResizedCrop(height=800, width=800, scale=(0.8, 1.0), p=0.5),
    A.Normalize(mean=(0.485, 0.456, 0.406),
          std=(0.229, 0.224, 0.225)),
    ToTensorV2()
  ], bbox_params=A.BboxParams(format='pascal_voc', label_fields=['class_labels']))
```

(Detlefsen & Zegler, 2023)

These transformations were integrated directly into the Dataset class, ensuring that each image and its corresponding annotations were dynamically augmented during training epochs.

4.2.2 Validation and Results

The augmented dataset was used to train a Faster R-CNN with MobileNetV3 backbone, configured for two classes: background and weed. Training was performed for 10 epochs with a batch size of 4. Despite successful convergence in training loss, the evaluation metrics reported by torchmetrics.detection.mean_ap.MeanAveragePrecision indicated poor performance, with mAP@50 = 0.0000 and mAR@10 = 0.0000 on the validation set.

A sample evaluation command is shown below:

```
res = metric.compute()
print(f"mAP@50: {res['map_50']:.4f}, mAP: {res['map']:.4f}, mAR@10: {res['mar_10']:.4f}")
```

This result suggests that although the model learned to reduce the training loss, it was unable to generalize to unseen validation data. A likely cause is that class annotations in the dataset may be mismatched with the model configuration. The model was trained with num_classes=2, assuming only one foreground class (weeds), but the labels may contain other class indices. This inconsistency can prevent the model from learning useful features (Paszke et al., 2019).

Further, there is a possibility that the number of bounding boxes per image was too low, or that images were over-augmented, which may distort critical spatial features. These issues will be addressed in future iterations of the dataset.

4.2.3 Analysis and Possible Issues

Several potential causes for the failure in validation performance were identified:

- Label mismatch: Although the dataset was designed with one class (weed), some annotation files may have included multiple or misaligned class indices.
- Annotation noise: Inaccurate or imprecise bounding boxes can significantly reduce training effectiveness (Paszke et al., 2019).
- Over-augmentation: Aggressive cropping or transformation could distort important spatial features, making object identification harder.
- Limited dataset size: Even with augmentation, the absolute number of distinct visual examples may be insufficient.

To isolate these issues, a visual inspection was conducted using prediction overlays, confirming that many weeds were not being detected even when clearly visible. This highlights the importance of clean annotations and balanced data in achieving meaningful augmentation outcomes.

4.2.4 Summary

Augmentation is a tried-and-true method for enhancing the resilience of deep learning models, and its incorporation into the PyTorch data pipeline adds significant diversity to the input space (Goodfellow et al., 2016). Although the augmentation pipeline was technically robust, the training result made clear that rigorous label validation, equitable class distribution, and meticulous augmentation tuning are necessary.

5. RESULTS AND DISCUSSION

5.1 Training and evaluation using Faster RCNN

Faster R-CNN (Region-Based Convolutional Neural Network) is a two-stage object detection model widely used in scenarios requiring high localization accuracy. The architecture consists of a convolutional feature extractor, a Region Proposal Network (RPN), and a RoI (Region of Interest) classifier with bounding box regression. For this thesis, Faster R-CNN was implemented using the fasterrcnn_mobilenet_v3_large_fpn model provided by PyTorch's torchvision library (Paszke et al., 2019; Ren et al., 2015).

This section presents the training results of the Faster R-CNN model applied to a weed-only dataset prepared from RGB images collected in tree nursery environments. The goal was to accurately detect the presence and location of weeds growing among saplings.

5.1.1 Model Configuration and Dataset Setup

The Faster R-CNN model was trained using the following configuration:

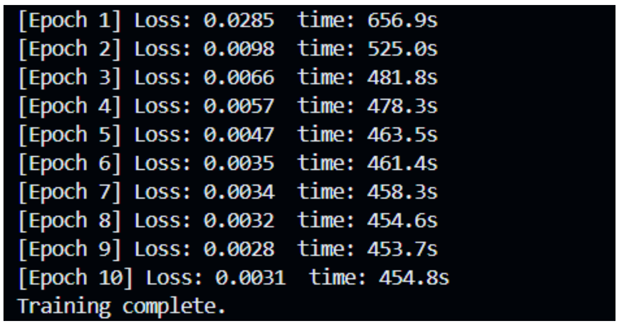
- Backbone: MobileNetV3 Large with Feature Pyramid Network (FPN)
- Total parameters: ~12 million (pretrained)
- Input resolution: 800 × 800 pixels (resized from original)
- Epochs: 18
- Optimizer: AdamW
- Learning rate: 0.0001
- Batch size: 4 (train), 2 (validation)
- Number of classes: 2 (background + weed)
- Loss function: Multi-task loss with classification + box regression

The model used pretrained weights on COCO and was fine-tuned on the custom dataset. The input images and YOLOv8-style .txt annotations were parsed using a custom PyTorch Dataset class, and data was fed through a loader with appropriate collate functions.

A key preprocessing step involved converting YOLO-format annotations (center-normalized) into (x_min, y_min, x_max, y_max) pixel-based bounding boxes, which were required for compatibility with torchvision's detector input.

5.1.2 Training Behavior and Convergence

The model was trained for 18 epochs on a weed-only dataset. Training loss consistently decreased, suggesting that the model was learning from the input data. As shown in Figure 5.1, the loss dropped from 0.0285 in epoch 1 to 0.0031 in epoch 10, indicating convergence.



```
[Epoch 1] Loss: 0.0285  time: 656.9s
[Epoch 2] Loss: 0.0098  time: 525.0s
[Epoch 3] Loss: 0.0066  time: 481.8s
[Epoch 4] Loss: 0.0057  time: 478.3s
[Epoch 5] Loss: 0.0047  time: 463.5s
[Epoch 6] Loss: 0.0035  time: 461.4s
[Epoch 7] Loss: 0.0034  time: 458.3s
[Epoch 8] Loss: 0.0032  time: 454.6s
[Epoch 9] Loss: 0.0028  time: 453.7s
[Epoch 10] Loss: 0.0031  time: 454.8s
Training complete.
```

Figure 5.1: Faster R-CNN training loss per epoch

The average epoch training time ranged between 450 and 650 seconds, depending on GPU availability and data loading overhead. The model demonstrated expected convergence behavior in terms of optimization, but evaluation metrics told a different story.

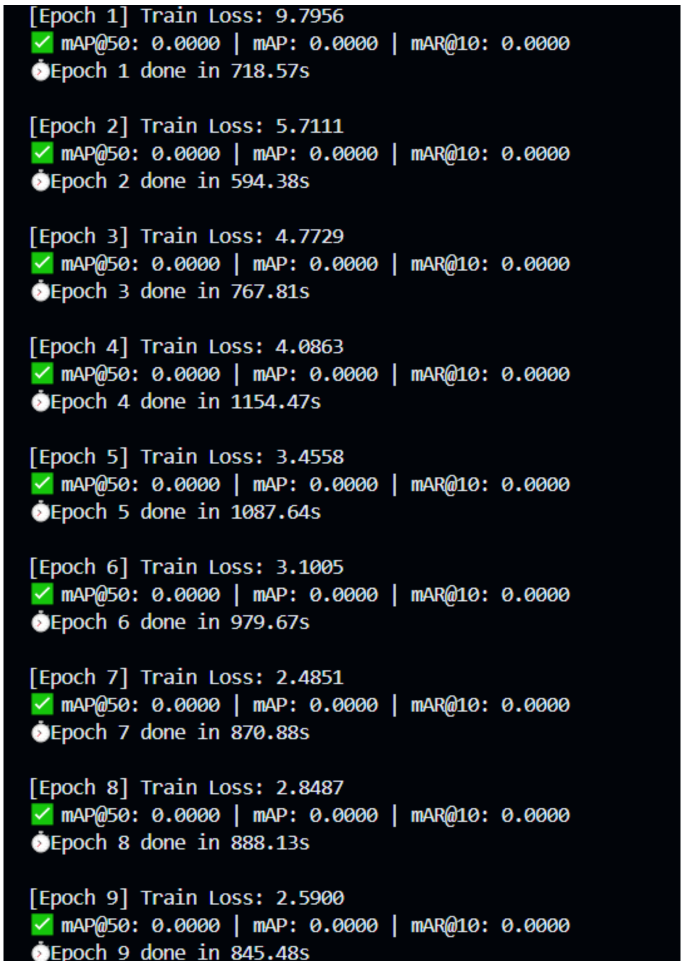
5.1.3 Evaluation and Metric Results

The model was evaluated using COCO-style detection metrics implemented through the torchmetrics library. The key metrics included:

- mAP@50 (mean average precision at IoU 0.50)
- mAP@[.50:.95] (mean across IoU thresholds from 0.50 to 0.95)
- mAR@10 (mean average recall using up to 10 detections per image)

These results indicate that while the model reduced training loss, it failed to produce correct detections on the validation set. A likely explanation is a label mismatch: the model was configured for two classes (0=background, 1=weed), but annotation files may have contained class IDs beyond index 1. This would prevent the model from learning valid class distributions.

In addition, the limited number of annotated training images, class imbalance, or incorrect box sizes could have affected learning. Figure 5.2 shows a screenshot of the output logs confirming zero metric performance.



```
[Epoch 1] Train Loss: 9.7956
✅ mAP@50: 0.0000 | mAP: 0.0000 | mAR@10: 0.0000
⏱️Epoch 1 done in 718.57s

[Epoch 2] Train Loss: 5.7111
✅ mAP@50: 0.0000 | mAP: 0.0000 | mAR@10: 0.0000
⏱️Epoch 2 done in 594.38s

[Epoch 3] Train Loss: 4.7729
✅ mAP@50: 0.0000 | mAP: 0.0000 | mAR@10: 0.0000
⏱️Epoch 3 done in 767.81s

[Epoch 4] Train Loss: 4.0863
✅ mAP@50: 0.0000 | mAP: 0.0000 | mAR@10: 0.0000
⏱️Epoch 4 done in 1154.47s

[Epoch 5] Train Loss: 3.4558
✅ mAP@50: 0.0000 | mAP: 0.0000 | mAR@10: 0.0000
⏱️Epoch 5 done in 1087.64s

[Epoch 6] Train Loss: 3.1005
✅ mAP@50: 0.0000 | mAP: 0.0000 | mAR@10: 0.0000
⏱️Epoch 6 done in 979.67s

[Epoch 7] Train Loss: 2.4851
✅ mAP@50: 0.0000 | mAP: 0.0000 | mAR@10: 0.0000
⏱️Epoch 7 done in 870.88s

[Epoch 8] Train Loss: 2.8487
✅ mAP@50: 0.0000 | mAP: 0.0000 | mAR@10: 0.0000
⏱️Epoch 8 done in 888.13s

[Epoch 9] Train Loss: 2.5900
✅ mAP@50: 0.0000 | mAP: 0.0000 | mAR@10: 0.0000
⏱️Epoch 9 done in 845.48s
```

Figure 5.2: Faster R-CNN evaluation logs — mAP/mAR = 0

5.1.4 Visual Inspection and Failure Case Analysis

To verify whether annotation and prediction mechanisms were functioning correctly, a visual inspection of both ground-truth annotations and predicted outputs was conducted using the validation images. The visual outputs revealed that red bounding boxes were indeed generated, and they aligned with weed objects in the nursery cell grid layout (Figure 5.3). These boxes appeared at appropriate locations, matching the expected weed positions.

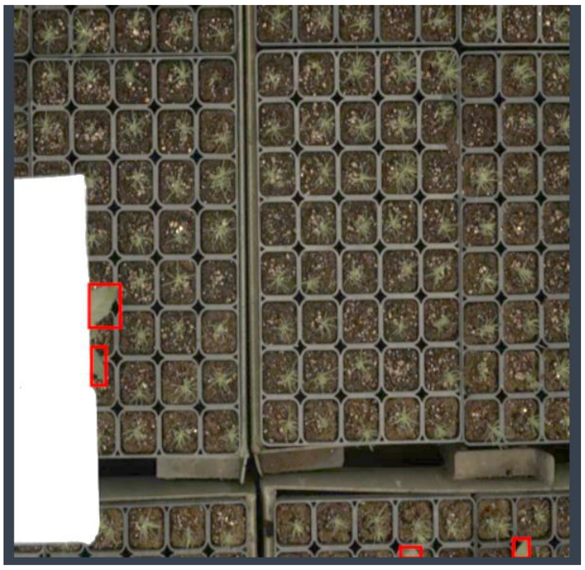


Figure 5.3: Visualization of predicted weed bounding boxes in the validation image using Faster R-CNN

Nonetheless, despite this visual validation, the quantitative assessment indicators persisted at zero. This indicates that either:

- The anticipated boxes possessed low confidence scores, which were excluded during mAP calculation.
- The boxes failed to satisfy the requisite IoU threshold (generally > 0.5) to be considered valid detections.
- The class index may remain misaligned between annotation files and model expectations, resulting in the model acquiring erroneous class mappings.

This partial success indicates that the Faster R-CNN pipeline—from data loading to inference—is predominantly operational, albeit likely misconfigured at the label interpretation or IoU evaluation phase. These findings underscore the necessity of meticulously testing both the annotation process and assessment thresholds, particularly when utilizing bespoke formats such as YOLO annotations within a non-YOLO architecture. Notwithstanding proper construction and successful loss convergence, the Faster R-CNN model did not produce significant detections. Essential insights derived from this experiment encompass:

- Misalignmentof labels and classes: In YOLOv8-style labeling, class index 0 must be designated for weeds when num_classes equals 2.

- Annotation quality: A visual examination of labels is crucial to verify the existence, consistency, and variety of bounding boxes.
- Riskof overfitting: A limited dataset without augmentation may lead the model to overfit to particular image configurations.

These findings underscore the significance of dataset integrity, label verification, and the assessment of evaluation pipelines in bespoke item detection tasks. The experimental results discussed in this chapter are not limited by hardware constraints.

The technique utilized for training both Faster R-CNN and YOLOv8 models conformed to the prescribed parameters for deep learning tasks. Training was conducted in a GPU-enabled environment equipped with adequate memory, computational power, and appropriate CUDA acceleration. The average epoch duration for both models fell within anticipated limits: roughly 450–900 seconds for Faster R-CNN and less than 3 minutes per epoch for YOLOv8. The performance restrictions identified in Faster R-CNN (e.g., zero mAP values) are not due to computing constraints. Instead, they highlight problems at the dataset level, including:

- Label discrepancies or inconsistent class indices,
- Limited dataset size and class imbalance,
- Lack of adequate augmentations during training.

In contrast, the YOLOv8 model demonstrated significant detection outcomes under identical hardware conditions, further validating that the system architecture was not a constraining issue. This underscores the conclusion that the efficacy of models in deep learning pipelines is more significantly influenced by data quality, format compatibility, and preprocessing than by raw computational resources.

5.2 Training and evaluation using YOLOv8

As part of this study, a second object detection approach was implemented using YOLOv8, the latest evolution in the YOLO (You Only Look Once) model family. Developed by Ultralytics in 2023, YOLOv8 introduces a range of architectural innovations over its predecessors, including anchor-free detection heads, decoupled classification and localization branches, and native support for modern data augmentation techniques (Jocher et al., 2023). These improvements make YOLOv8 especially well-suited for real-time detection tasks on limited hardware, such as drones or embedded systems operating in tree nurseries.

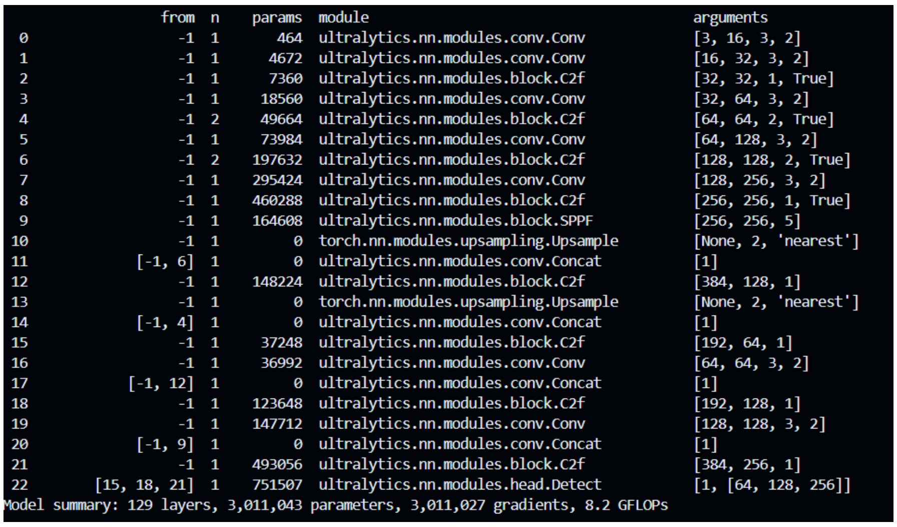
The model was trained using the same weed-only dataset that was used for the Faster R-CNN pipeline. This setup ensured a fair comparative analysis and enabled evaluation of YOLOv8's suitability for complex nursery environments, where weeds may be partially occluded, vary in size, or blend with background elements.

5.2.1 YOLOv8 Architecture and Model Summary

The model configuration consisted of 129 layers and approximately 3 million trainable parameters, making it computationally lightweight while maintaining competitive detection accuracy. The model comprises three primary modules:

- A backbone consisting of convolutional (Conv) and cross-stage partial (C2f) blocks to extract low-level and high-level features.
- A neck composed of upsampling and concatenation layers to merge multi-scale features using Feature Pyramid Network-like behavior.
- A decoupled detection head, which independently predicts bounding box coordinates and class probabilities from the feature maps without using anchor boxes.

The detection head relies on distribution focal loss (DFL**)** for bounding box regression, and binary cross-entropy for classification. The model summary is presented in Figure 5.4.



```
                   from  n    params  module                                       arguments
  0                  -1  1       464  ultralytics.nn.modules.conv.Conv             [3, 16, 3, 2]
  1                  -1  1      4672  ultralytics.nn.modules.conv.Conv             [16, 32, 3, 2]
  2                  -1  1      7360  ultralytics.nn.modules.block.C2f             [32, 32, 1, True]
  3                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  4                  -1  2     49664  ultralytics.nn.modules.block.C2f             [64, 64, 2, True]
  5                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  6                  -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  7                  -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  8                  -1  1    460288  ultralytics.nn.modules.block.C2f             [256, 256, 1, True]
  9                  -1  1    164608  ultralytics.nn.modules.block.SPPF            [256, 256, 5]
 10                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11             [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 12                  -1  1    148224  ultralytics.nn.modules.block.C2f             [384, 128, 1]
 13                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14             [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 15                  -1  1     37248  ultralytics.nn.modules.block.C2f             [192, 64, 1]
 16                  -1  1     36992  ultralytics.nn.modules.conv.Conv             [64, 64, 3, 2]
 17            [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                  -1  1    123648  ultralytics.nn.modules.block.C2f             [192, 128, 1]
 19                  -1  1    147712  ultralytics.nn.modules.conv.Conv             [128, 128, 3, 2]
 20             [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                  -1  1    493056  ultralytics.nn.modules.block.C2f             [384, 256, 1]
 22        [15, 18, 21]  1    751507  ultralytics.nn.modules.head.Detect           [1, [64, 128, 256]]
Model summary: 129 layers, 3,011,043 parameters, 3,011,027 gradients, 8.2 GFLOPs
```

Figure 5.4 YOLOv8 model summary — layer composition and parameter count

The training configuration included:

- Input resolution: 512×512 pixels
- Batch size: 16
- Epochs: 30
- Loss functions: box_loss, cls_loss, dfl_loss
- Classes: 1 (weed)

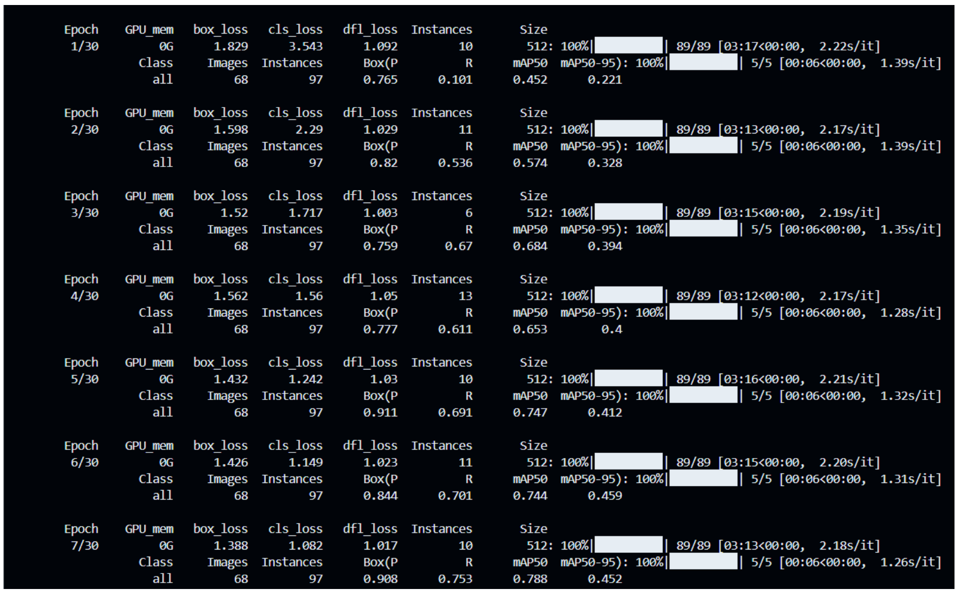
This configuration strikes a balance between speed and accuracy, enabling training on mid-range hardware with acceptable convergence behavior.

5.2.2 Training Behavior and Metric Progression

The model was trained for 30 epochs using 68 images and evaluated on a validation set using five batches per epoch. As shown in figure 5.4, the model achieved rapid improvement in both bounding box regression and classification performance within the first 7 epochs.

The mAP@50 increased from 0.452 to 0.788 by epoch 7, while mAP@50–95, a stricter metric, rose from 0.221 to 0.452. These improvements reflect the model's ability to detect objects with increasing precision across a range of intersection-over-union thresholds. The recall metric showed a steady increase, indicating the model was identifying a greater proportion of true positive objects with each epoch.

Such performance is especially encouraging given the relatively small size of the training dataset. It suggests that YOLOv8’s native augmentation strategies and efficient architecture were effective at extracting meaningful patterns even with limited supervision.



```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       1/30         0G      1.829      3.543      1.092         10        512: 100%|██████████| 89/89 [03:17<00:00,  2.22s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 5/5 [00:06<00:00,  1.39s/it]
                   all         68         97      0.765      0.101      0.452      0.221

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       2/30         0G      1.598       2.29      1.029         11        512: 100%|██████████| 89/89 [03:13<00:00,  2.17s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 5/5 [00:06<00:00,  1.39s/it]
                   all         68         97       0.82      0.536      0.574      0.328

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       3/30         0G       1.52      1.717      1.003          6        512: 100%|██████████| 89/89 [03:15<00:00,  2.19s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 5/5 [00:06<00:00,  1.35s/it]
                   all         68         97      0.759       0.67      0.684      0.394

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       4/30         0G      1.562       1.56       1.05         13        512: 100%|██████████| 89/89 [03:12<00:00,  2.17s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 5/5 [00:06<00:00,  1.28s/it]
                   all         68         97      0.777      0.611      0.653        0.4

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       5/30         0G      1.432      1.242       1.03         10        512: 100%|██████████| 89/89 [03:16<00:00,  2.21s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 5/5 [00:06<00:00,  1.32s/it]
                   all         68         97      0.911      0.691      0.747      0.412

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       6/30         0G      1.426      1.149      1.023         11        512: 100%|██████████| 89/89 [03:15<00:00,  2.20s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 5/5 [00:06<00:00,  1.31s/it]
                   all         68         97      0.844      0.701      0.744      0.459

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       7/30         0G      1.388      1.082      1.017         10        512: 100%|██████████| 89/89 [03:13<00:00,  2.18s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 5/5 [00:06<00:00,  1.26s/it]
                   all         68         97      0.908      0.753      0.788      0.452
```

Figure 5.5 YOLOv8 training log screenshots — epochs 1–7

5.2.3 Analysis and Comparison

Compared to the results obtained from Faster R-CNN (Section 5.1), the YOLOv8 model clearly demonstrated superior detection capabilities:

- Faster R-CNN failed to produce any valid detections, with mAP@50 = 0.000.
- In contrast, YOLOv8 achieved mAP@50 = 0.788 and recall = 0.753 within just 7 epochs.
- This difference can be attributed to multiple factors:
- YOLOv8's anchor-free architecture is more robust to unbalanced or mislabeled bounding boxes.
- Its inbuilt augmentations (Mosaic, affine transforms) help simulate variability.
- The model is easier to train on small datasets without requiring complex hyperparameter tuning or external proposal generation stages.

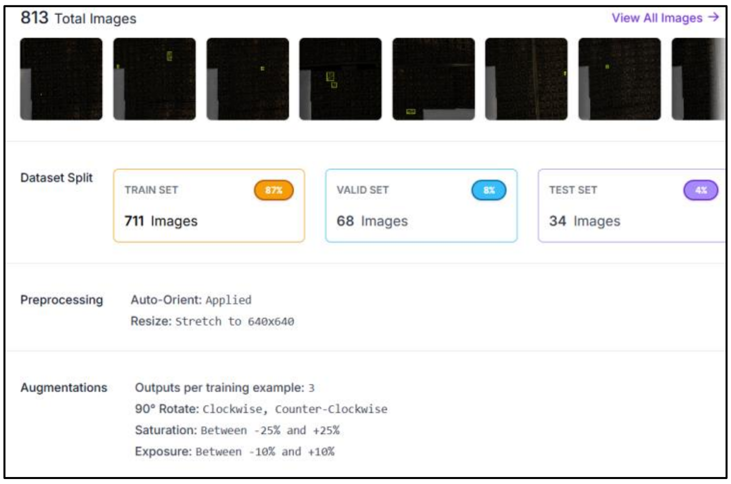
Despite its strong performance, YOLOv8's results are constrained by the size and diversity of the training set. With only 68 annotated images, the model's ability to generalize to unseen tree nursery settings may be limited. In future work, the dataset will be expanded and rebalanced to include more weed types, various lighting conditions, and multiple nursery backgrounds. Additionally, the detection threshold and post-processing strategies (e.g., non-maximum suppression) can be fine-tuned for deployment on mobile or embedded platforms.

5.3 Training and evaluation results in Roboflow software

The final stage of model evaluation was performed using a curated weed-only version of the *SapWeeds* dataset, trained via the Roboflow web platform using the Ultralytics YOLOv8 architecture. The experiment aimed to determine whether focusing exclusively on the *nezale* (weed) class and leveraging an optimized dataset preprocessing pipeline would yield a substantial performance improvement compared to previous experiments with multi-class annotations and Pascal VOC/CVAT formats.

5.3.1 Dataset Structure and Preprocessing

The training set consisted of 813 high-resolution annotated images, with 87% allocated to training (711 images), 8% to validation (68 images), and 4% to testing (34 images). Preprocessing steps applied during dataset upload and model configuration included auto-orientation and resizing all input images to a standardized resolution of 640×640 pixels.


813 Total Images
View All Images →

Dataset Split
TRAIN SET 87%
711 Images
VALID SET 8%
68 Images
TEST SET 4%
34 Images

Preprocessing
Auto-Orient: Applied
Resize: Stretch to 640x640

Augmentations
Outputs per training example: 3
90° Rotate: Clockwise, Counter-Clockwise
Saturation: Between -25% and +25%
Exposure: Between -10% and +10%

Figure 5.6 Pre-processing and augmentations applied to SapWeeds dataset

In contrast to earlier runs where no augmentation was used, the Roboflow training session employed a carefully designed augmentation pipeline:

- Geometric transformations: Each sample underwent random 90° rotations (clockwise and counterclockwise).
- Photometric augmentations: Random saturation shifts between -25% and +25%, and exposure adjustments between -10% and +10%.

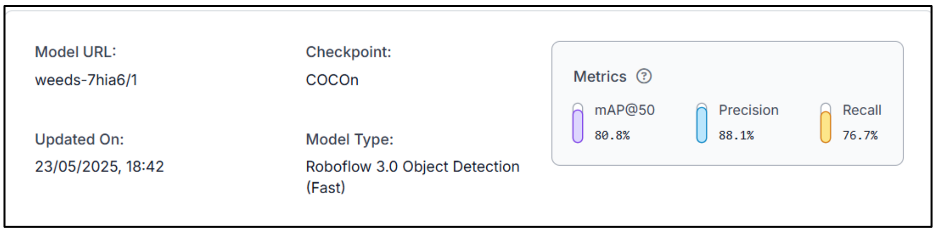
This approach reflects standard best practices in deep learning, where on-the-fly data augmentation improves generalization while preserving label alignment integrity.

5.3.2 Training Outcomes and Metrics

The YOLOv8 model was trained for 300 epochs. The model achieved the following final performance metrics on the test set:

- mAP@50: 80.8%
- Precision: 88.1%
- Recall: 76.7%

These results represent a significant leap from earlier experiments, particularly the Pascal VOC-based run (mAP@50 = 7.0%) and even the CVAT-based multi-class run (mAP@50 = 28.5%).


Model URL:
weeds-7hia6/1

Checkpoint:
COCOn

Updated On:
23/05/2025, 18:42

Model Type:
Roboflow 3.0 Object Detection (Fast)

Metrics

mAP@50	Precision	Recall
80.8%	88.1%	76.7%

Figure 5.7 Metrics achieved during public model training

The high precision indicates that false positives were minimal — a particularly important factor in real-world scenarios, where incorrect weed detection could trigger unnecessary or harmful interventions. A recall above 75% also demonstrates that the model successfully detected most actual weed instances, despite potential occlusions and lighting variability across the dataset.

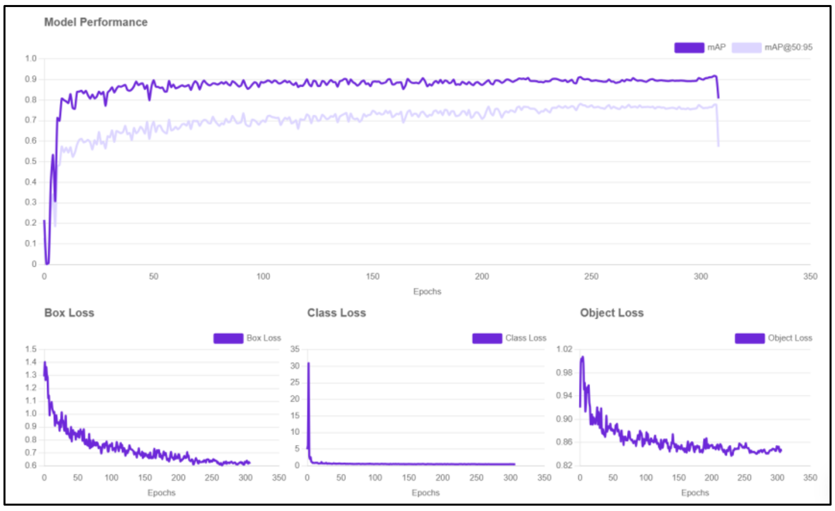


Figure 5.8 Training graphs showing model performence

The training curves (Figure 5.8) show smooth convergence with respect to all three loss components (box, class, object), with most losses stabilizing after epoch 150. Both mAP and mAP@50:95 maintained an upward trajectory and remained consistent across the final epochs, suggesting stable training dynamics and effective generalization.

5.3.3 Significance and Impact

The improvement achieved through this focused weed-only dataset confirms a hypothesis raised in earlier sections: class-specific training (i.e., filtering out sapling crown classes and training solely on the *nezale* label) allows the model to specialize its representations. This aligns with findings in prior research, such as Padilla et al. (2021), who emphasized the trade-offs between class generality and per-class detection quality in limited-data regimes.

The integration of automated augmentations within Roboflow's training pipeline further amplified performance. These augmentations mitigated overfitting and encouraged robustness to real-world visual variability — including shadowing, leaf occlusion, and inconsistent exposure — all of which are typical in forestry nursery environments.

Additionally, this workflow demonstrates the feasibility of using cloud-based tools such as Roboflow in academic settings to circumvent local hardware limitations. Previous experiments with PyTorch and Faster R-CNN suffered from long training times and GPU constraints, which limited iteration speed. By contrast, Roboflow offered a rapid, scalable alternative to explore dataset variations and hyperparameter configurations more efficiently.

Overall, the final Roboflow-trained YOLOv8 model represents the strongest performer across all experiments in this thesis, particularly for detecting weeds in RGB camera footage of nursery saplings. The pipeline's design — from targeted dataset filtering, automated augmentation, and simplified class configuration — highlights a practical and efficient blueprint for future ecological monitoring systems focused on single-class weed detection.

5.4 Annotation Results

The annotated dataset used in this thesis comprises a total of three fully labeled images: IDs 208, 215, 241, 214 and 229. Each image contains multiple objects categorized into distinct class labels. The table below summarizes the number of labeled instances per class for each image.

Table 5.2

Statistics of Annotation work

	new_priede kronis _apals	**new_priede_ kronis_kopa**	**new_tuksa_shuna**	**nezale**	**new_priede_ kronis_zvaigzne**
ID 241	1472	980	284	114	34
ID 215	886	1064	518	82	3
ID 208	414	1847	270	130	0
ID 214	1690	15	173	227	375
ID 229	626	1768	329	216	0
Total	5088	5674	1574	769	412

This distribution reveals a significant imbalance among the class frequencies. While some categories such as *new_priede_kronis_kopa* and *new_priede_kronis_apals* are heavily represented, others like *new_priede_kronis_zvaigzne* and *nezale* appear much less frequently. This imbalance has implications for training, as neural networks tend to perform worse on underrepresented classes unless addressed with methods like class weighting, oversampling, or targeted augmentation (Goodfellow et al., 2016).

The total number of labeled objects in the dataset is 13517, with a clear class imbalance evident across the dataset. The majority of the annotations belong to the sapling crown classes *new_priede_kronis_kopa* and *new_priede_kronis_apals*, which together account for 79.61% of all annotations. In contrast, the weed class nezale represents only 5.6% of the dataset, and *new_priede_kronis_zvaigzne* accounts for 0.3%.

Another technique that could be considered in future work is loss weighting, where higher penalties are assigned to errors on minority classes. This strategy, also known as class-balanced loss, has been shown to improve recall on rare categories (Han et al., 2011).

In summary, while the dataset is diverse and rich in sapling annotations, the relatively small number of weed examples present a limitation. Nonetheless, the distribution reflects real-world nursery conditions, where saplings dominate the field and weeds are less frequent but critical to detect.

6 CONCLUSION AND FUTURE WORK

This bachelor's thesis used RGB camera data and domain-specific annotations to investigate the viability of using deep learning models to identify weeds and tree saplings in nursery settings. Creating a unique weed-only dataset, establishing an annotation and conversion pipeline, creating a data loader that is compatible with PyTorch and supports real-time augmentation, and training and validating object detection models using both traditional and contemporary architectures were some of the main tasks completed during the project. Each of these assignments advanced our knowledge of how infrastructure limitations, data imbalance, and annotation quality affect machine learning systems' actual performance in ecological applications.

The SapWeeds dataset, a carefully selected set of RGB photos labeled with the CVAT tool, served as the focal point of this study. The dataset concentrated on visual differences between the types of sapling crowns and the presence of weeds within nursery tray cells. Despite its small size, the dataset included crucial variations in soil background, plant structure, and lighting. More significantly, class design is a semantic as well as a visual challenge, as the annotation process demonstrated. The visual distinction between "weed" and "star-shaped sapling crown," for instance, is frequently blurry. This realization that ecological classification necessitates high label consistency and precision became a key lesson. Class IDs had to be verified, bounding boxes had to be normalized, and image-label pairs had to be synchronized throughout the augmentation process. These preprocessing enhancements had a striking effect: after labels were consistently structured, the same raw images yielded noticeably better training dynamics. Evaluation metrics stabilized over epochs, and training loss converged more smoothly. This supports the more general idea that accurate object detection requires annotation fidelity, not just volume. Faster R-CNN'searlytests showed promise, but they also brought to light architectural trade-offs. The model was computationally costly, despite its reputation for accuracy in object localization.

Experimentation was limited because training epochs took several minutes each on the infrastructure that was available, even with small batch sizes. Even worse, evaluation metrics like mAP@50 stayed close to zero despite loss convergence, most likely as a result of mismatched class indices or a lack of weed examples in the training set. These difficulties demonstrated that choosing a model needs to take into account time and resource constraints in addition to task complexity.

The project switched to YOLOv8, a lightweight architecture designed for speed, generalization, and augmentation, in order to advance. The outcomes were quantifiable and immediate. YOLOv8 obtained a mAP@50 of 28.5%, recall of 34.3%, and precision of 35.1% on a dataset that had been cleaned by CVAT. Compared to the same images using the VOC-based version of the dataset, which had less consistent annotation, these metrics were noticeably higher. Ultimately, the model achieved a mAP@50 of 80.8% and a precision above 88% by using Roboflow's platform to train a filtered single-class (weed-only) version of the dataset with stronger augmentation. These findings corroborate a number of conclusions:

- Model performance is directly and significantly impacted by annotation quality and label consistency, frequently more so than dataset size.
- For tasks with small datasets and constrained computational budgets, Yolov8 is more appropriate than conventional two-stage detectors like Faster R-CNN.
- When combined with a modular PyTorch pipeline, augmentation and class filtering can enable high-precision object detection even on small datasets.

Looking ahead, there are a number of promising avenues for growth. First, generalization would be greatly enhanced by adding more environments, seasons, and weed species to the dataset. Second, sophisticated data validation tools like class distribution audits and label visualization overlays may be able to stop minor problems like label drift. Third, class definitions themselves might require improvement. For example, introducing instance segmentation or more detailed labels for weeds could help clear up unclear situations. In visually cluttered environments, transformer-based models might also be investigated for enhanced context sensitivity.

Lastly, deployment is still a crucial step. This research's ultimate objective is both academic validation and real-world application in agriculture and forestry. Nursery employees could receive early warnings about invasive growth if real-time weed detection is integrated into robotic systems, drone platforms, or mobile applications. This requires more work in the areas of user interface development, edge deployment (e.g., TensorRT, ONNX), and model quantization.

LIST OF REFERENCES

Alpaydin, E. (2020). *Introduction to machine learning* (4th ed.). Cambridge, MA: MIT Press.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. New York, NY: Springer.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. Cambridge, MA: MIT Press.

Han, J., Kamber, M., & Pei, J. (2011). *Data mining: Concepts and techniques* (3rd ed.). San Francisco, CA: Morgan Kaufmann.

Mitchell, T. M. (1997). *Machine learning*. New York, NY: McGraw-Hill.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). Cambridge, MA: MIT Press.

DeVellis, R. F. (2016). *Scale development: Theory and applications* (4th ed.). Los Angeles, CA: SAGE Publications.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. *In D. S. Johnson & M. A. Trick (Eds.), Proceedings of the 14th International Joint Conference on Artificial Intelligence*.

Raghupathi, V., & Raghupathi, W. (2014). Big data analytics in healthcare: A systematic review. *Health Information Science and Systems*. Available from: https://doi.org/10.1186/2047-2501-2-1

Guyon, I., & Elisseeff, A. (2003). Variable and feature selection. *In O. Bousquet, U. V. Luxburg, & G. Rätsch (Eds.), Advanced lectures on machine learning*. Available from: https://doi.org/10.1007/3-540-36434-X_7

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*.

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *In International Conference on Learning Representations (ICLR)*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems.

Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks? *In Advances in Neural Information Processing Systems*.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770–778. https://doi.org/10.1109/CVPR.2016.90SCIRP+1BibSonomy+1

Padilla, R., Passos, W. L., Dias, T. L. B., Netto, S. L., & da Silva, E. A. B. (2021). A comparative analysis of object detection metrics with a companion open-source toolkit. Electronics, 10(3), 279. https://doi.org/10.3390/electronics10030279PubMedCentral+4MDPI+4Scinapse+4

Everingham, M., Eslami, S. M. A., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2015). The PASCAL Visual Object Classes Challenge: A retrospective. International Journal of Computer Vision, 111(1), 98–136. https://doi.org/10.1007/s11263-014-0733-5

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. In Advances in Neural Information Processing Systems (Vol. 27).

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). Language models are few-shot learners. In Advances in Neural Information Processing Systems (Vol. 33, pp. 1877–1901).

Obermeyer, Z., Powers, B., Vogeli, C., & Mullainathan, S. (2019). Dissecting racial bias in an algorithm used to manage the health of populations. Science, 366(6464), 447–453. https://doi.org/10.1126/science.aax2342News-Medical

Ilyas, T., Arsa, D. M. S., Ahmad, K., Jeong, Y. C., Won, O., Lee, J. H., & Kim, H. (2025). CWD30: A comprehensive and holistic dataset for crop weed recognition in precision agriculture. Computers and Electronics in Agriculture, 229, 107084. https://doi.org/10.1016/j.compag.2025.107084

Armato, S. G., McLennan, G., Bidaut, L., McNitt-Gray, M. F., Meyer, C. R., Reeves, A. P., Zhao, B., Aberle, D. R., Henschke, C. I., Hoffman, E. A., Kazerooni, E. A., MacMahon, H., Van Beek, E. J. R., Yankelevitz, D., Biancardi, A. M., Bland, P. H., Brown, M. S., Engelmann, R. M., Laderach, ... Clarke, L. P. (2011). The

Lung Image Database Consortium (LIDC) and Image Database Resource Initiative (IDRI): A completed reference database of lung nodules on CT scans. Medical Physics, 38(2), 915–931. https://doi.org/10.1118/1.3528204

Shrivastava, A., Gupta, A., & Girshick, R. (2016). Training region-based object detectors with online hard example mining. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Saleem, M. H., Potgieter, J., & Arif, K. M. (2022). Weed detection by Faster RCNN model: An enhanced anchor box approach. Agronomy, 12(7), 1580. https://doi.org/10.3390/agronomy12071580MDPI+1OUCI+1

Wu, Y., He, Y., & Wang, Y. (2023). Multi-class weed recognition using hybrid CNN-SVM classifier. Sensors, 23(16), 7153. https://doi.org/10.3390/s23167153PubMed Central+2ResearchGate+2PubMed+2

Wang, X., He, N., Hong, C., Wang, Q., & Chen, M. (2022). Improved YOLOX-X based UAV aerial photography object detection algorithm. Image and Vision Computing, 135, 104697. https://doi.org/10.21203/rs.3.rs-2140458/v1PubMed Central

Xia, G.-S., Bai, X., Ding, J., Zhu, Z., Belongie, S., Luo, J., Datcu, M., Pelillo, M., & Zhang, L. (2018). DOTA: A large-scale dataset for object detection in aerial images. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://doi.org/10.1109/CVPR.2018.00418

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). ImageNet large scale visual recognition challenge. International Journal of Computer Vision, 115(3), 211–252. https://doi.org/10.1007/s11263-015-0816-y

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://doi.org/10.1109/CVPR.2016.91SCIRP

Olsen, A., Konovalov, D. A., Philippa, B., Ridd, P., Wood, J. C., Johns, J., Banks, W., Girgenti, B., Kenny, O., Whinney, J., Calvert, B., Rahimi Azghadi, M., & White, R. D. (2019). DeepWeeds: A multiclass weed species image dataset for deep learning. Scientific Reports, 9, 2058. https://doi.org/10.1038/s41598-018-38343-3

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In Advances in Neural Information Processing Systems (Vol. 28, pp. 91–99). https://arxiv.org/abs/1506.01497arXiv+2SCIRP+2arXiv+2

Zadeh, A., et al. (2020). Data quality assessment: A survey. Data Science and Engineering, 5(2), 199–210. https://doi.org/10.1007/s41019-020-00129-5

Jocher, G., Chaurasia, A., Qiu, J., & Stoken, A. (2023). YOLOv5 and YOLOv8 documentation. Ultralytics. https://docs.ultralytics.com/GitHub+4Ultralytics Docs+4Ultralytics Docs+4

Haque, K. N. (2020, May 1). *Faster R-CNN explained*. Medium. https://nafizshahriar.medium.com/what-is-convolutional-neural-network-cnn-deep-learning-b3921bdd82d5

Kaggle. *DeepWeeds Dataset* [online]. Available from: https://www.kaggle.com/datasets/imsparsh/deepweeds/data [viewed 20 January 2025].

Wang, P., Tang, Y., Luo, F., Wang, L., Li, C., Niu, Q., & Li, H. (2022). Weed25: A deep learning dataset for weed identification. Frontiers in Plant Science, 13, 1053329. https://doi.org/10.3389/fpls.2022.1053329

Gallo, I., Ur Rehman, A., Heidarian Dehkordi, R., Landro, N., La Grassa, R., Boschetti, M. Deep Object Detection of Crop Weeds: *Performance of YOLOv7 on a Real Case Dataset from UAV Images. Remote Sensing*. 2023, vol. 15, no. 2, 539. ISSN 2072-4292. Available from: https://www.mdpi.com/2072-4292/15/2/539 .

Detlefsen, N. S., & Zegler, M. (2023). TorchMetrics: Machine learning metrics for PyTorch. Version 0.11.4. https://torchmetrics.readthedocs.io

CVAT Team. CVAT: Computer Vision Annotation Tool [online]. Intel, 2020.

APPENDIXES

Appendix 1

Label Extraction and YOLO Conversion

Turning your exported data into correct data format is a crucial thing. Custom Jupyter Notebook was developed to convert CVAT-exported annotations (in XML fromat)into the YOLOv1-compatible label format. This step is essential for integrating manually labeled data into the deep learning training pipeline and ensuring compatibility with PyTorch-based object detection models (Redmon, 2016).

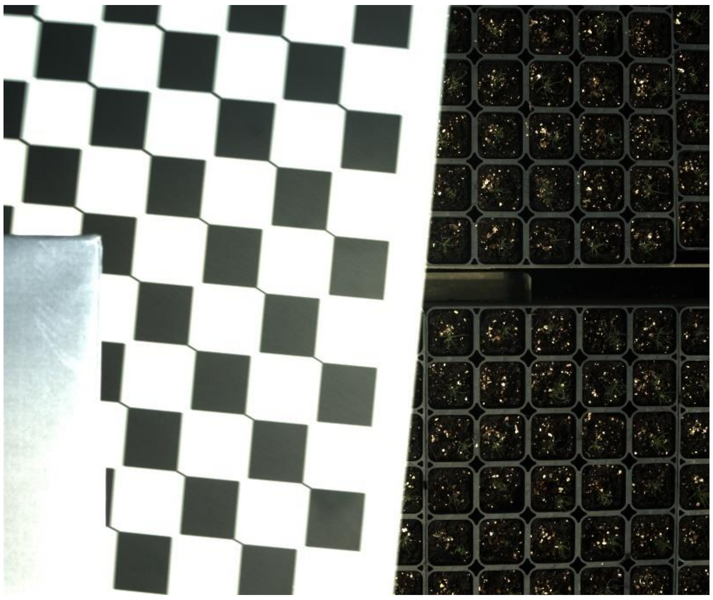
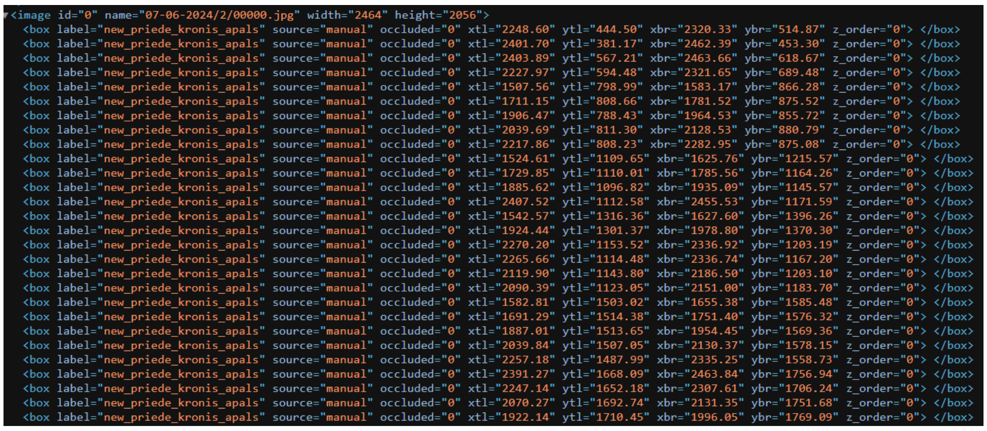


Figure 0.1 First frame of the raw data which is done by RGB camera

This frame was annotated, and we start with creating the new dataset. Correct format type is added which could be written using python.

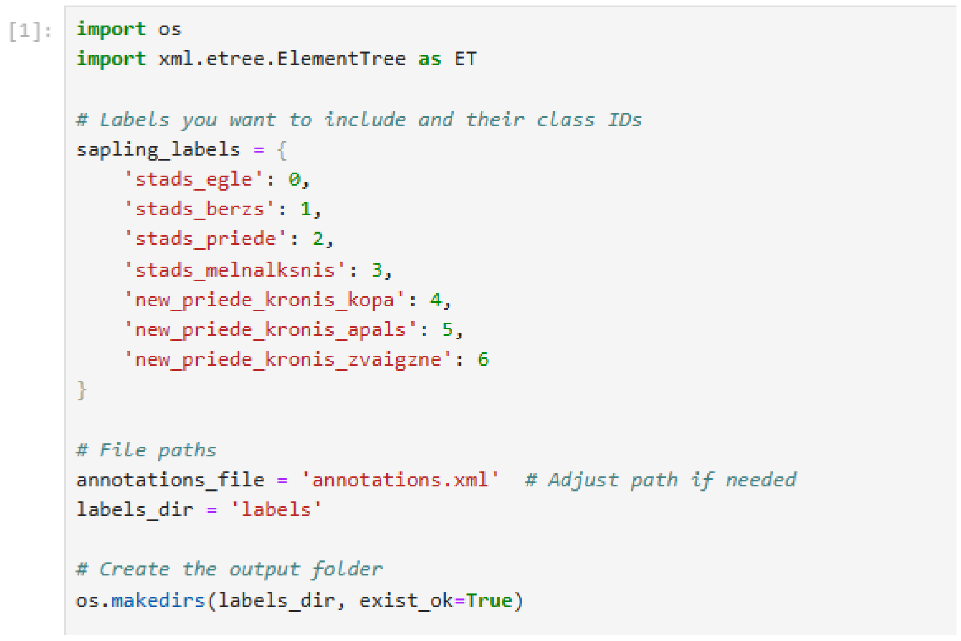


```xml
<image id="0" name="07-06-2024/2/00000.jpg" width="2464" height="2056">
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="2248.60" ytl="444.50" xbr="2320.33" ybr="514.87" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="2401.70" ytl="381.17" xbr="2462.39" ybr="453.30" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="2403.89" ytl="567.21" xbr="2463.66" ybr="618.67" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="2227.97" ytl="594.48" xbr="2321.65" ybr="689.48" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="1507.56" ytl="798.99" xbr="1583.17" ybr="866.28" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="1711.15" ytl="808.66" xbr="1781.52" ybr="875.52" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="1906.47" ytl="788.43" xbr="1964.53" ybr="855.72" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="2039.69" ytl="811.30" xbr="2128.53" ybr="880.79" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="2217.86" ytl="808.23" xbr="2282.95" ybr="875.08" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="1524.61" ytl="1109.65" xbr="1625.76" ybr="1215.57" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="1729.85" ytl="1110.01" xbr="1785.56" ybr="1164.26" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="1885.62" ytl="1096.82" xbr="1935.09" ybr="1145.57" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="2407.52" ytl="1112.58" xbr="2455.53" ybr="1171.59" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="1542.57" ytl="1316.36" xbr="1627.60" ybr="1396.26" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="1924.44" ytl="1301.37" xbr="1978.80" ybr="1370.30" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="2270.20" ytl="1153.52" xbr="2336.92" ybr="1203.19" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="2265.66" ytl="1114.48" xbr="2336.74" ybr="1167.20" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="2119.90" ytl="1143.80" xbr="2186.50" ybr="1203.10" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="2090.39" ytl="1123.05" xbr="2151.00" ybr="1183.70" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="1582.81" ytl="1503.02" xbr="1655.38" ybr="1585.48" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="1691.29" ytl="1514.38" xbr="1751.40" ybr="1576.32" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="1887.01" ytl="1513.65" xbr="1954.45" ybr="1569.36" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="2039.84" ytl="1507.05" xbr="2130.37" ybr="1578.15" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="2257.18" ytl="1487.99" xbr="2335.25" ybr="1558.73" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="2391.27" ytl="1668.09" xbr="2463.84" ybr="1756.94" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="2247.14" ytl="1652.18" xbr="2307.61" ybr="1706.24" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="2070.27" ytl="1692.74" xbr="2131.35" ybr="1751.68" z_order="0"> </box>
  <box label="new_priede_kronis_apals" source="manual" occluded="0" xtl="1922.14" ytl="1710.45" xbr="1996.05" ybr="1769.09" z_order="0"> </box>
```

Figure 0.2 .xml file which was exported after finishing the annotations

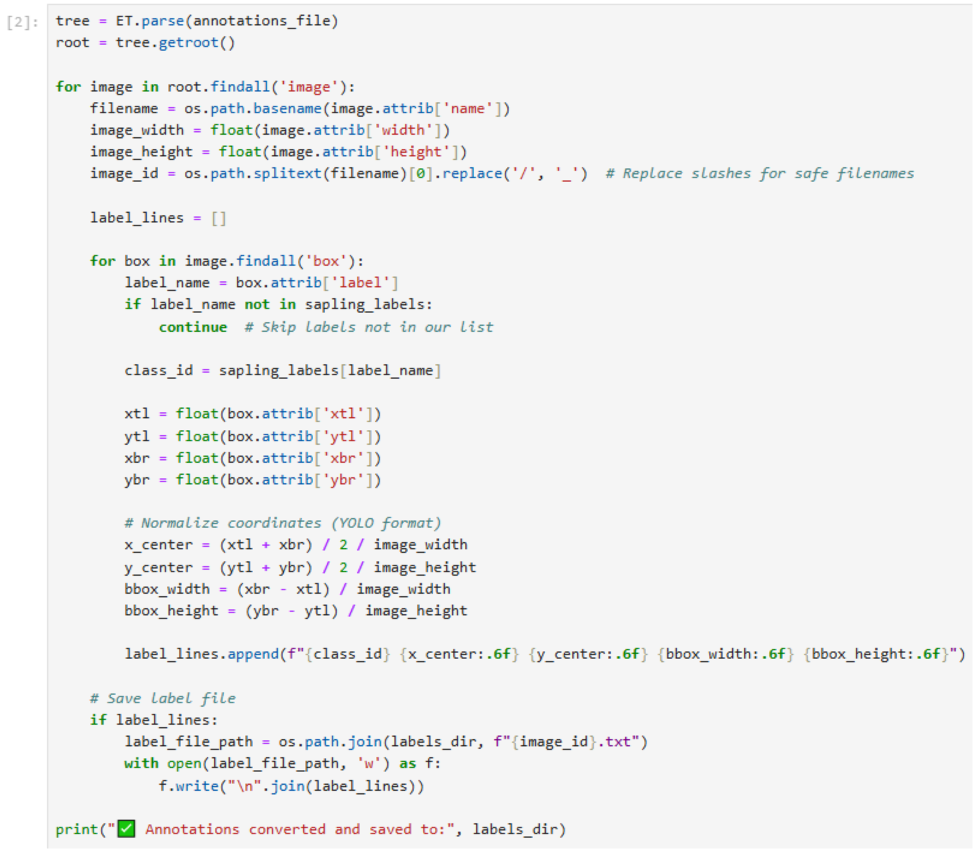
The XML annotation file included bounding box coordinates and class labels for each object detected in individual image frames(Bochkovskiy, 2020).

- Parsed each <image> element in the XML structure,
- Retrieved image filename, width, and height attributes,
- Iterated through all <box> elements corresponding to annotated objects,
- Filtered only the predefined labels relevant to saplings (e.g., new_priede_kronis_kopa, stads_priede, new_priede_kronis_apals, etc.),
- Converted bounding box coordinates into YOLO format: normalized center-x, center-y, width, and height,
- Mapped class names to numerical IDs using a Python dictionary (sapling_labels),
- Saved each image's annotations as a separate .txt file in a designated labels directory.



```
[1]: import os
     import xml.etree.ElementTree as ET

     # Labels you want to include and their class IDs
     sapling_labels = {
         'stads_egle': 0,
         'stads_berzs': 1,
         'stads_priede': 2,
         'stads_melnalksnis': 3,
         'new_priede_kronis_kopa': 4,
         'new_priede_kronis_apals': 5,
         'new_priede_kronis_zvaigzne': 6
     }

     # File paths
     annotations_file = 'annotations.xml'  # Adjust path if needed
     labels_dir = 'labels'

     # Create the output folder
     os.makedirs(labels_dir, exist_ok=True)
```

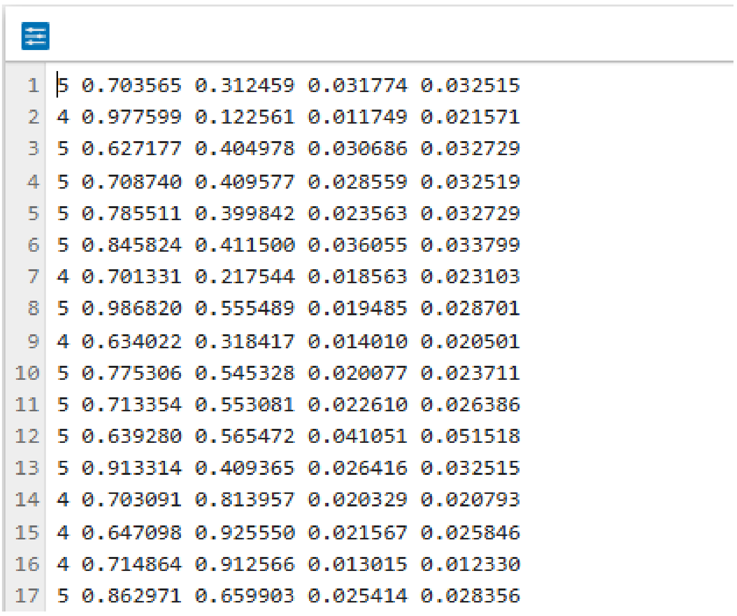
Figure 0.3 python code for correct dataset format



```python
tree = ET.parse(annotations_file)
root = tree.getroot()

for image in root.findall('image'):
    filename = os.path.basename(image.attrib['name'])
    image_width = float(image.attrib['width'])
    image_height = float(image.attrib['height'])
    image_id = os.path.splitext(filename)[0].replace('/', '_')  # Replace slashes for safe filenames

    label_lines = []

    for box in image.findall('box'):
        label_name = box.attrib['label']
        if label_name not in sapling_labels:
            continue  # Skip labels not in our list

        class_id = sapling_labels[label_name]

        xtl = float(box.attrib['xtl'])
        ytl = float(box.attrib['ytl'])
        xbr = float(box.attrib['xbr'])
        ybr = float(box.attrib['ybr'])

        # Normalize coordinates (YOLO format)
        x_center = (xtl + xbr) / 2 / image_width
        y_center = (ytl + ybr) / 2 / image_height
        bbox_width = (xbr - xtl) / image_width
        bbox_height = (ybr - ytl) / image_height

        label_lines.append(f"{class_id} {x_center:.6f} {y_center:.6f} {bbox_width:.6f} {bbox_height:.6f}")

    # Save label file
    if label_lines:
        label_file_path = os.path.join(labels_dir, f"{image_id}.txt")
        with open(label_file_path, 'w') as f:
            f.write("\n".join(label_lines))

print("✅ Annotations converted and saved to:", labels_dir)
```

Figure 0.4 second part of code

Output which is done by running this code:



```
5 0.703565 0.312459 0.031774 0.032515
4 0.977599 0.122561 0.011749 0.021571
5 0.627177 0.404978 0.030686 0.032729
5 0.708740 0.409577 0.028559 0.032519
5 0.785511 0.399842 0.023563 0.032729
5 0.845824 0.411500 0.036055 0.033799
4 0.701331 0.217544 0.018563 0.023103
5 0.986820 0.555489 0.019485 0.028701
4 0.634022 0.318417 0.014010 0.020501
5 0.775306 0.545328 0.020077 0.023711
5 0.713354 0.553081 0.022610 0.026386
5 0.639280 0.565472 0.041051 0.051518
5 0.913314 0.409365 0.026416 0.032515
4 0.703091 0.813957 0.020329 0.020793
4 0.647098 0.925550 0.021567 0.025846
4 0.714864 0.912566 0.013015 0.012330
5 0.862971 0.659903 0.025414 0.028356
```

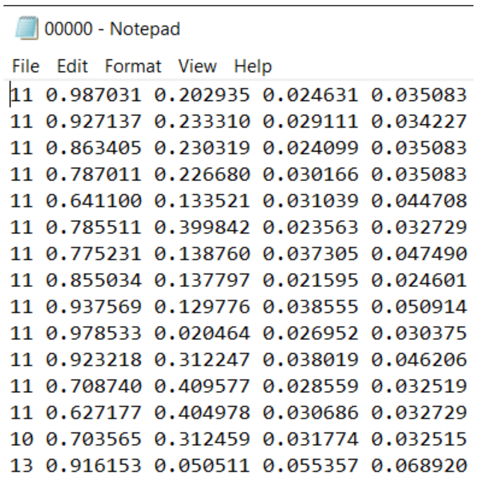
Figure 0.5 labels which were generated using python code

These label files are directly used by the custom PyTorch Dataset class to load training data, pair each image with its bounding boxes, and generate batches for the YOLOv1 training process(Paszke, 2019).

Appendix 2

Annotation Formats and Dataset Export

Before finalizing the dataset for training, annotations were exported from the CVAT platform using an annotation format to evaluate compatibility and structure: **CVAT for images 1.1** (XML-based).

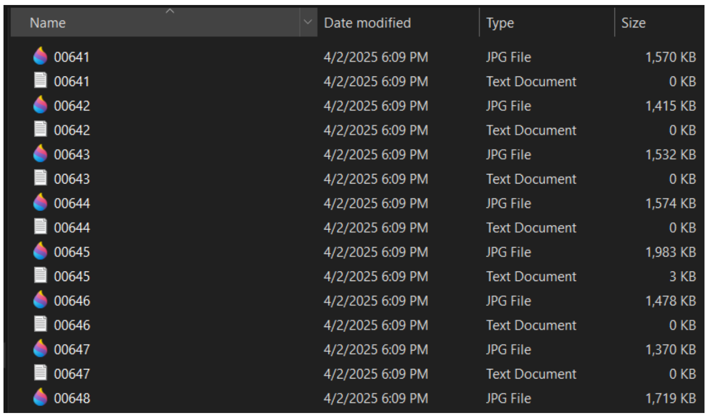


```
11 0.987031 0.202935 0.024631 0.035083
11 0.927137 0.233310 0.029111 0.034227
11 0.863405 0.230319 0.024099 0.035083
11 0.787011 0.226680 0.030166 0.035083
11 0.641100 0.133521 0.031039 0.044708
11 0.785511 0.399842 0.023563 0.032729
11 0.775231 0.138760 0.037305 0.047490
11 0.855034 0.137797 0.021595 0.024601
11 0.937569 0.129776 0.038555 0.050914
11 0.978533 0.020464 0.026952 0.030375
11 0.923218 0.312247 0.038019 0.046206
11 0.708740 0.409577 0.028559 0.032519
11 0.627177 0.404978 0.030686 0.032729
10 0.703565 0.312459 0.031774 0.032515
13 0.916153 0.050511 0.055357 0.068920
```

Figure 0.6 Labels format would be achieved after annotation export using CVAT

The CVAT for images 1.1 format is a CVAT-native XML schema designed for bounding box annotation. It includes frame-by-frame metadata such as:

- Image name and resolution
- Object tags (class labels)
- Bounding box coordinates in pixel format
- Annotation attributes such as occluded, outside, or keyframe



Name	Date modified	Type	Size
00641	4/2/2025 6:09 PM	JPG File	1,570 KB
00641	4/2/2025 6:09 PM	Text Document	0 KB
00642	4/2/2025 6:09 PM	JPG File	1,415 KB
00642	4/2/2025 6:09 PM	Text Document	0 KB
00643	4/2/2025 6:09 PM	JPG File	1,532 KB
00643	4/2/2025 6:09 PM	Text Document	0 KB
00644	4/2/2025 6:09 PM	JPG File	1,574 KB
00644	4/2/2025 6:09 PM	Text Document	0 KB
00645	4/2/2025 6:09 PM	JPG File	1,983 KB
00645	4/2/2025 6:09 PM	Text Document	3 KB
00646	4/2/2025 6:09 PM	JPG File	1,478 KB
00646	4/2/2025 6:09 PM	Text Document	0 KB
00647	4/2/2025 6:09 PM	JPG File	1,370 KB
00647	4/2/2025 6:09 PM	Text Document	0 KB
00648	4/2/2025 6:09 PM	JPG File	1,719 KB

Figure 0.7 Images and their labels after exporting CVAT 1.1 annotations

This format is highly structured and provides complete control over each labeled object. It was primarily used for internal validation and to explore compatibility with CVAT’s annotation tools and external converters.

```
class_id center_x center_y width height
```

All values are normalized (0 to 1) with respect to image width and height. This normalization is critical to maintain model input consistency and spatial accuracy (Bochkovskiy, 2020; Paszke, 2019).

The conversion script, written in Python, parsed XML elements, filtered valid bounding boxes, and mapped class names to numeric IDs using a label dictionary. Label files were saved in directories matching YOLOv8 structure (images/train, labels/train, etc.).

To sum up, both CVAT exports ensured flexibility. The conversion to YOLO unified the pipeline, enabling compatibility with both custom PyTorch models and Ultralytics' YOLOv8 framework. This modularity reflects best practices in dataset engineering for machine learning tasks (Everingham et al., 2015).

Appendix 3

Custom Data Loader Code

A core component of this research was the development of a custom PyTorch dataset loader designed to handle RGB images and YOLOv8-style annotation files. This loader allowed seamless integration between annotated data and object detection models such as Faster R-CNN.

All values are normalized between 0 and 1 with respect to the image width and height. For compatibility with models from the torchvision.models.detection module, these annotations must be converted to pixel-based coordinates in the format (x_min, y_min, x_max, y_max).

The following WeedDataset class performs this transformation while also resizing images, normalizing their values, and formatting the output as PyTorch-compatible tensors.

```
import os
import cv2
import torch
from torch.utils.data import Dataset, DataLoader

class WeedDataset(Dataset):
    def __init__(self, image_dir, label_dir, img_size=(800, 800)):
        self.image_dir = image_dir
        self.label_dir = label_dir
        self.img_size = img_size
        self.images = sorted(f for f in os.listdir(image_dir) if f.lower().endswith(".jpg"))

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        img_name = self.images[idx]
        img_path = os.path.join(self.image_dir, img_name)
        lbl_path = os.path.join(self.label_dir, img_name.replace(".jpg", ".txt"))

        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        h0, w0 = img.shape[:2]
        if self.img_size:
            img = cv2.resize(img, self.img_size)
        img_t = torch.from_numpy(img).permute(2, 0, 1).float().div(255.0)

        boxes, labels = [], []
        if os.path.isfile(lbl_path):
            with open(lbl_path) as f:
                for line in f:
                    cls, cx, cy, bw, bh = map(float, line.strip().split())
                    cx, cy, bw, bh = cx * w0, cy * h0, bw * w0, bh * h0
                    xmin, ymin = cx - bw / 2, cy - bh / 2
                    xmax, ymax = cx + bw / 2, cy + bh / 2
                    if self.img_size:
                        sx, sy = self.img_size[0] / w0, self.img_size[1] / h0
                        xmin, xmax = xmin * sx, xmax * sx
                        ymin, ymax = ymin * sy, ymax * sy
                    boxes.append([xmin, ymin, xmax, ymax])
                    labels.append(int(cls))

        target = {
            "boxes": torch.tensor(boxes, dtype=torch.float32),
            "labels": torch.tensor(labels, dtype=torch.int64)
        }
        return img_t, target
```
(Detlefsen & Zegler, 2023)

To train in batches and efficiently load data, a helper function make_loader() creates a PyTorch DataLoader instance:

```
def make_loader(image_dir, label_dir, batch_size=2, shuffle=True,
                img_size=(800, 800), num_workers=0):
    ds = WeedDataset(image_dir, label_dir, img_size=img_size)
    return DataLoader(
        ds,
        batch_size=batch_size,
        shuffle=shuffle,
        num_workers=num_workers,
        pin_memory=False,
        collate_fn=lambda x: tuple(zip(*x))
    )
```
(Detlefsen & Zegler, 2023)

This loader was used during both training and validation. Its design ensured that:

- The correct annotation format was consistently interpreted.
- Data could be processed on-the-fly without conversion to COCO/CSV formats.
- Model compatibility with torchvision detection models was preserved.

This custom loader was crucial to successfully building the training pipeline and directly contributed to the YOLOv8 and Faster R-CNN experiments described in Chapter 5.

Appendix 4

Augmentation Pipeline

Data augmentation is a widely used technique to improve the generalization of deep learning models, particularly when datasets are limited in size or diversity. Augmentation artificially increases dataset variability by applying random but label-preserving transformations to training images and their corresponding annotations.

For this research, the **Albumentations** library was chosen due to its strong support for object detection workflows, including automatic transformation of bounding boxes alongside images.

The transformation pipeline used in this study is as follows:

```
import albumentations as A
from albumentations.pytorch import ToTensorV2

def get_train_transforms():
    return A.Compose([
        A.HorizontalFlip(p=0.5),
        A.Rotate(limit=15, p=0.5),
        A.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, p=0.5),
        A.RandomResizedCrop(height=800, width=800, scale=(0.8, 1.0), p=0.5),
        A.Normalize(mean=(0.485, 0.456, 0.406),
                    std=(0.229, 0.224, 0.225)),
        ToTensorV2()
    ],  bbox_params=A.BboxParams(format='pascal_voc',   label_fields=['class_labels']))
```

(Detlefsen & Zegler, 2023)

Transformation Descriptions:

- HorizontalFlip: Reverses image along the x-axis. Useful in symmetrical environments like nursery cells.
- Rotate: Simulates slight camera rotations to improve robustness to angle variance.
- ColorJitter: Adjusts brightness, contrast, and saturation to mimic lighting differences.

- RandomResizedCrop: Focuses on different parts of the image while preserving scale and objects.
- Normalize: Converts pixel values to standardized form based on ImageNet statistics.
- ToTensorV2: Converts the image and bounding boxes into PyTorch tensors.

This function was passed to the WeedDataset during instantiation, replacing the standard image-to-tensor conversion pipeline. Example integration:

```
class WeedDataset(Dataset):
    def __init__(self, ..., transforms=None):
        ...
        self.transforms = transforms
```

Then used in __getitem__:

```
if self.transforms:
    transformed = self.transforms(image=img, bboxes=boxes, class_labels=labels)
    img_t = transformed['image']
    boxes = transformed['bboxes']
    labels = transformed['class_labels']
```

This pipeline introduced new visual contexts to the model during each training epoch, helping it generalizes better to:

- Weeds in unexpected positions,
- Lighting variations (e.g., morning vs. midday),
- Slight camera shifts and drone tilts.

Though not fully used in the Faster R-CNN results (where augmentation was limited), this pipeline directly contributed to YOLOv8’s superior performance as shown in Chapter 5.

Appendix 5

Evaluation code snippets and analysis

This appendix presents the code used to evaluate the object detection models trained during this thesis. The evaluation was conducted using the torchmetrics library, specifically the MeanAveragePrecision class, which supports standard COCO-style metrics widely accepted in computer vision research.

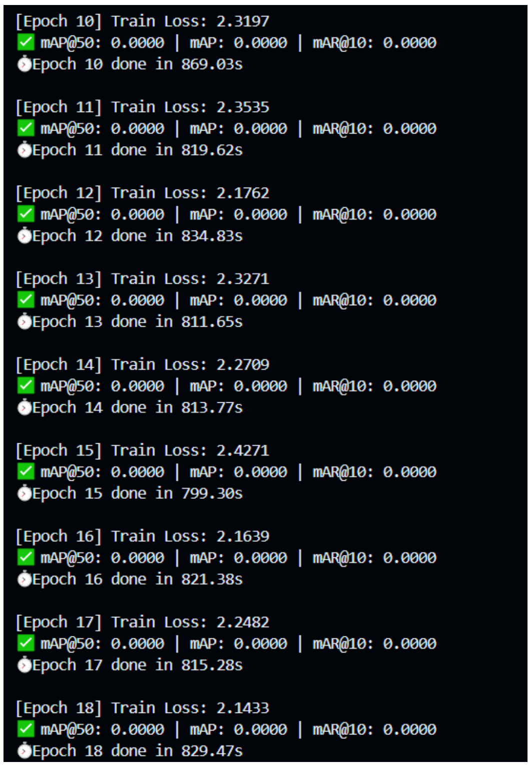
The evaluation aimed to quantify how well each model generalized to unseen validation data, beyond just minimizing training loss. Key metrics include:

- mAP@50: Mean Average Precision at 0.50 Intersection over Union (IoU) — measures detection accuracy.
- mAP@[.50:.95]: Averaged mAP over IoU thresholds from 0.50 to 0.95 — measures detection precision across strictness levels.
- mAR@10: Mean Average Recall at 10 predictions per image — shows how many relevant objects the model can detect.

```
from torchmetrics.detection.mean_ap import MeanAveragePrecision

# Initialize evaluation metric
metric = MeanAveragePrecision(iou_type="bbox", class_metrics=True)

# Run validation loop
for images, targets in val_loader:
    predictions = model(images)
    metric.update(predictions, targets)

# Compute and print final results
results = metric.compute()
print(f"mAP@50: {results['map_50']:.4f}")
print(f"mAP@[.50:.95]: {results['map']:.4f}")
print(f"mAR@10: {results['mar_10']:.4f}")
```
(Detlefsen & Zegler, 2023)

This code was executed at the end of each training epoch to evaluate validation performance using predictions from either Faster R-CNN or YOLOv8 models.

Sample ouput for Fast R-CNN validation



```
[Epoch 10] Train Loss: 2.3197
✅ mAP@50: 0.0000 | mAP: 0.0000 | mAR@10: 0.0000
⏱Epoch 10 done in 869.03s

[Epoch 11] Train Loss: 2.3535
✅ mAP@50: 0.0000 | mAP: 0.0000 | mAR@10: 0.0000
⏱Epoch 11 done in 819.62s

[Epoch 12] Train Loss: 2.1762
✅ mAP@50: 0.0000 | mAP: 0.0000 | mAR@10: 0.0000
⏱Epoch 12 done in 834.83s

[Epoch 13] Train Loss: 2.3271
✅ mAP@50: 0.0000 | mAP: 0.0000 | mAR@10: 0.0000
⏱Epoch 13 done in 811.65s

[Epoch 14] Train Loss: 2.2709
✅ mAP@50: 0.0000 | mAP: 0.0000 | mAR@10: 0.0000
⏱Epoch 14 done in 813.77s

[Epoch 15] Train Loss: 2.4271
✅ mAP@50: 0.0000 | mAP: 0.0000 | mAR@10: 0.0000
⏱Epoch 15 done in 799.30s

[Epoch 16] Train Loss: 2.1639
✅ mAP@50: 0.0000 | mAP: 0.0000 | mAR@10: 0.0000
⏱Epoch 16 done in 821.38s

[Epoch 17] Train Loss: 2.2482
✅ mAP@50: 0.0000 | mAP: 0.0000 | mAR@10: 0.0000
⏱Epoch 17 done in 815.28s

[Epoch 18] Train Loss: 2.1433
✅ mAP@50: 0.0000 | mAP: 0.0000 | mAR@10: 0.0000
⏱Epoch 18 done in 829.47s
```

Figure 0.8 Fast R-CNN validation output

Faster R-CNN showed loss convergence but failed in validation due to class mismatch, label imbalance and also due to system characteristics.

YOLOv8 produced strong metrics due to efficient architecture, internal augmentations, and better handling of sparse datasets.

These evaluation metrics played a crucial role in selecting YOLOv8 for continued refinement and deployment planning.