Master Thesis

at the University of Osnabrück

(for the degree of M.Sc. in Cognitive Science)

Topic

Disease Detection on Plant Leaves using Few-shot Learning

Submitted By

Dibyashree Nahak

Matriculation Number: 988371

Submitted on

24.03.2023

First Supervisor

Dr. Joachim Hertzberg

Second Supervisor

M.Sc. Lena Herrmann

Universität Osnabrück

Institute of Cognitive Science

Osnabrück

Universität Osnabrück

Acknowledgement

I want to express my gratitude to Professor Dr. Joachim Hertzberg for agreeing to supervise this project.

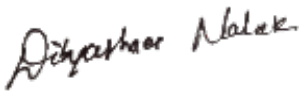
I also extend my heartfelt gratitude to my second supervisor and guide M.Sc. Lena Herrmann for introducing me to the research area of few-shot learning. Her guidance has been priceless in helping me gain a deeper understanding of the subject. I can not thank her enough for her valuable insights throughout my research related to the research problem and her input on writing and structuring the draft.

Many thanks to M.Sc. Naeem Iqbal for his valuable time and insights while helping me with the Object Detection research.

Lastly, I want to thank my friends who have supported me with proofreading parts of the thesis and giving constructive feedback.

Declaration of Authorship

I hereby certify that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other university.



Signature

Yokohama, Japan 24-03-2023

City, Date

Abstract

Detecting unhealthy leaves in crops is crucial for early disease and pest management. Traditional object detection methods often require large amounts of labelled data, which can be challenging to obtain in the agriculture domain. Few-shot object detection offers a promising solution by enabling models to learn from limited examples. This study investigates the effectiveness of few-shot object detection techniques using the PlantVillage dataset and YOLOv5 as the model backbone. The model was fine-tuned and compared to the Frustratingly Simple Few-Shot Object Detection (FSFSOD) method. The adapted YOLOv5 model with certain setup conditions, achieved a mean average precision (mAP) of 0.628 at IoU=0.5, outperforming the FSFSOD method on the PlantVillage dataset. This research demonstrates that the fine-tuned YOLOv5 model might effectively assist farmers in monitoring crop health and taking early action against diseases and pests. The findings contribute to the development of more efficient and data-scarce deep learning applications in agriculture. Future work may focus on extending the model to detect other plant diseases or pests, improving the model's generalization across various agricultural scenarios, and integrating the model into a real-time monitoring system for precision agriculture.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the first chapter, the introduction to the thesis is provided, offering a comprehensive understanding of the research context. The chapter begins by identifying the research gap in the field of interest, which highlights the unexplored areas or unresolved questions that the current study aims to address. Next, the objectives of the research are presented, which outline the specific goals and aims the study intends to achieve. Following that, the scope of the research is delineated, providing clarity on the extent and boundaries of the study.

Moreover, this introductory chapter serves as a guide for readers, as it presents the structure of the thesis, detailing how each subsequent chapter is organized and interconnected. By offering a clear roadmap of the thesis, readers can navigate through the document with ease, understanding how each chapter contributes to the overall narrative and purpose of the study. The first chapter, therefore, plays a pivotal role in setting the stage for the entire research, giving readers a strong foundation upon which to build their comprehension of the study.

1.1 Research gap

Object detection is an essential task in many agricultural applications, including plant disease detection, yield estimation, and crop monitoring. In particular, the detection of unhealthy or diseased leaves is crucial for identifying and mitigating plant diseases, which can cause significant losses in crop yield and quality. Traditional object detection methods typically require large amounts of annotated training data to achieve high accuracy, which can be challenging and expensive to obtain in the agricultural domain. This is especially true for rare or uncommon plant diseases, where there may be very little training data available. [73]

On the other hand, few-shot learning is a promising approach for addressing the data scarcity problem in agricultural object detection. Few-shot learning aims to learn a model that can recognize new classes of objects with very few labelled examples, by leveraging knowledge from previously seen classes. Despite its potential, few-shot object detection in agriculture, specifically for healthy and unhealthy leaf detection, is a relatively new research area with several gaps. One of the main challenges is developing effective few-shot learning algorithms that can adapt to the diverse range of leaf shapes, sizes, and colours that are present in agricultural images. Additionally, there is a lack of standardized benchmarks and datasets that can be used to evaluate and compare different few-shot object detection methods. [11] [16]

Another challenge in few-shot object detection in agriculture is the lack of interpretability and explainability of the models. It is important to understand how the model makes its predictions, especially in a domain such as agriculture where the consequences of misclassifying diseased leaves can be significant. There is a need for practical and scalable few-shot object detection methods that can be deployed in real-world agricultural settings. This requires addressing issues such as model efficiency, robustness to environmental factors (such as lighting and weather conditions), and ease of use by non-expert users.

1.2 Objective and scope

As a passionate AI researcher, I'm excited to focus my thesis on improving the interpretability and explainability of object detection models in agriculture associated with few-shot learning techniques. I believe that in order to trust and adopt AI-driven solutions, especially in critical domains like agriculture, people need to understand how these models make their decisions. My aim is to unravel the complex decision-making process behind detecting diseased leaves, ensuring that farmers and other stakeholders can have confidence in the technology.

The objective of this research is to find out the usage of few-shot object detection in the Agriculture

domain. By investigating the usability of few-shot detection in detecting healthy or unhealthy leaves, I aim to contribute to the understanding of how this approach can be applied to solve real-world problems in agriculture. To achieve my objective, I am going to conduct experiments using different setups and compare the performance of the models using different numbers of labelled samples for the few-shot dataset. By doing so, I shall evaluate how well the models can generalize to new, unseen classes with limited labelled data, and identify the minimum number of labelled samples needed to achieve good results. Additionally, I plan to investigate how the performance of the models varies depending on the specific characteristics of the few-shot dataset, such as the level of similarity between the few-shot classes and the base classes, the number of examples per class, and the level of intra-class and inter-class variation. To ensure a fair comparison between the different setups, I also want to use standardized evaluation metrics, such as mean average precision (mAP). By conducting experiments and analyzing the results, I will gain insights into how few-shot detection can be used to detect healthy or unhealthy leaves, and how to optimize the performance of the models with limited labelled data. These insights can inform the development of more effective and practical solutions for agricultural applications, and contribute to the advancement of few-shot learning research.

The scope of my research on **Few-shot Object Detection for Detecting Healthy or Unhealthy Leaves** would be quite broad and multidisciplinary. From an agricultural perspective, my research falls under the domain of precision agriculture, which is the use of technology and data-driven approaches to optimize crop production and management. Within this domain, this research specifically focuses on developing new approaches for detecting and diagnosing plant diseases, which is an important area of research with significant implications for food security and sustainable agriculture.

From a computer vision perspective, it falls under the domain of object detection, which is the task of identifying and localizing objects of interest within an image. Within this domain, it will focus specifically on developing new approaches for object detection that can learn from limited labelled data, which is a challenging and an active area of research known as few-shot learning.

In terms of the methodology, my research will involve developing and evaluating new algorithms and models for few-shot object detection in agriculture, and conducting experiments to validate the effectiveness of these approaches. I also aim to carefully design my experiments, collect and label data, and use appropriate evaluation metrics to measure the performance of the models.

Overall, this research would contribute to finding out the intersection of computer vision and agriculture, and will most certainly have practical implications for improving crop management and reducing the impact of plant diseases on agricultural productivity.

1.3 Structure of the Thesis

The structure of the thesis is designed to provide a comprehensive overview of the research, with each chapter serving a distinct purpose. In this first chapter, the introduction, the focus is on identifying the research gap in the field of interest. This is crucial, as it sets the stage for the entire study by outlining the context and rationale for the research. The objectives and scope of the research are also presented, which provide a roadmap for the reader and offer clear expectations for what the study aims to achieve.

The second chapter delves into the background and related work that informs the current research. A thorough literature review is conducted, covering the theoretical background, previous studies, current approaches, and challenges in the field. This chapter helps to establish the foundation upon which the current research is built, demonstrating how it is situated within the larger body of knowledge. Additionally, this chapter presents relevant methodologies and techniques that have been employed in similar research, helping to set the stage for the research design.

In the third chapter, the research design is detailed, offering a complete picture of the methodology used to carry out the study. This includes presenting the research questions or hypotheses, experiment setups, data collection methods, and data analysis techniques.

The fourth and final chapter focuses on the results and discussion of the research findings. It begins with an overview of the findings. Then the interpretation and discussion of these results are presented, answering the research questions or hypotheses, and highlighting the implications for real-world applications. This chapter also acknowledges the limitations of the study and offers suggestions for future research.

This thesis is structured to provide a logical and coherent presentation of the research, from the identification of the research gap to the interpretation and discussion of the results. The bibliography and appendices, which include the codes used in the research, details of the dataset, and elaborated results from experiments, serve as valuable supplementary materials to support and enhance the reader's understanding of the research.

Chapter 2

Background & Related work

In this chapter, the background topics and related work pertaining to the research are discussed in depth to provide readers with a comprehensive understanding of the relevant concepts and methodologies. The research topic is few-shot object detection in the domain of agriculture, and the chapter will delve into various interconnected topics, outlining their significance and connection to the research. The chapter begins with a discussion on object detection, exploring its fundamental principles, techniques, and applications. This foundational knowledge sets the stage for understanding the more specialized topic of few-shot object detection. Next, the concept of domain adaptation is introduced, highlighting its importance in applying machine learning models to different scenarios, particularly in agriculture.

Following this, few-shot learning is discussed in general terms, explaining its core principles and methodologies, as well as its significance in addressing the limitations of traditional machine learning techniques. Building upon this foundation, few-shot classification is examined, focusing on the specific challenges and techniques involved in this sub-domain of few-shot learning.

The chapter then delves into the main topic of few-shot object detection, discussing the state-of-the-art techniques, algorithms, and challenges associated with it. This section provides a comprehensive understanding of the current landscape of few-shot object detection and its relevance to the research.

Finally, the chapter addresses few-shot learning in the agriculture domain, exploring its unique challenges, potential applications, and recent advancements. By highlighting the significance of few-shot learning in this specific context, the chapter establishes the importance and relevance of the research topic, setting the stage for the subsequent chapters on research design, results, and discussion.

2.1 Object detection

2.1.1 Object Detection

Object detection is a fundamental task in computer vision, which involves detecting and localizing objects within an image or video. In recent years, significant progress has been made in object detection, thanks to advances in machine learning, specifically deep learning.

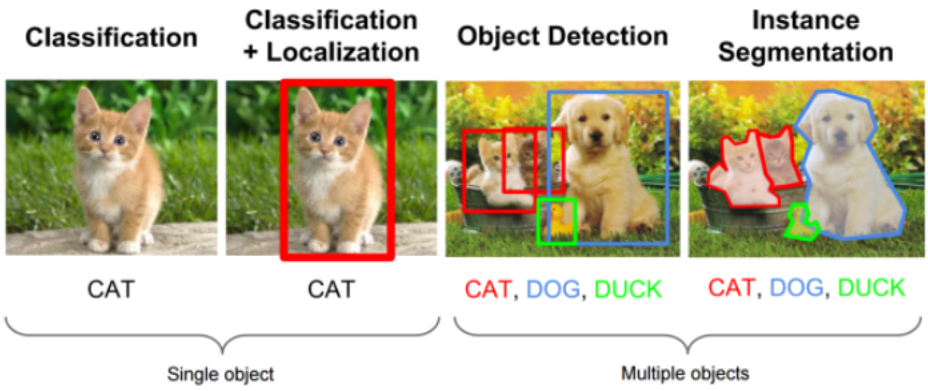


Figure 2.1: A Comprehensive Idea

Figure 2.1 [32] gives a comprehensive idea about what object classification, localization, and segmentation look like.

There are several approaches to object detection, but two common methods are the Single Shot Detector (SSD) and Two Shot Detector (TSD).

1. **Single Stage Detector**

   The Single Shot Detector (SSD) [45] is a type of neural network architecture that performs object detection in a single pass through an image. It is a popular method in the field of computer vision due to its speed and accuracy in detecting objects. SSDs use a convolutional neural network (CNN) [41] to extract features from an image and predict bounding boxes for objects in a single forward pass.

   The architecture of SSD consists of a base network, followed by multiple convolutional layers, and a set of detection heads that predict object class probabilities and the coordinates of bounding boxes. The detection heads are responsible for predicting the location and class of objects in the input image. The SSD architecture also uses feature maps at different scales to capture objects of different sizes and aspect ratios.

   SSD has shown to be effective and efficient for object detection tasks, achieving state-of-the-

art results in benchmarks such as the COCO dataset. It is also commonly used in applications such as self-driving cars, surveillance systems, and robotics.

"SSD: Single Shot MultiBox Detector" by Wei Liu et al. (2016): [45] This paper introduced the original SSD architecture that used a single CNN to predict object class probabilities and bounding boxes in a single forward pass. The authors demonstrated that SSD achieved state-of-the-art performance on object detection benchmarks such as PASCAL VOC [15] and COCO [43].

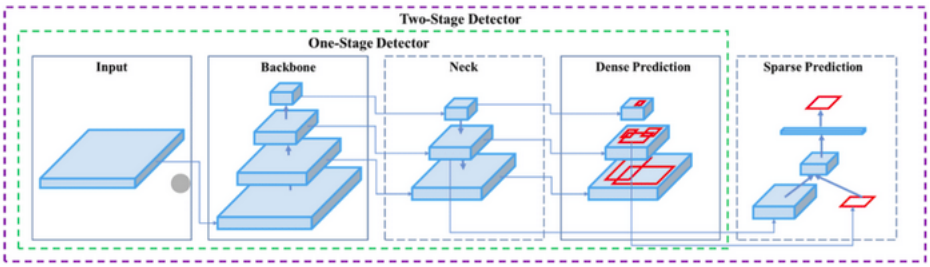


Figure 2.2: Two stage detector vs. Single shot detector

The figure 2.2 shows the difference between Single Shot Detector (SSD) [45] and Two-stage Detector.

2. **Two Stage Detector**

   A two-stage detector is an object detection model that consists of two main stages: region proposal and object classification. In the first stage, the model proposes a set of candidate regions likely to contain objects of interest using a region proposal network (RPN) [19]. In the second stage, these candidate regions are classified into object or background classes using a classifier.

   Two-stage detectors have been popular in object detection due to their accuracy and robustness. They are particularly effective for detecting small objects and objects in cluttered scenes. Some popular two-stage detectors include Faster R-CNN, R-CNN, and Mask R-CNN.

   - R-CNN: The Region-based Convolutional Neural Network (R-CNN) [20] was introduced in 2014, and it is considered one of the first successful deep learning approaches for object detection. R-CNN operates in two stages: first, it generates region proposals, and then it classifies each proposal as an object or not. R-CNN achieved state-of-the-art results on the PASCAL VOC [15] benchmark.
   - Fast R-CNN: Fast R-CNN [19], introduced in 2015, is an improvement over R-CNN that performs object detection in a single stage. It uses a Region of Interest (RoI) pooling layer to extract fixed-size features from each proposal, which are then fed into a fully connected network for classification and bounding box regression.
   - Faster R-CNN: Faster R-CNN is a popular two-stage detector that was proposed in 2015 by Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun [58]. The model uses a region proposal network (RPN) to propose candidate regions and a Fast R-CNN classifier to classify objects. Faster R-CNN has achieved state-of-the-art performance on a number of object detection benchmarks.
   - Mask R-CNN: Mask R-CNN is an extension of Faster R-CNN that was proposed in 2017 by Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick [25]. In addition to object detection, Mask R-CNN also predicts object masks, which provides pixel-level segmentation of objects. Mask R-CNN has achieved state-of-the-art performance on several object detection benchmarks, including COCO [43] and Pascal VOC [15].

2.1.2 Performance Metrics and Common Techniques

- **Intersection over Union (IoU)**:
  Intersection over Union is a metric used to evaluate the accuracy of object recognition and segmentation algorithms (IoU). [15] It measures the overlap between the predicted bounding box or mask and the ground truth bounding box or mask.

  The IoU is defined as the ratio of the area of intersection between the predicted and ground truth regions to the area of union between them.

  It is expressed as:

  $$\text{IoU} = (\text{Area of Intersection}) / (\text{Area of Union})$$

  where the area of intersection is the area common to both the predicted and ground truth regions, and the area of union is the total area covered by both regions. IoU ranges from 0 to 1, with higher values indicating better agreement between the predicted and ground truth regions. A value of 1 means the predicted and ground truth regions are identical, while a value of 0 means there is no overlap between them. IoU is commonly used in computer vision tasks such as object detection, instance segmentation, and semantic segmentation to evaluate the performance of algorithms.

- **NMS**:
  Non-Maximum Suppression (NMS) is a post-processing algorithm used in object detection to eliminate redundant detections. It works by selecting the highest-scoring detection and suppressing all other detections that overlap with it significantly. The NMS algorithm takes as input a set of bounding boxes or regions of interest (ROIs) and their corresponding confidence scores [51].

  The steps of the algorithm are as follows:

  1. Sorting the ROIs based on their confidence scores in descending order.
  2. Selecting the ROI with the highest confidence score and output it as a detection.
  3. Removing all ROIs that overlap significantly with the selected ROI, using a threshold overlap score.
  4. Repeating steps 2 and 3 until there are no more ROIs left.

  The IoU threshold overlap score is usually set to a value between 0.3 and 0.5, depending on the application and dataset. The NMS algorithm reduces the number of detections while preserving the highest-scoring and non-overlapping ones. It improves the efficiency and accuracy of object detection systems.

- **mAP**:

  Mean Average Precision (mAP) is a popular evaluation metric used to measure the accuracy of object detection and instance segmentation models. It measures both the precision and recall of the model across all classes and generates a single scalar value to represent the overall performance of the model.

  The mAP is calculated as the mean of the average precision (AP) values for each class. The AP for a class is computed by first calculating the precision and recall values at different confidence score thresholds for the class. The precision and recall values are then used to create a precision-recall (PR) curve, which is a plot of precision against recall at different confidence score thresholds. The AP for a class is the area under the PR curve. The mAP is calculated as:

  $$\text{mAP} = (\text{AP1} + \text{AP2} + ... + \text{APn}) / \text{n}$$

  where n is the number of classes and APi is the average precision for the ith class.

  The mAP ranges from 0 to 1, with higher values indicating better performance. A value of 1 means that the model has perfect precision and recall across all classes. The mAP is commonly used in object detection and instance segmentation competitions such as the COCO (Common Objects in Context) and PASCAL VOC (Visual Object Classes) challenges [15].

2.1.3 State-of-the-art: YOLO

The YOLO (You Only Look Once) [55] algorithm is a series of real-time object detection models that are designed for speed and efficiency. These models have evolved over time, with each version introducing improvements and optimizations. Here's an overview of the YOLO algorithm, from the original version to YOLOv4:

YOLO (Original): The original YOLO algorithm divides an input image into a grid and predicts bounding boxes and class probabilities simultaneously using a single convolutional neural network (CNN). It is known for its speed and ability to process object detection tasks in real time. However, the original YOLO had some limitations, including difficulties with small object detection and suboptimal localization accuracy.[55]

YOLOv2: YOLOv2[56] addressed some of the limitations of the original YOLO, introducing several improvements such as:

Better localization using anchor boxes, which helped the model handle varying object dimensions and aspect ratios more effectively, increased resolution (416x416) for better small object detection, multi-scale training to allow the model to adapt to various object scales, batch normalization in the CNN layers for faster convergence and higher overall accuracy.

YOLOv3: YOLOv3[57] further improved the YOLO algorithm with enhancements like:

A deeper and wider network architecture (Darknet-53) for better feature extraction, introduction of multi-scale predictions with three different scales to increase detection accuracy across a range of object sizes, integration of logistic regression for class predictions instead of softmax, which improved performance in scenarios with multiple overlapping objects of different classes.

YOLOv4: YOLOv4[6] is a significant leap forward in terms of accuracy and efficiency. Some key advancements in YOLOv4 include:

A new backbone network, CSPDarknet53, which uses Cross Stage Hierarchical Networks (CSH) for improved feature fusion and representation, integration of multiple advanced techniques such as Bag of Freebies (BoF) like new data augmentation techniques, new loss function CIoU, and Bag of Specials (BoS) like PANet [44] and SPP [26] blocks in the neck of the network, using CSPDarknet53 which combines the CSPNet with Darknet53 [57] architecture to optimize the model's performance and speed. The use of the Mish activation function, has been shown to improve model accuracy without a significant increase in computational cost.

Throughout its evolution, the YOLO algorithm has consistently focused on achieving real-time object detection while improving accuracy and performance. From the original YOLO to YOLOv4, each version has introduced new techniques and optimizations to advance the state of the art in object detection.

YOLOv5:

YOLOv5 is the fifth iteration of the YOLO (You Only Look Once) algorithm for object detection. It was developed by Ultralytics and builds upon the previous versions of the algorithm by introducing several new features and improvements. [35]

- YOLOv5 Architecture:

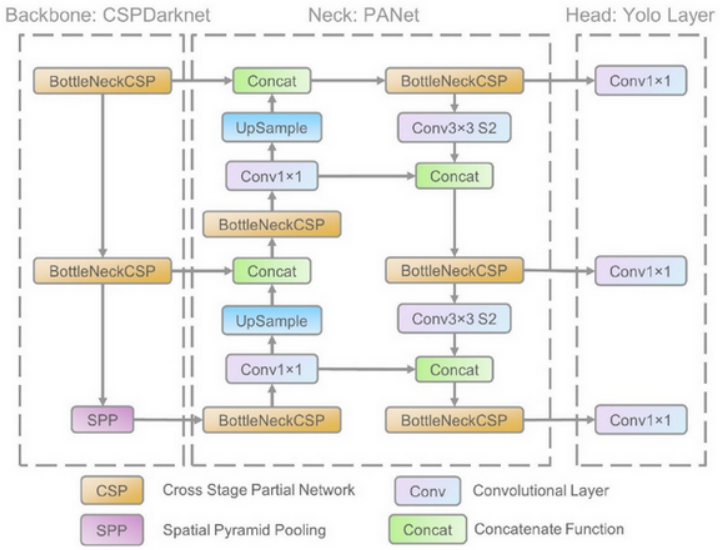


Figure 2.3: Network Architecture of YOLOv5 [31] [79]

Backbone:

The backbone of an object detection model like YOLOv5 is part of the architecture that primarily focuses on feature extraction. It is a convolutional neural network (CNN) [41] that processes the input image and extracts meaningful features at various scales, which are then used by the rest of the model to perform object detection.

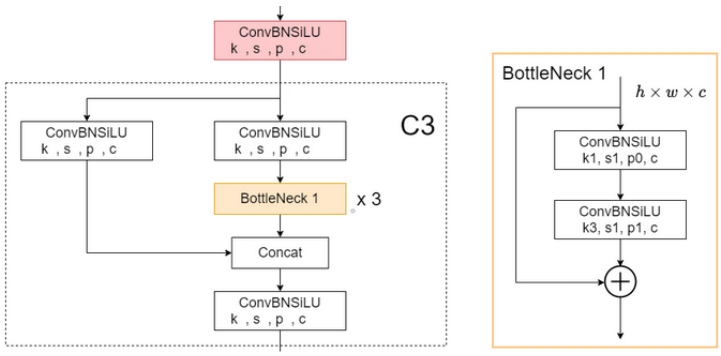


Figure 2.4: BottleNeckCSP module architecture [31] [35]

In YOLOv5, the backbone is based on the CSPNet (Cross Stage Partial Network) architecture. [72] The CSPNet divides the feature map into two parts at various stages of the network, with one part being processed and the other part is skipped. The processed part is then merged back with the skipped part through a cross-stage hierarchy. This approach allows for better gradient flow, which in turn improves the learning capability of the model.

The CSPNet architecture in YOLOv5 is based on CSPDarknet53, an enhanced version of Darknet53. [57] Darknet53 was the backbone used in previous YOLO versions like YOLOv3. CSPDarknet53 improves upon Darknet53 by incorporating the CSPNet technique, which enhances the model's learning capability while maintaining computational efficiency.

The benefits of using the CSPNet-based backbone in YOLOv5 compared to previous versions include:

- Better gradient flow: The cross-stage hierarchy in CSPNet allows for more effective gradient flow during backpropagation, which helps the model learn more efficiently and achieve better performance.
- Enhanced learning capability: By dividing the feature map into two parts and processing them separately, CSPNet enables the model to learn more discriminative features, leading to improved object detection accuracy.
- Computational efficiency: The CSPNet architecture maintains the computational efficiency of Darknet53 while providing improved learning capability, making YOLOv5 a faster and more accurate object detection model compared to its predecessors.

Neck:

The neck of an object detection model is responsible for connecting the backbone (feature extraction) with the head (prediction). It helps aggregate and refine the features extracted by the backbone before passing them to the head for final prediction. In YOLOv5, the neck consists of two main components: SPP (Spatial Pyramid Pooling) [26] and PANet (Path Aggregation Network) [44].

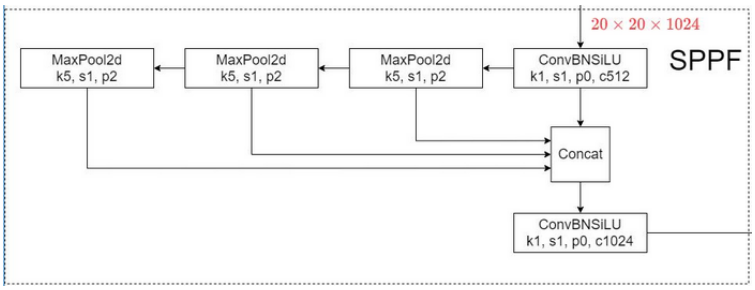


Figure 2.5: Structure of SPPF block[31] [35]

SPP (Spatial Pyramid Pooling): SPP is a technique that allows the model to handle objects of different scales and aspect ratios more effectively. It applies pooling operations (e.g., max pooling) at multiple scales to the input feature map, and then concatenates the resulting pooled feature maps. This results in a multi-scale feature representation, which helps the model detect objects at various sizes and aspect ratios. [26] Mathematically, SPP can be represented as:

Given an input feature map F, SPP applies a series of pooling operations P_1, P_2, ..., P_n with different kernel sizes k_1, k_2, ..., k_n and strides s_1, s_2, ..., s _n. The pooled feature maps are then concatenated to form the output feature map F':

$$(F' = concatenate(P_1(F), P_2(F), ..., P_n(F))$$

PANet (Path Aggregation Network): PANet is responsible for aggregating feature maps across different scales in the network. It does this by connecting the feature maps from different layers in a top-down and bottom-up manner. This helps the model incorporate context information from both low-level (fine-grained) and high-level (coarse-grained) features. [44] The aggregation of feature maps in PANet can be represented as:

Given feature maps F_1, F_2, ..., F_n at different scales, PANet generates aggregated feature maps A_1, A_2, ..., A_n through a combination of lateral connections (L), top-down connections (T), and bottom-up connections (B):

$$A\_1 = L(F\_1) + T(A\_2)$$

$$A\_i = L(F\_i) + T(A\_(i+1)) + B(A\_(i-1))$$

$$A\_n = L(F\_n) + B(A\_(n-1))$$

By incorporating SPP [26] and PANet [44] in the neck of YOLOv5, the model is better equipped to handle objects of varying scales and aspect ratios. This leads to improved detection accuracy compared to previous versions of YOLO, which did not have these components in their architecture.

Simply put, the neck of YOLOv5 refines and aggregates the features extracted by the backbone before passing them to the head for object detection. SPP helps the model handle objects of different scales and aspect ratios by pooling features at multiple scales. At the same time, PANet aggregates features across different layers to incorporate context information from low-level and high-level features. These components make YOLOv5 [35] more accurate in detecting objects compared to previous versions.

Head:

The head of an object detection model makes the final predictions based on the features extracted by the backbone and processed by the neck. In the case of YOLOv5, the head generates predictions for object classes, objectness scores, and bounding box coordinates. [55] [56]

In simple terms, the head of YOLOv5 does the following:

Predicts object classes: For each grid cell in the feature map, the model predicts the probability of each object class (e.g., "car," "person," "dog"). This allows the model to determine which class of object, if any, is present in each region of the input image.

Predicts objectness scores: For each grid cell and anchor box, the model predicts an objectness score, which represents the likelihood that an object is present in the bounding box defined by the grid cell and anchor box. This helps filter out false positives and reduce the number of bounding boxes that need to be processed.

Predicts bounding box coordinates: For each grid cell and anchor box, the model predicts the coordinates of the bounding box (e.g., the centre coordinates, width, and height) that best encloses the detected object.

Compared to previous YOLO versions, the head of YOLOv5 is quite similar in terms of its overall function. It still predicts object classes, objectness scores, and bounding box

coordinates for each grid cell and anchor box. However, there may be some differences in the implementation details, such as the number of anchor boxes used, the loss functions, and the prediction decoding process.

Loss function:

In YOLOv5, the loss function consists of three main components: classification loss, objectness loss, and box regression loss. These losses are calculated for each grid cell and anchor box in the prediction output. [35]

Classification loss: This loss measures the difference between the predicted class probabilities and the ground truth class labels. Typically, a cross-entropy loss is used for classification loss. For each grid cell and anchor box, the classification loss is computed only if there is an object present in that anchor box (i.e., the objectness score is 1).

Objectness loss: This loss measures the difference between the predicted objectness scores (how likely an object is present in the bounding box) and the ground truth objectness scores. The loss can be computed using binary cross-entropy loss or another suitable loss function. Objectness loss is calculated for each grid cell and anchor box, considering both positive and negative samples.

Box regression loss: This loss measures the difference between the predicted bounding box coordinates and the ground truth bounding box coordinates. The loss is typically computed using the mean squared error (MSE) or other suitable distance metrics. Box regression loss is calculated for each grid cell and anchor box, but only for those anchor boxes containing an object (i.e., the objectness score is 1).

The total loss for YOLOv5 is the weighted sum of the classification loss, objectness loss, and box regression loss:

$$\text{Total\_loss} = \lambda\_\text{cls} * \text{Classification\_loss} + \lambda\_\text{obj} * \text{Objectness\_loss} + \lambda\_\text{box} * \text{Box\_regression\_loss}$$

Several types of YOLOv5 models are available, each designed to address different requirements and constraints. Some of the most commonly used YOLOv5 models as shown in 2.6 include:

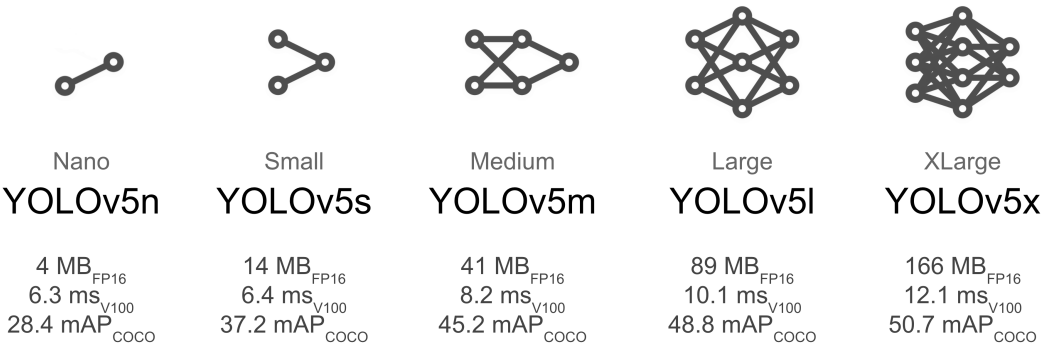


Figure 2.6: Models in YOLOv5

- YOLOv5n: YOLOv5n is a variant of the YOLOv5 algorithm that uses a more minor neural network architecture called YOLOv5-Nano. This architecture is designed to be even smaller and faster than the YOLOv5s model, making it suitable for low-power devices such as embedded systems and smartphones.
- YOLOv5s: This is the smallest and fastest YOLOv5 model for low-power devices and real-time applications. It has fewer layers and parameters than the other YOLOv5 models but still achieves high accuracy.
- YOLOv5m: This is a medium-sized YOLOv5 model, designed to balance speed and accuracy. It has more layers and parameters than YOLOv5s, but is still fast enough for real-time applications.
- YOLOv5l: This is a larger and more accurate YOLOv5 model, designed for applications that require higher precision. It has even more layers and parameters than YOLOv5m, but is slower and requires more computational resources.
- YOLOv5x: This is the most extensive and accurate YOLOv5 model, designed for applications that require the highest precision. It has the most layers and parameters of any YOLOv5 model but is also the slowest and most computationally intensive.

2.2 Domain Adaptation

Domain adaptation is a subfield of machine learning that focuses on adapting models trained on one domain to perform well on another related but different domain. Domain adaptation is a challenging task due to the fact that the training and test data are not drawn from the same distribution. This leads to a mismatch between the distribution of the data used during training and the distribution of the data the model will encounter at test time. Therefore, models trained on one domain may not generalize well to another domain. Domain adaptation methods aim to address this challenge by reducing the mismatch between the training and test distributions.

2.2.1 Types

The several types of domain adaptation techniques are:

- **Transfer Learning:**:

  One of the most widely used techniques for domain adaptation is transfer learning. In transfer learning, a model is first trained on a large dataset in a source domain, and then the learned knowledge is transferred to a related but different target domain. The transfer can be achieved by freezing some or all of the model layers or by fine-tuning the model on the target domain data. Transfer learning is a machine learning technique where a pre-trained model is used as a starting point for a new task or problem. In transfer learning, the knowledge learned by the pre-trained model on a large dataset is transferred to a smaller, more specific dataset for a different task. This is done by using the pre-trained model as a feature extractor, and training a new model on top of these extracted features. Transfer learning has become an increasingly popular technique in machine learning, as it can save time and computational resources, and improve performance on smaller datasets. Some common applications of transfer learning include image classification, natural language processing, and speech recognition.

  Here's an example of how transfer learning can be used in image classification:

  Let's say we have a pre-trained convolutional neural network (CNN) [41] that was trained on a large dataset of images, such as ImageNet [14]. This pre-trained model has learned to recognize general features of images, such as edges, textures, and shapes. Now, let's say we want to use this pre-trained model to classify images of dogs and cats. We can use the pre-trained model as a feature extractor, by passing each image through the pre-trained model and extracting the output from one of the last layers of the network. These extracted features can then be used as input to a new, smaller neural network that is trained to classify images of dogs and cats. The new neural network only needs to learn to classify based on these

pre-extracted features, rather than learning from scratch on a small dataset of dogs and cats. This can save a lot of time and computational resources, and often leads to better performance than training a new model from scratch on a small dataset.

Some popular pre-trained models that are often used for transfer learning include:

  - VGG (Visual Geometry Group) [63] models: a family of CNN models developed by the Visual Geometry Group at the University of Oxford
  - ResNet (Residual Network) [27] models: a family of CNN models developed by Microsoft Research
  - Inception models [66]: a family of CNN models developed by Google

- **Adversarial Training**: Another popular approach to domain adaptation is adversarial training, which involves training a model to predict a label while also training a domain discriminator to predict whether the input data is from the source or target domain. The model is then optimized to fool the discriminator, forcing it to learn features that are invariant across domains. Adversarial training is a technique used in deep learning to improve the robustness of a neural network to adversarial attacks. Adversarial attacks involve deliberately modifying the input to a neural network in such a way that it causes the network to make a mistake, even when the original input was correctly classified. Adversarial training involves augmenting the training data with adversarial examples, which are examples that have been intentionally crafted to cause the neural network to make a mistake. By including these examples in the training data, the network learns to be more robust to such attacks. The basic idea behind adversarial training is to train the network to not only correctly classify the input data, but also to correctly classify the perturbed data. This is achieved by adding an additional network, called the "adversary", that is trained to generate perturbed examples that are difficult for the original network to classify correctly. The original network is then trained to correctly classify both the original and perturbed examples. [18]
- **Domain Separation Networks**: Domain Separation Networks (DSN) [7] is a type of deep neural network architecture designed to learn shared representations across different domains, while also preserving domain-specific information. The model is trained to classify data while also minimizing the domain discrepancy between the source and target domains. This approach is effective when the domains share some common features but also have domain-specific features. This can be useful in tasks such as transfer learning, where the knowledge learned from one domain can be transferred to another related domain. The basic idea behind DSN is to have separate branches for each domain, which share some layers to learn shared representations while having some separate layers to capture the domain-specific features. These separate branches are then merged together, and the shared representations are passed

through additional layers to make the final prediction.

- **Dataset Shift**: Dataset shift is another important challenge in domain adaptation. This occurs when the marginal distribution of the features changes between the training and test data, leading to a shift in the data distribution. Various techniques have been proposed to address this challenge, including importance weighting, re-weighting, and kernel mean matching. Dataset shift is a phenomenon that occurs when the distribution of the training data is different from the distribution of the test data. This can cause problems in machine learning models, as the model may not perform well on the test data even if it performs well on the training data. In domain adaptation, dataset shift refers specifically to the situation where the training and test data come from different domains, i.e., different distributions.[54] Domain adaptation aims to overcome this problem by learning a model that is robust to the differences between the training and test domains. This is typically achieved by learning a shared representation between the domains, which captures the common features of both domains, while also learning domain-specific features.
- **Unsupervised Domain Adaptation**: Unsupervised domain adaptation is a challenging scenario where no labelled data is available in the target domain. In this case, techniques such as domain-adversarial training and DSNs have been successfully applied. However, recent advances in generative models have also shown promising results for unsupervised domain adaptation. Unsupervised domain adaptation is a type of domain adaptation in which the target domain has no labelled data. In this scenario, the aim is to learn a model that can perform well on the target domain using only the labelled data from a different, but related source domain. The key challenge in unsupervised domain adaptation is to learn a representation that is invariant to the domain shift, i.e., that captures the common features between the source and target domains, while also preserving the domain-specific information. One approach to unsupervised domain adaptation is to use adversarial training [18], where an additional network, called the domain discriminator, is trained to distinguish between the source and target domains. The main network is then trained to generate features that are difficult for the domain discriminator to distinguish, which encourages the network to learn domain-invariant representations. Another approach is to use a maximum mean discrepancy (MMD) loss [23], which measures the distance between the distributions of the source and target domains in the feature space. The model is trained to minimize this distance, which encourages it to learn features that are similar across domains.
- **Semi-supervised Domain Adaptation**: Semi-supervised domain adaptation is a scenario where some labelled data is available in the target domain. In this case, techniques such as fine-tuning [39] and model adaptation have been successfully applied. Semi-supervised

domain adaptation is a type of domain adaptation that aims to leverage both labelled and unlabeled data in the source and target domains to improve model performance. The goal is to learn a model that can perform well on the target domain, even when only a limited amount of labelled data is available. In semi-supervised domain adaptation, the labelled data in the source domain is used to learn a model that can generalize to the target domain, while the unlabeled data in both domains is used to learn a more robust and domain-invariant feature representation. One approach to semi-supervised domain adaptation is to use a combination of supervised and unsupervised learning techniques, such as adversarial training [18] or MMD loss [23], as in unsupervised domain adaptation. However, in semi-supervised domain adaptation, the labelled data in the source domain is used to provide additional supervision during the training process. Another approach is to use a technique called "pseudo-labelling", where the model is first trained on the labelled data in the source domain, and then used to generate labels for the unlabeled data in the target domain. These pseudo-labels are then used to train the model further on the target domain.

Overall, domain adaptation is an important and challenging task in machine learning, and various techniques have been proposed to address this challenge. Transfer learning, adversarial training, DSNs, and dataset shift are some of the most important approaches in domain adaptation. The choice of technique depends on the specific problem at hand and the availability of labelled data in the target domain.

2.2.2 Transfer Learning vs. Meta Learning

Table 2.1: Transfer Learning vs. Meta Learning

Aspect	**Transfer Learning**	**Meta Learning**
Goal	Leverage knowledge from a pre-trained model to perform well on a related task. [53] [84]	Design models that can quickly adapt to new tasks with minimal training. [17] [67]
Approach	Fine-tune a pre-trained model with data from the target task.	Incorporate prior knowledge and experience into the model's structure or algorithm.
Key-Techniques	Feature extraction, fine-tuning.	MAML (Model-Agnostic Meta-Learning), Reptile, Prototypical Networks.
Use Cases	Tasks with limited data, domain-specific applications.	Rapid adaptation to new tasks, learning with limited labelled data.
Task Relationship [60]	Assumes a strong relationship between source and target tasks, often from the same domain.	Assumes tasks are related but does not require a strong relationship between them.
Data Requirements [9]	Can be effective with relatively small amounts of data for the target task, as it leverages a pre-trained model.	Typically requires more diverse data across multiple tasks to learn a more generalizable model.
Generalizability [3]	May not generalize well to tasks that are very different from the source task.	Aims to learn a more generalizable model that can adapt to a wider range of tasks.
Optimization [52]	Involves fine-tuning a pre-trained model, which can be computationally less expensive.	Meta-learning optimization techniques (like MAML) can be more computationally expensive.

2.3 Few-shot Learning

Few-shot learning is a subfield of machine learning that focuses on building models that can learn new tasks from minimal data. In traditional machine learning, a model is trained on a large dataset containing many labelled examples per class. However, in many real-world scenarios, collecting a vast amount of labelled data for each class can be difficult, time-consuming, or expensive. Few-shot classification aims to tackle this issue by developing algorithms that can effectively learn from a small number of samples.

The few-shot problem is commonly formulated in the context of meta-learning, where models are trained on a series of tasks called episodes. Each episode consists of a support set and a query set, both containing instances from a small number of classes. The support set is used to adapt the model to the new classes, while the query set is used to evaluate the model's performance on the unseen instances of those classes. Traditional machine learning algorithms require a large amount of labelled data to learn, but few-shot learning algorithms aim to learn from a few examples (i.e., a few shots) of each class or task.

For example, in a 5-shot 5-ways classification task, the model is trained on a set of tasks consisting of five examples from five different classes. The model is then evaluated on a set of tasks where each task consists of five examples from five previously unseen classes. The goal is to classify each of the five examples into one of the five classes.

"n-shot k-ways" evaluation is a widely used evaluation metric in few-shot learning, where the goal is to evaluate the performance of a model in classifying new classes with few examples. In n-shot k-ways classification, the model is presented with a set of n examples from k previously unseen classes and asked to classify new examples into these k classes.

This evaluation protocol is designed to test how well a model can learn new classes with only a few examples. It is a challenging task that requires the model to generalize from a few examples. It also allows researchers to compare the performance of different models on a standard benchmark dataset.

The n-shot k-ways evaluation protocol has been used in various few-shot learning algorithms, such as Prototypical Networks and Model-Agnostic Meta-Learning (MAML) [17]. These algorithms have achieved state-of-the-art performance on different benchmark datasets, such as Omniglot [40] and Mini-ImageNet [71].

2.3.1 Supervised learning vs. Few-shot Learning

Supervised learning is a machine learning algorithm where the model is trained on labelled data to learn the relationship between input variables and their corresponding output variables. In other words, the model learns to map input features to their correct output labels, given a set of labelled examples. Supervised learning is typically used in applications where the goal is to predict an outcome based on given input variables, such as in image classification or natural language processing [21].

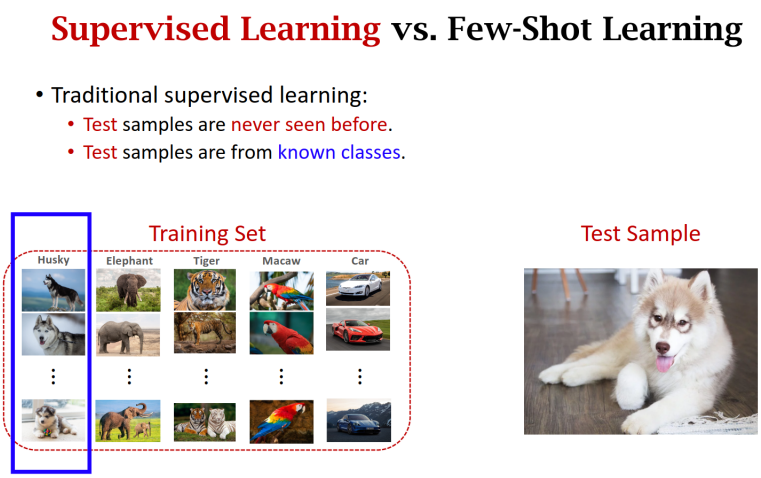


Figure 2.7: Supervised learning vs. FSL part_1

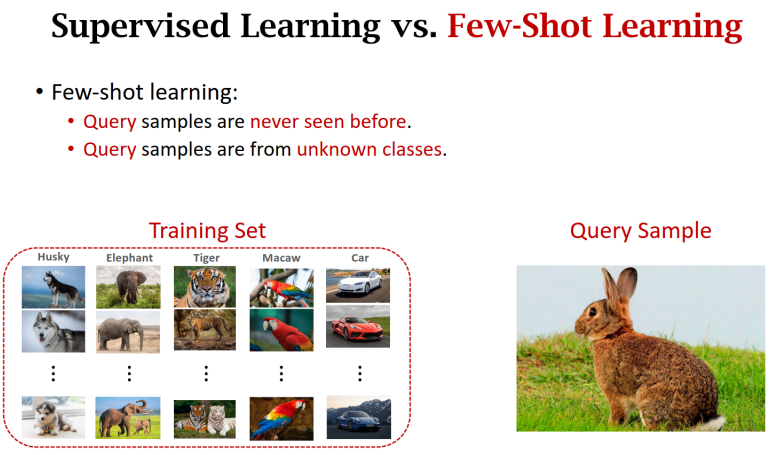


Figure 2.8: Supervised learning vs. FSL part_2

Figure 2.7 2.8 shows a simple explanation of the difference between traditional supervised learning and few-shot learning. [74]

Few-shot learning, on the other hand, is a type of machine learning algorithm that focuses on learning from only a few examples of a new category or concept. This is in contrast to traditional machine learning algorithms that require large amounts of labelled data to learn effectively. Few-shot learning has gained significant attention in recent years due to its ability to learn new concepts quickly and efficiently, making it well-suited for applications such as robotics, self-driving cars, and personalized medicine. [61] [71]

One of the key differences between supervised learning and few-shot learning is the amount of labelled data required to train a model. Supervised learning typically requires large amounts of labelled data to train a model effectively, whereas few-shot learning can learn from a few examples, making it more efficient and cost-effective. Another key difference between supervised learning and few-shot learning is the way they handle new or previously unseen data. In supervised learning, the model is typically retrained on new data to ensure that it can generalize well to unseen examples. In contrast, few-shot learning is designed to generalize well to new data, even if it has not seen it before, making it more robust and adaptable. [5]

In recent years, there has been significant research into few-shot learning, with several new models and algorithms proposed. One of the most popular few-shot learning models is the Siamese Neural Network, which uses a twin neural network architecture to learn similarities between images. Another popular model is the Prototypical Network [64], which learns a metric space to represent different categories and can classify new data by computing the distance between it and the category prototypes.

2.3.2 Approches

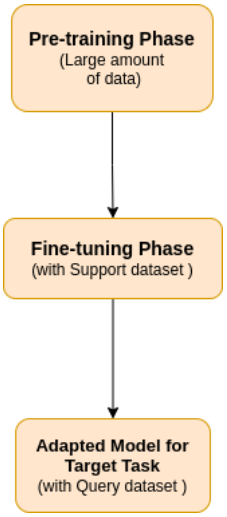


Figure 2.9: General approach to few-shot learning

The following figure 2.9 shows the two main steps involved in the few-shot learning approach using pre-training and fine-tuning:

Pre-training Phase: Train a model with abundant data, often from a similar or related domain.

Fine-tuning Phase (Few-shot Learning): Fine-tune the pre-trained model using a small number of examples from the target task (e.g., 1, 2, ... 10 samples). After the fine-tuning phase, the adapted model can be used for the target task. [75]

There are several approaches to few-shot learning, but one common approach is to use meta-learning. Meta-learning involves training a model on a set of tasks so that it can learn to learn new tasks more efficiently. Specifically, the model is trained on a set of "meta-train" tasks and then tested on a set of "meta-test" tasks. During training, the model learns to quickly adapt to new tasks by updating its weights based on the few examples of each task.

One popular few-shot learning model that uses meta-learning is called MAML (Model-Agnostic Meta-Learning), which was introduced by Finn et al. in 2017 [17]. MAML trains a model to learn how to update its weights for each task so that it can quickly adapt to new tasks with few examples.MAML has been applied to various tasks, including image classification, regression, and reinforcement learning.

Other approaches to few-shot learning include metric-based learning, which learns a distance metric between examples to compare them and classify new examples, and generative models, which learn to generate new examples from a small set of examples.

Matching Networks: Vinyals et al. proposed matching networks in 2016 [71], which is another

metric-based approach to few-shot learning. Matching networks use a "memory-augmented neural network" to learn a similarity function between examples. During training, the model is trained to match each example to its corresponding class, and during testing, it uses the similarity function to classify new examples.

Meta-learning with Memory-Augmented Neural Networks: Santoro et al. introduced Memory-Augmented Neural Networks (MANNs) for few-shot learning in 2016 [61]. MANNs use an external memory module to store information about each class, which allows them to generalize to new classes with few examples. During training, the model is trained to read and write to the memory module, and during testing, it uses the memory module to classify new examples.

Dynamic Few-Shot Visual Learning without Forgetting: Lee et al. introduced Dynamic Few-Shot Visual Learning without Forgetting in 2019 [83], which is a few-shot learning algorithm that learns to adapt to new tasks without forgetting previously learned tasks. The model uses a dynamic memory module to store information about each task and adapts to new tasks by updating the memory module.

Few-shot Learning via Embedding Adaptation with Set-to-Set Functions: Triantafillou et al. proposed Few-shot Learning via Embedding Adaptation with Set-to-Set Functions in 2020, which is a few-shot learning algorithm that learns to adapt embeddings of examples to new tasks. The model uses a set-to-set function to adapt the embeddings, which allows it to handle variable-sized sets of examples.[68]

Few-shot learning has many practical applications, such as in natural language processing, computer vision, and robotics. It has also been used in recent years to improve the performance of reinforcement learning algorithms, which traditionally require a large amount of data to learn.

2.3.3 Few-shot classification

1. Learning to Compare: Relation Network for Few-Shot Learning: This paper [62] proposes the Relation Network (RN) for few-shot learning, which learns to compare the similarity between examples in the embedding space. The method consists of two modules: an embedding module that generates feature representations for input examples, and a relation module that computes the similarity between those feature representations. The RN is trained end-to-end with episodic training, making it a highly competitive approach for few-shot classification tasks.

2. Prototypical Networks for Few-shot Learning:

In this [64] paper, the researchers introduce Prototypical Networks, a few-shot learning method based on metric learning. The idea is to learn a representation for each class prototype, which is the mean of the embedded support set instances for that class. During the inference, the class label for a query instance is assigned based on the nearest class prototype in the embedding space. Prototypical Networks are simple, efficient, and demonstrate strong performance in few-shot classification tasks.

3. TADAM: Task Dependent Adaptive Metric for Improved Few-Shot Learning:

one of the methods called TADAM extends Prototypical Networks by learning a task-dependent adaptive metric. [82] TADAM introduces a learnable scaling factor and a task conditioning mechanism to adapt the metric space based on the specific task. This adaptation enables the model to improve its few-shot classification performance, outperforming other state-of-the-art methods on various few-shot learning benchmarks.

2.3.4 Few-shot Object detection (FSOD)

Few-shot object detection is a subfield of computer vision that aims to detect new objects with only a few examples. Traditional object detection algorithms require a large number of annotated examples of the object of interest in order to learn a model that can detect it accurately. However, in real-world scenarios, it may be difficult or time-consuming to obtain such a large dataset.

Few-shot object detection addresses this challenge by using a small set of annotated examples to learn to detect new objects. The goal is to develop models that can quickly learn to detect new objects with few examples, similar to the way humans can recognize new objects with limited exposure. Few-shot object detection is an active and rapidly developing field, with many promising approaches and potential applications in various domains. It can enable more efficient and flexible object detection in scenarios where annotated data is limited, such as in medical imaging or robotics.

One of the earliest works in few-shot object detection is the one-shot object detection (OSOD) method proposed by Kang et al. (2018) [36]. OSOD is a framework that learns to detect novel objects using only one example. It achieves this by adapting a pre-trained object detector to new classes using a few-shot learning algorithm.

Meta-learning has emerged as a popular approach to few-shot object detection. MetaDet [80] is a meta-learning framework for object detection that learns to adapt to new object classes with few examples. It uses a meta-learner to learn a set of parameters that can be quickly adapted to new tasks.

Another meta-learning-based approach is Meta R-CNN [80], which introduced a meta-learning framework for object detection that enables fast adaptation to new object classes with only a few examples. It uses a meta-learner to learn an initialization that can be quickly adapted to new tasks.

Generative models such as Generative Adversarial Networks (GANs) [22] and Variational Autoencoders (VAEs) [38] have also been explored for few-shot object detection. One example is Few-Shot Object Detection with Attention-RPN and Multi-Relation Detector (ARD-MRD) [77], which introduced a generative adversarial network to generate synthetic images of the novel classes. These synthetic images are used to augment the few-shot training data and improve the model's performance.

One notable approach is to use attention mechanisms to improve few-shot object detection. Attention mechanisms can help the model focus on the most relevant parts of the input, which can be particularly useful when dealing with limited training data. For example, the Attention Augmented Few-Shot Object Detection (AAFOD) method [10]introduced a self-attention mechanism to improve feature representation and reduce intra-class variation in the few-shot support set.

Another approach is to incorporate prior knowledge about novel objects to improve few-shot object

detection. For instance, the Few-Shot Object Detection with Prior Knowledge (FSOD-PK) method [33] introduced a method that leverages prior knowledge in the form of object attributes to improve few-shot object detection performance.

Graph neural networks have also been explored for few-shot object detection. The Graph R-CNN [81] method introduced a graph neural network to capture the relationships between objects and improve few-shot object detection performance. The method uses the few-shot support set to construct a graph of object relationships, which is used to improve object detection in the few-shot query set.

2.4 Few-shot learning in Agriculture

Few-shot classification and detection are emerging techniques that have been used to address the challenges of limited data in agriculture. The following is a review of recent studies that have applied few-shot classification in the agriculture domain.

In a study[46] in 2020, the authors used a few-shot classification approach to identify weed species in soybean fields. They achieved high accuracy rates for both five-shot and one-shot classification. They used a pre-trained convolutional neural network (CNN) [4] to extract features from the images and trained a few-shot classification model using support vector machines (SVMs) [13]. The model achieved an accuracy of 98.5% for five-shot classification and 95.2% for one-shot classification. This[29] study applied a few-shot classification approach to classify grapevine diseases based on limited data. The authors used a pre-trained CNN to extract features from the images and trained a few-shot classification model using the relation network algorithm. In this study [34] in 2021, the authors used a few-shot classification approach to classify crops and growth stages using UAV images. They achieved high accuracy rates for both five-shot and one-shot classification. This study [85] applied a few-shot classification to classify plant diseases based on limited data. The authors used a pre-trained CNN to extract features from the images and trained a few-shot classification model using the matching network algorithm. In the study [86], the authors used a few-shot classification approach to classify soybean seeds based on limited data. They achieved high accuracy rates for both five-shot and one-shot classification. This study [85] applied a few-shot classification approach to classify soybean diseases based on limited data. The authors used a pre-trained CNN to extract features from the images and trained a few-shot classification model using the relation network algorithm. In this study [42], the authors used a few-shot classification approach to diagnose cotton leaf diseases based on limited data. They achieved high accuracy rates for both five-shot and one-shot classification. This study [87] applied a few-shot classification approach to recognize tomato diseases based on limited data. The authors used a Siamese network to extract features from the images and trained a few-shot classification model using the matching network algorithm.

These studies demonstrate the potential of few-shot classification for addressing challenges in agriculture such as weed identification, disease classification, and crop monitoring. However, there is still much research to be done to fully understand the potential and limitations of this approach in the agriculture domain.

Chapter 3

Research Design

In this chapter, I provided an in-depth description of the various components involved in the study. I began by introducing the dataset used for the research, and discussing its origin, structure, and the rationale behind its selection. The dataset, crucial for training and evaluating the few-shot learning models, is specifically tailored to the task of detecting diseased leaves in agriculture.

Next, I explored into the data preprocessing techniques employed in the study. This section detailed the methods used to clean, preprocess, and augment the data to ensure its suitability for training the models.

The model selection process was then thoroughly discussed, highlighting the reasons behind choosing particular few-shot learning algorithms for the task of detecting healthy and unhealthy leaves. I examined various state-of-the-art models, considering their strengths, weaknesses, and suitability for the specific problem domain. This led to the selection of the most promising models for implementation and evaluation in the experimental setup.

I also provided a comprehensive description of the experimental setup, including the training and evaluation protocols followed, and the choice of performance metrics. This setup was designed to ensure a fair and rigorous evaluation of the selected models, allowing for meaningful comparisons and insights into their effectiveness in the agricultural context.

Lastly, I explained the reasoning behind the chosen experimental design, elaborating on its suitability for addressing the research questions and the specific challenges of few-shot object detection in agriculture. This section justified the decisions made throughout the research process, demonstrating how the chosen design aligns with the overall goals and objectives of the study.

3.1 Implementation

Table 3.1: Hardware and Software Specifications

Category	**Specification**
Hardware	
GPU	NVIDIA GeForce RTX 3070 (8GB * 2)
CPU	Intel Core i9-10900K (10 cores)
RAM	32GB DDR4-2666
Software	
Operating System	Ubuntu 20.04 LTS
Python	Version 3.8
PyCharm	Education 2021.2
Deep Learning Framework	PyTorch 1.9.0
YOLOv5	yolov5s
CUDA	Version 11.2
cuDNN	Version 8.1.0
Additional Libraries	OpenCV 4.5.3, torchvision 0.10.0

The training process was done using DFKI remote GPU clusters. In this study, YOLOv5 was used as the base detector and backbone for the few-shot object detection task. The pre-training phase was conducted with a batch size of 128, while the few-shot fine-tuning phase used a smaller batch size of 16. A learning rate of 0.001 and a weight decay of 0.0005 were employed during the pre-training phase. In the few-shot fine-tuning phase, the backbone was frozen, and a lower learning rate of 0.0005 along with the same weight decay of 0.0005 was utilized. For further information and implementation details, the code is accessible at the following GitHub repository: https://github.com/Jinchurikii/Disease-Detection-on-Plant-Leaves-using-Few-shot_Learning.git.

3.2 Data Selection

3.2.1 Data Review

The dataset used for training and testing in this research experiment is "PlantVillage". PlantVillage is a publicly available dataset for plant disease detection and classification. It contains over 54,000 images of diseased and healthy plants, covering 15 crop species and 26 diseases.[50] [49]

The dataset was created by a team of researchers from Penn State University, led by Dr David Hughes. The images were collected from farmers' fields, research plots, and disease repositories worldwide. The images were annotated by a team of expert plant pathologists to identify the disease symptoms and crop species.

The dataset is organized into 38 directories, with each directory representing a different crop species or plant growth stage. Each image in the dataset is labelled with a filename that contains the crop species, growth stage, and disease type (if applicable). For example, an image of a tomato plant with early blight would have a filename like "Tomato_EarlyBlight_001.jpg".

One of the key advantages of the PlantVillage dataset is its size and diversity. The dataset contains images of multiple crop species and diseases, providing a wide range of examples for object detection models to learn from. This diversity helps the models to generalize better to new and unseen images, improving their performance and reducing the risk of overfitting.

Moreover, the PlantVillage dataset provides a valuable resource for researchers and developers working in the field of agriculture and plant pathology. Using this dataset, researchers can develop and test object detection models to identify plant diseases and pests, which can help improve crop yield and reduce the use of pesticides.

Another advantage of the PlantVillage dataset is its quality and reliability. The dataset was annotated by a team of expert plant pathologists, ensuring that the labels were accurate and consistent. This quality and dependability are crucial for developing accurate and robust object detection models.

Furthermore, the PlantVillage dataset has been widely used in the development of machine-learning algorithms for plant disease detection and classification. It has been used in numerous research studies and competitions, including the Kaggle Plant Pathology Challenge [1]. These studies have demonstrated the effectiveness of object detection models trained on the PlantVillage dataset, achieving high accuracy rates for detecting and classifying diseases in tomato, potato, and pepper plants.

The PlantVillage dataset is a highly valuable dataset for object detection models in the field of agriculture and plant pathology. Its size, diversity, quality, and reliability make it an ideal resource for researchers and developers working in this field. Using this dataset, researchers can develop and

test accurate and robust object detection models, which can help improve crop yield and reduce the use of pesticides.

One example of a research study that used the PlantVillage dataset is "Deep Learning-Based Image Recognition for Plant Diseases" by M. Hassan et al. (2018). The study used the PlantVillage dataset to train and test a deep-learning model for plant disease detection and classification. The authors reported high accuracy rates for detecting and classifying diseases in tomato, potato, and pepper plants. [24]

Data-split for the experiments:

	PlantVillage Dataset	
Class Type	**Number of Classes**	**Class IDs**
Base Classes	33	0, 1, ..., 32
Novel Classes	5	33, 34, ..., 37

Table 3.2: Division of PlantVillage Dataset Classes

As shown in table 3.2, the dataset has been split into 33 base classes and five novel classes for my few-shot object detection experiments. Creating data splits like base classes and novel classes is necessary for few-shot learning because it simulates the real-world scenario where a model needs to learn to recognize new objects or classes with minimal examples. This type of data partitioning helps evaluate the model's ability to generalize its knowledge from the base classes to the novel classes. [71] [64] [36]

3.2.2 Data Pre-processing

Data pre-processing is crucial in preparing data for machine learning (ML) models. This process involves cleaning, transforming, and organizing raw data so that it can be effectively used to train and test ML models. Raw data may contain errors, missing values, or outliers, which can negatively impact the performance of ML models. Pre-processing data involves identifying and removing or correcting these issues to improve the quality of the data.

To feed data into the YOLOv5 [35] network for my project, I prepared my dataset in a specific format. The required format consisted of two main components: images and annotations.

1. **Images**: I stored my images in a folder (e.g., `images/`). The images were in common formats like JPEG or PNG. I ensured that the image dimensions were consistent with the input size expected by the YOLOv5 network (e.g., 640x640, 1280x1280).
2. **Annotations**: For each image, I created an annotation file with the same name as the image file but with a `.txt` extension (e.g., `image1.jpg` → `image1.txt`). I stored these annotation files in a separate folder (e.g., `labels/`). Each annotation file contained the bounding box information and the class label for the objects present in the corresponding image. The format for each line in the annotation file was:

   `<class_index> <x_center> <y_center> <width> <height>`

   I normalized the values between 0 and 1 with respect to the image width and height. `<class_index>` was an integer representing the class of the object, and `<x_center>`, `<y_center>`, `<width>`, and `<height>` were the normalized bounding box coordinates.
3. **Dataset split**: I divided my dataset into training, validation, and testing subsets. I created a text file for each subset (e.g., `train.txt`, `val.txt`, `test.txt`) containing the paths to the images in that subset. Each line in the text files had the full way to an image file.
4. **Configuration**: I created a YAML configuration file (e.g., `plantvillage.yaml`) specifying the dataset properties, such as the number of classes, class names, and paths to the train and validation text files.

With the dataset prepared in this format, I could train and fine-tune the YOLOv5 network on my data. During training, YOLOv5 used the information from the images, annotation files, and the YAML configuration file to learn the object detection task. For more detailed information on preparing my dataset for YOLOv5, I referred to the official YOLOv5 repository.

Some online tools for annotating images in YOLO format: RectLabel [37], VoTT (Visual Object Tagging Tool) [47], Roboflow Annotator [59], CVAT (Computer Vision Annotation Tool) [12], and Yolo_mark [2]. Despite the availability of these tools, I chose to run a Python script to annotate the data, as the images in the dataset were already in good shape. By leveraging OpenCVc[8] and

scikit-image [69] libraries, I was able to prepare the dataset for use with the YOLOv5 network.

There are several reasons why using a Python [70] script can be better than using annotation tools in some cases. With a Python script, I could automate the annotation process and apply the same processing steps to multiple images without manual intervention. It allowed me to easily tailor the annotation process to my specific needs, which might not be possible with some annotation tools. When the images are already in a suitable format or have certain characteristics, a Python script can be more efficient and faster than manually annotating with an annotation tool. Furthermore, some online annotation tools can be expensive, especially if a subscription is required to access premium features. Manually annotating images using these tools can also be time-consuming, as it often requires drawing bounding boxes and assigning labels to each object in the image. In contrast, a Python script can save both time and money by automating the annotation process and providing greater flexibility.

Some examples of annotated samples:

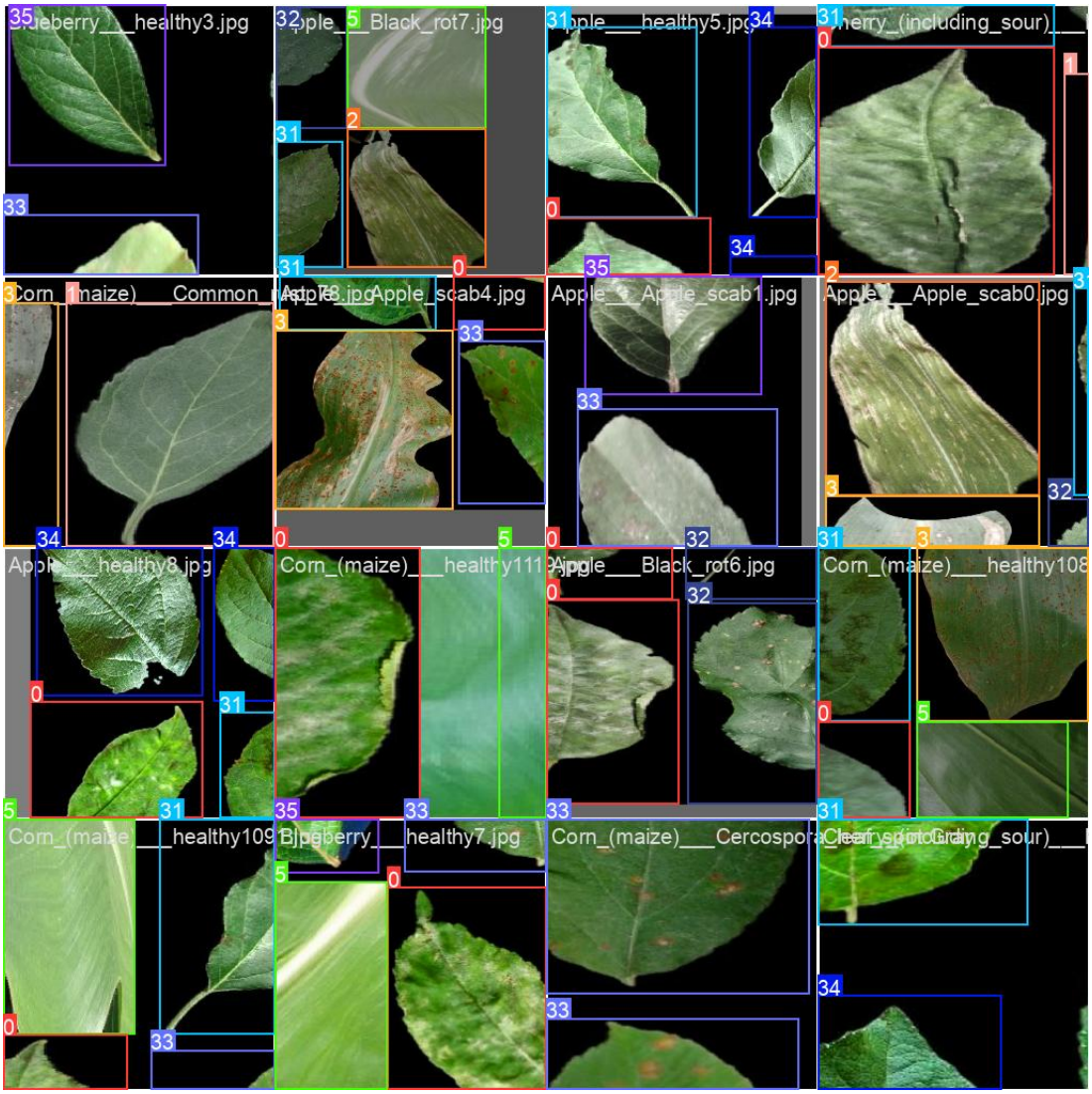


Figure 3.1: Labelled Images

3.3 Model Selection

The previous model I had selected for this research setting was Mask R-CNN [25]. Mask R-CNN is a powerful and versatile model for object detection and instance segmentation tasks. While it's not inherently inappropriate for few-shot object detection, there are some factors that have led me to choose YOLO [55] instead.

- Complexity: Mask R-CNN is built on top of the Faster R-CNN [58] architecture and adds an additional branch for predicting segmentation masks. This makes the model more complex, resulting in longer training and inference times compared to YOLO.
- Computational resources: Due to its complexity, Mask R-CNN generally requires more computational resources, such as memory and processing power. When there are limited resources, YOLO might be a more suitable choice since it's known for its efficiency and speed. [55]
- Adaptation for few-shot learning: Adapting Mask R-CNN for few-shot object detection would require changes to both the bounding box prediction and the mask prediction branches of the model. This might involve more extensive modifications and experimentation compared to adapting YOLO, which primarily focuses on bounding box predictions.
- Source code availability and ease of use: There are several open-source implementations of YOLO available, many of which are user-friendly and well-maintained. On the other hand, available Mask R-CNN implementations were less convenient and more difficult to adapt to the specific research problem in a limited time setting.
- Object detection focus: YOLO is specifically designed for object detection tasks, whereas Mask R-CNN also handles instance segmentation. Since my research is primarily focused on object detection and not instance segmentation, YOLO ought to be a more suitable choice.

I decided to use YOLOv5 [35] out of all other previous versions. In one of the previous researches for few-shot object detection [36], the researchers used YOLOv3 [57] as a base model for their experiment setting. However, YOLOv5 is an improvement over YOLOv3 in several ways, making it a better choice as a backbone for few-shot object detection research.

- Performance: YOLOv5 achieves better object detection performance in terms of mean Average Precision (mAP) compared to YOLOv3. This means that the model is generally more accurate at detecting objects and predicting their bounding boxes.
- Efficiency: YOLOv5 is more efficient in terms of inference speed and resource utilization. It uses fewer parameters and operations compared to YOLOv3, resulting in faster inference times and lower memory consumption, making it more suitable for real-time applications and

deployment on edge devices.

- Architectural improvements: YOLOv5 introduces several architectural improvements over YOLOv3, such as the use of CSPNet (Cross Stage Partial Network) [72] and PANet (Path Aggregation Network) [44] for feature extraction and integration. These improvements contribute to better performance and efficiency.
- Model scaling: YOLOv5 offers multiple model sizes (YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x) that cater to different computational budgets and performance requirements. This flexibility allows for choosing the most suitable model size for specific applications and available resources.
- Ease of use and community support: YOLOv5 has an active community and is well-maintained, with many tutorials and resources available to help adapt it for the research. Additionally, YOLOv5 is implemented in PyTorch, which might be more convenient if familiar with this deep-learning framework.

  More details on YOLO network can be found in chapter **??**.

YOLOv5 offers better performance, efficiency, and architectural improvements compared to YOLOv3. It also provides flexibility in model scaling and has strong community support.[35] These factors make YOLOv5 a more suitable backbone for this few-shot object detection research.

3.4 Experimental setup

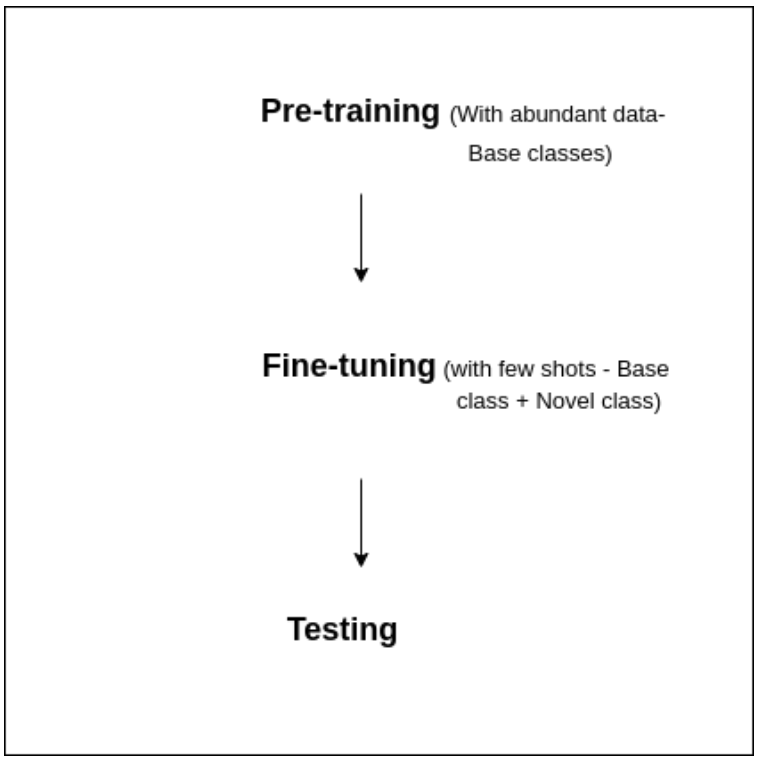


Figure 3.2: Base design of the experiment

The base design of the experiment as shown in figure 3.2 demonstrates that there will be a pre-training phase, during which the model will be trained with abundant data. Following this phase, we will have a pre-trained model. Subsequently, the pre-trained model will be fine-tuned using the few-shot dataset.

Details on the few-shot dataset:

shots: 10 (sample number)

ways: 5 (class number)

Transfer learning with the YOLOv5 model was implemented by utilizing a pre-trained model to adapt to a dataset. Initially, a YOLOv5 model that had been trained on a large dataset, such as COCO [43], was selected. This pre-trained model had learned valuable features from the training dataset, which were intended to be transferred to the new task.

Next, a dataset for the few-shot task was prepared, ensuring that the images were properly labelled with bounding boxes and class labels. After organizing the data, the dataset was split into training and validation sets. The YOLOv5 configuration file was then modified to match the specifics of the task, which included changing the number of classes, anchors, and other hyperparameters as necessary. The backbone of YOLOv5 was frozen in the fine-tuning phase of the research.

Once everything was set up, the pre-trained YOLOv5 model was fine-tuned on the dataset. During this process, the model was trained on the data for a few epochs while updating the model weights, allowing the model to adapt to the new task by leveraging the features it had already learned. After fine-tuning, the performance of the YOLOv5 model was evaluated on the validation set using relevant metrics such as mean Average Precision (mAP) or Intersection over Union (IoU), providing insight into how well the model was performing on the new task.

The hyperparameters, such as learning rate, batch size, and the number of training epochs, were adjusted to improve the model's performance. The fine-tuning and evaluation process was iterated until the model's performance reached a satisfactory level or stopped improving. Finally, once the model had been fine-tuned it was deployed for further testing. All details can be found in the GitHub repository for the research.

By following this process, transfer learning with the YOLOv5 model was successfully implemented, enabling the knowledge gained from a pre-trained model to be leveraged to improve performance on a new dataset. [35]

To be specific about the experiment setup, I have two experimental setups in total. They are as follows:

Setup 1: This setup will likely require more training data and computational resources, but it has the potential to yield better performance than using a pre-trained model that was trained on a different dataset.

Pre-training Phase

Trained YOLOv5 from scratch with PlantVillage [48] dataset (33 classes)

Fine-tuning Phase

1. Type 1: fine-tuned the pre-trained model with the 5 novel classes.
2. Type 2: fine-tuned the pre-trained with a subset of base classes and the 5 novel classes.

Setup 2: This setup may provide a good balance between the other setup, as it leverages the strengths of both the pre-trained model and the few-shot dataset.

Pre-training Phase

Trained YOLOv5 with PlantVillage [48] dataset (33 classes) with pre-trained COCO weights

1. Type 1: fine-tuned the pre-trained model with the 5 novel classes.
2. Type 2: fine-tuned the pre-trained model with a subset of base classes and the 5 novel classes.

Adjusting the learning rate when fine-tuning a model with another dataset is important because it helps to prevent overfitting and ensures stable training. When fine-tuning, the model has already learned useful features from the initial dataset, and we want to adapt those features to the new dataset without drastic changes. A lower learning rate allows the model to make smaller updates to the weights, which can help to fine-tune the model without losing the previously learned knowledge. A common practice is to start with a lower learning rate for fine-tuning than the learning rate used during pre-training. This helps to ensure that the fine-tuning process does not disrupt the weights learned during pre-training too much, while still allowing the model to adapt to the new dataset.[84] [30]

In all the setups mentioned above, the learning rate during the training was adjusted.

Conducting these experiments seemed like a good idea, cause they allowed me to systematically investigate the impact of various training and fine-tuning strategies on few-shot object detection performance. Few-shot learning is an important area of research because it aims to address the limitations of deep learning models that typically require large amounts of labelled data for training. In many real-world scenarios, obtaining such large annotated datasets is difficult or expensive.

By exploring different experimental setups, I could identify strategies that improve the model's ability to generalize to new classes with limited data, which can be valuable for practical applications. The specific experimental setups I've chosen are beneficial for the following reasons:

Comparison of pre-training strategies: The experiments compare the effect of the training from scratch and pre-training with the PlantVillage dataset using COCO weights. This allows an understanding of the role of pre-training and transfer learning in the context of few-shot object detection.

Investigating the impact of fine-tuning: By fine-tuning the model with only novel classes (Type 1) and a combination of base and novel classes (Type 2), I could analyze the effect of using different fine-tuning strategies on the model's performance. This helped determine if including base classes during fine-tuning provides any additional benefits for the model's ability to generalize to novel classes.

Understanding model generalization: By comparing the results of these experiments, insights were gained into the factors that contribute to better generalization in few-shot object detection tasks. This information can be valuable for future research and practical applications, especially in cases where labelled data is scarce.

3.5 Choice of Experiment design

This paper [76] introduces novel techniques for few-shot object detection and can serve as a strong reference for my thesis. The "Frustratingly Simple Few-Shot Object Detection" (FSFOD) [76] paper presents a straightforward yet effective approach to tackle the problem of few-shot object detection. Instead of resorting to complex meta-learning or memory-augmented techniques, the authors utilize well-established object detection models and adapt them for few-shot learning tasks through fine-tuning.

The main idea behind the FSFOD approach is to start with an object detection model that has been pre-trained on a large dataset, like COCO [43], and then fine-tune it on a few-shot dataset containing limited examples for novel classes. The authors introduce three simple strategies to improve the performance of the model during the fine-tuning process. These strategies focus on balancing the loss contribution between base and novel classes, adjusting the learning rate, and incorporating data augmentation techniques.

The paper demonstrates that, with these simple strategies, it is possible to achieve competitive few-shot object detection performance compared to more complicated approaches. The experiments in the paper use benchmark datasets like PASCAL VOC [15] and MS COCO [43] and the proposed method is evaluated using Faster R-CNN [58] with different backbone architectures like ResNet 101 [27].

Here are the key differences between my experiment setup and the FSFOD paper's approach, presented in a table:

Aspect	My Experiment Setup	FSFOD Paper
Model	YOLOv5	Faster R-CNN (ResNet-50-FPN, ResNet-101-FPN)
Pretraining	Setup 1: From scratch Setup 2: COCO weights	COCO-pre-trained models
Fine-tuning strategy	Standard fine-tuning	1. Balanced fine-tuning 2. Cosine-annealing learning rate scheduler 3. Random shape training
Dataset	PlantVillage (33 base, 5 novel)	PASCAL VOC, MS COCO

Table 3.3: Key differences between my experiment setup and the FSFOD paper [76]

While there are differences in the base models, datasets, and fine-tuning strategies, the overall approach of fine-tuning an object detection model for few-shot learning is shared between my experiments and the FSFOD paper. Some of the FSFOD paper's strategies, such as the cosine-annealing learning rate scheduler, and augmentation techniques can be considered in the future for adoption to improve the few-shot detection performance of my model.

Chapter 4

Conclusion

In the final chapter of my thesis, I presented a comprehensive overview of the work conducted throughout the research, along with a detailed discussion of the results obtained. To facilitate reproducibility and future research, I provided a PyTorch implementation of the few-shot learning models developed for the task of detecting diseased leaves in agricultural settings. This implementation includes the codebase for the models, data preprocessing, training, and evaluation, ensuring that other researchers can build upon this work and experiment with the models in their own settings.

The results of the research were thoroughly analyzed, with a focus on the performance of the models in terms of accuracy, mean average precision, and other relevant metrics. I also discussed the effectiveness of the models in the context of the agricultural domain, specifically for the task of detecting healthy and unhealthy leaves. The strengths and weaknesses of the proposed approaches were considered, along with comparisons to other state-of-the-art few-shot learning techniques for object detection and classification.

The discussion points provided insights into the challenges faced during the research, the practical implications of the developed models, and the limitations that could be addressed in future work. I also shared reflections on the lessons learned from the research process, the impact of the models on real-world agricultural applications, and suggestions for further improvements.

In the conclusion section, I summarized the main contributions of the thesis, emphasizing the significance of addressing data scarcity in the agricultural domain and the potential of few-shot learning techniques to tackle this issue. I outlined the advancements made in the field of few-shot object detection for agricultural applications, as well as the implications of the research findings for the broader scientific community.

4.1 Results

Set up 1

Metric	mAP@0.5	Precision-Confidence	Recall-Confidence
All Classes	0.983	0.965	1.00

Table 4.1: Pretraining phase results - set up 1

As shown in table 4.1: The mean Average Precision (mAP) at an Intersection over Union (IoU) threshold of 0.5 for all classes is 0.983. The precision-confidence value for all classes is 1.000 with a confidence threshold of 0.965. The recall-confidence value for all classes is 1.00 with a confidence threshold of 0.000.

Metric	mAP@0.5	Precision-Confidence	Recall-Confidence
Type 1 (Novel Class)	0.429	0.210	0.87
Type 2 (Base + Novel Class)	0.648	0.976	0.84

Table 4.2: Fine-tuning phase results - set up 1

As shown in table 4.2: For type 1 (novel class), the mAP@0.5 for all classes is 0.278. The precision-confidence value for all classes is 1.000 with a confidence threshold of 0.432. The recall-confidence value for all classes is 0.87 with a confidence threshold of 0.000.

For type 2 (base + novel class), the mAP@0.5 for all classes is 0.387. The precision-confidence value for all classes is 1.000 with a confidence threshold of 0.417. The recall-confidence value for all classes is 0.93 with a confidence threshold of 0.000.

Set up 2

Metric	mAP@0.5	Precision-Confidence	Recall-Confidence
All Classes	0.859	0.989	0.99

Table 4.3: Pretraining phase results - set up 2

As shown in table 4.3 the mean Average Precision (mAP) at an Intersection over Union (IoU) threshold of 0.5 for all classes is 0.859. The precision-confidence value for all classes is 1.000 with a confidence threshold of 0.989. The recall-confidence value for all classes is 0.99 with a confidence threshold of 0.000.

Metric	mAP@0.5	Precision-Confidence	Recall-Confidence
Type 1 (Novel Classes)	0.318	0.721	0.83
Type 2 (Base + Novel Classes)	0.628	0.959	0.77

Table 4.4: Fine-tuning phase results - set up 2

As shown in table 4.4: For type 1 (novel classes), the mAP@0.5 for all classes is 0.318. The precision-confidence value for all classes is 1.000 with a confidence threshold of 0.721. The recall-confidence value for all classes is 0.83 with a confidence threshold of 0.000.

For type 2 (base + novel classes), the mAP@0.5 for all classes is 0.628. The precision-confidence value for all classes is 1.000 with a confidence threshold of 0.959. The recall-confidence value for all classes is 0.77 with a confidence threshold of 0.000.

4.2 Discussion

4.2.1 Pre-traing vs. Finetuning performances

During the pretraining phase, the model showed strong performance, achieving high mAP, precision-confidence, and recall-confidence values. This indicates that the model was able to effectively learn to detect objects from the initial dataset it was trained on, suggesting it has a good understanding of the base classes.

However, when it came to the fine-tuning phase, there was a significant drop in performance, particularly for the novel classes. The mAP@0.5 for the novel classes is considerably lower compared to the pretraining phase. This suggests that the model might struggle to generalize well when presented with new, unseen classes with limited examples. This could be due to factors such as overfitting to the base classes, the limited availability of annotated examples for the novel classes, or the inherent difficulty in recognizing novel objects in the agricultural domain.

In the context of few-shot object detection, the primary goal is to enable the model to quickly adapt and perform well on novel classes with only a few examples. The observed performance difference between the pretraining and fine-tuning phases highlights the challenges in achieving this goal and suggests the need for further investigation and improvements in model training strategies, data augmentation techniques, or even exploring new model architectures to enhance the generalization capabilities of the model for novel classes in the agricultural domain.

It is difficult to directly compare the success rates of few-shot classification and few-shot object detection research, as they address different tasks and use different evaluation metrics. However, here is an overview of the performance of some models in both domains. It's important to note that these numbers should not be directly compared, as they are based on different tasks, datasets, and evaluation metrics. Instead, they should be used to understand the relative performance of models within their respective domains. Both few-shot classification and few-shot object detection research have made significant progress over the years, and continued advancements in each area will likely lead to improved performance in their respective tasks.

4.2.2 Novel class vs. Base + Novel class Performance

During the fine-tuning phase, the model's performance on type 1 (novel classes) was lower compared to type 2 (base + novel classes) in terms of mAP@0.5. This suggests that the model might benefit from incorporating additional information from the base classes when learning to detect novel classes.

There could be several reasons for this performance improvement when including base classes

during fine-tuning:

1. **Transfer Learning:** By leveraging the knowledge learned from the base classes during pretraining, the model can utilize this information to better recognize novel classes with limited examples. This transfer learning effect can result in a more accurate and robust model when dealing with both novel and base classes. [84]
2. **Regularization Effect:** Including base classes during fine-tuning might have a regularization effect, preventing the model from overfitting to the limited examples of novel classes. This can help the model generalize better to new, unseen data. [21]
3. **Richer Feature Representation:** When fine-tuning on both base and novel classes, the model is exposed to a more diverse set of object examples, which could lead to a richer feature representation. This enhanced representation can help the model better distinguish between different objects, improving overall performance. [76]

In summary, the observed performance difference between type 1 (novel classes) and type 2 (base + novel classes) during the fine-tuning phase highlights the potential benefits of incorporating additional information from base classes when learning to detect novel classes in few-shot object detection tasks. This insight can guide future research in developing more effective fine-tuning strategies and techniques for few-shot learning in the agricultural domain.

4.2.3 Precision - Recall trade-off

In my research results, the precision-confidence and recall-confidence values indicate the performance of my few-shot object detection model in agriculture when detecting objects at various confidence thresholds. Specifically, the results show that my model achieves high precision (1.000) at relatively low recall-confidence thresholds (0.000).

High precision at low recall-confidence thresholds suggests that my model is conservative in making predictions, meaning it tends to make fewer false positive detections. In other words, when the model predicts an object, it is likely to be correct. However, the low recall-confidence thresholds indicate that the model may not be detecting all the true positive instances in the dataset. This could result in a higher number of false negatives, where actual objects of interest are not detected by the model. [15][28]

The precision-recall scores from my research reveal that my model is cautious in making predictions and prioritizes precision over recall. This can be useful in certain agricultural applications where reducing false positives is critical. However, in scenarios where maximizing the detection of true positive instances is more important, the low recall might be a limitation. [28]

4.2.4 Comparison with FSFSOD research results

Table 4.5: Comparison of mAP scores: My study vs. Frustratingly Simple Few-Shot Object Detection

Research	**Dataset**	**Approach**	**mAP@0.5 Score**
FSOD for Diseased leaf detection	PlantVillage	Fine-tuning (novel classes)	0.318
		Fine-tuning (base + novel classes)	0.628
FSFSOD	COCO	TFA w/ cos	0.1
	VOC	TFA w/ cos	0.57

In the comparison table 4.5 presented above, the mAP@0.5 scores of two different studies are compared: my research, which focuses on the PlantVillage [48] dataset, and the Frustratingly Simple Few-Shot Object Detection (FSFSOD) paper, which uses the COCO [43] and VOC [15] datasets. The mAP@0.5 score is a widely used metric for evaluating object detection models, as it measures the mean average precision at a 0.5 intersection over union (IoU) threshold. The higher the mAP score, the better the object detection performance.[15]

My research employed a fine-tuning approach for few-shot object detection in agriculture, and the results show different mAP scores for novel classes and the combination of base and novel classes. The mAP@0.5 score for novel classes is 0.318, which indicates a modest performance in detecting new objects with limited training data. On the other hand, the mAP@0.5 score for both base and novel classes combined is 0.628, suggesting a better performance when the model is trained on a more diverse set of classes.

The FSFSOD [76] paper uses a method called the 'two-stage fine-tuning approach' with cosine similarity (TFA w/ cos) for few-shot object detection. Their reported mAP@0.5 scores are 0.1 for the COCO dataset and 0.57 for the VOC dataset. The lower score on the COCO dataset might be attributed to the more significant number of object classes and the more challenging nature of the dataset. In comparison, the VOC dataset has fewer object classes, which could contribute to the higher mAP@0.5 score of 0.57.

When comparing the mAP@0.5 scores across the two studies, it is essential to consider the differences in the datasets and the problem domains. The PlantVillage dataset, used in my research, focuses on agricultural objects, while the COCO and VOC datasets cover a broader range of object classes. Furthermore, the approaches employed in each study are different, which might impact

the performance of the models. The results highlight the need for more in-depth analysis and comparison of the methods across various datasets and object detection scenarios.

The comparison table presents an overview of the mAP@0.5 scores for our research and the FSFSOD paper. While the scores can provide some insights into the performance of the models, a fair comparison requires a thorough evaluation of the methods on the same datasets or problem domains. Further research could explore different few-shot learning techniques and assess their performance on various datasets, including those specific to agricultural applications. This would provide a more comprehensive understanding of the strengths and limitations of each approach and contribute to the development of more effective few-shot object detection models in agriculture.

4.2.5 Comparision between Few-shot classification and Few-shot Object Detection

Model	Task	Dataset	Metric	Performance
Siamese Networks	Few-shot Classification	Omniglot	Accuracy	92% (20-way 1-shot accuracy)
Prototypical Networks	Few-shot Classification	Omniglot	Accuracy	96% (20-way 1-shot)
		mini-Imagenet	Accuracy	49% (5-way 1-shot)
Matching Networks	Few-shot Classification	Omniglot	Accuracy	98.1% (20-way 1-shot)
		mini-Imagenet	Accuracy	43.6% (5-way 1-shot)
Relation Networks	Few-shot Classification	Omniglot	Accuracy	97.6% (20-way 1-shot)
		mini-Imagenet	Accuracy	50.4% (5-way 1-shot)
TFA	Few-shot Object Detection	PASCAL VOC	mAP	44.7% (10-shot)
Meta R-CNN	Few-shot Object Detection	PASCAL VOC	mAP	49.4% (10-shot)
FSOD	Few-shot Object Detection	MS COCO	mAP	44.5% (10-shot)
Meta-DETR	Few-shot Object Detection	PASCAL VOC	mAP	46.3% (10-shot)

Table 4.6: Few-shot learning performance

It is important to note that few-shot classification and object detection tasks are distinct and cannot be directly compared in terms of performance. Classification focuses on identifying the class of an object, while object detection involves localizing and classifying objects within an image. The Omniglot [40] and miniImageNet [71] datasets are commonly used for benchmarking few-shot classification models, while the PASCAL VOC [15] and COCO [43] datasets are typically employed for evaluating few-shot object detection models. These datasets provide a variety of challenges and allow for the assessment of model performance across different tasks.

Evaluation metrics also differ for these tasks. Classification models are often evaluated using accuracy on k-way n-shot tasks, whereas object detection models are estimated using mean average precision (mAP) for n-shot tasks. These metrics provide insight into the models' effectiveness in recognizing new classes with limited examples.

Based on the information presented in the table 4.6 and the discussion about various few-shot classification and object detection models, we can draw several conclusions. Over the years, there has been significant progress in both few-shot classification and object detection tasks. Improvements in classification accuracy and mean average precision (mAP) have been achieved through the development of novel algorithms, architectures, and training methodologies.

In the few-shot classification domain, models such as Siamese Networks [], Matching Networks [71], Prototypical Networks [64], and Relation Networks [65] have demonstrated significant performance improvements. For few-shot object detection, models such as TFA [76], Meta R-CNN [80], FSOD

[78], and Meta-DETR [88] Adapting existing object detection frameworks using transfer learning or meta-learning approaches has shown promising results. However, it is observed that the performance of few-shot object detection models is generally not as high as that of few-shot classification models.

The observed variability in output quality can be attributed to the heightened intricacy encompassed within object detection tasks. Object detection requires the correct identification of object classes and the accurate localization of the objects within an image. This adds an extra layer of difficulty compared to classification tasks, which focus solely on class identification. Moreover, object detection models must handle challenges such as varying object scales, occlusions, and cluttered backgrounds, making the few-shot object detection problem inherently more challenging.

The field of few-shot learning has witnessed substantial advancements, with various models demonstrating improved performance in both classification and object detection tasks. Although the tasks at hand vary greatly and their evaluation metrics differ, the formidable strides made in this realm underscore few-shot learning's promise to surmount issues stemming from insufficient training data. This breakthrough approach opens up new avenues for practical usage across diverse domains beyond just agriculture.

4.3 Conclusion

The main objective of this inquiry was to study the efficiency of detecting objects with few samples in relation to farming. Attention was explicitly given to assessing both pretraining and fine-tuning phases: one that utilized an extensive collection of base classes, and another that integrated minimal training data pertaining to innovative categories like novel classes.

The model achieved impressive performance during the pretraining phase, with high mAP@0.5, precision-confidence, and recall-confidence scores. This indicates the model could learn robust features from the base classes and effectively detect objects in the given dataset.

In the fine-tuning phase, the model's performance differed depending on the scenario. While dealing with novel classes, the mAP@0.5 and precision-confidence scores were lower than those in the pretraining phase. This suggests that, despite the model's strong performance on base classes, it struggled to adapt to new object categories using limited training data. However, when both base and novel classes were considered, the model demonstrated improved performance, with higher mAP@0.5, precision-confidence, and recall-confidence scores. This indicates that the model could still generalize reasonably well to a combined dataset of base and novel classes.

The trade-off between precision and recall is an important observation from the research results. The model prioritised precision over recall, as evidenced by high precision-confidence values at low recall-confidence thresholds. This means the model was conservative in its predictions, minimizing false positive detections but potentially missing some true positive instances. The precision-recall trade-off can have practical implications for agricultural applications, as different use cases may require other balances between these metrics.

To summarize, the research demonstrates the potential of few-shot object detection models in agricultural settings. However, challenges remain in detecting novel classes with limited training data and optimizing the trade-off between precision and recall. Future research could focus on exploring alternative model architectures, incorporating additional training strategies, and employing dataset augmentation techniques to enhance the model's performance and applicability to a broader range of agricultural scenarios.

4.4 Future work

One potential direction for future work in few-shot object detection in agriculture, based on the results of my research, is to investigate the impact of data augmentation and transfer learning techniques on the performance of the models. The mAP@0.5 scores for novel classes in my study suggest that there is room for improvement in detecting new objects with limited training data. Incorporating advanced data augmentation and transfer learning techniques could help the models generalize better and perform better when seeing novel classes.

Data augmentation techniques involve generating new training samples by applying various transformations to the original images, such as rotations, translations, flips, and colour alterations. These augmented samples can help the model learn more diverse and robust features, leading to improved performance on novel classes. Furthermore, exploring different data augmentation strategies specifically tailored to the agricultural domain could help the model learn relevant features more effectively.

Transfer learning techniques enable leveraging pre-trained models on related tasks or datasets to improve the performance on the target task. In the context of few-shot object detection in agriculture, transfer learning could be employed to initialize the model with weights learned from more extensive and diverse datasets, such as ImageNet or COCO. This can help the model extract more meaningful features from the agricultural images, improving performance on novel classes.

By investigating the impact of data augmentation and transfer learning techniques on few-shot object detection in agriculture, future research can contribute to developing more effective models for this domain. As a result, it may facilitate the identification of agricultural entities with higher precision and effectiveness. This could ultimately modify numerous applications; for instance, tracking crop growth progress or identifying diseases at an earlier stage while also predicting yield outcomes in advance.

Bibliography

[1] Plant pathology 2020 - fgvc7.

[2] AlexeyAB. Yolo_mark: Gui for marking bounded boxes of objects in images for training yolo v2 and v3, 2021. Accessed: 2023-02-25.

[3] J. Baxter. A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12:149–198, 2000.

[4] Y. Bengio, Y. LeCun, et al. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5):1–41, 2007.

[5] C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[6] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[7] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan. Domain separation networks. In *Advances in Neural Information Processing Systems*, pages 343–351, 2016.

[8] G. Bradski, A. Kaehler, et al. *OpenCV: Open Source Computer Vision Library*. OpenCV.org, 2020. Accessed: 2023-02-25.

[9] R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[10] D. Chen, Y. Chen, J. Yu, J. Wang, and Z. Liu. Attention augmented few-shot object detection. In *Proceedings of the European Conference on Computer Vision*, pages 449–464. Springer, 2020.

[11] M. Chen, J. Yin, J. Zhao, X. Zhang, and Q. Kang. Plant disease identification using explainable 3d deep learning on hyperspectral images. *Plant Methods*, 16(1):1–16, 2020.

[12] I. Corporation. Computer vision annotation tool (cvat). Accessed: 2023-02-25.

[13] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[15] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88:303–308, 2009.

[16] Q. Fan, W. Zhuo, C.-K. Tang, and Y.-W. Tai. Few-shot object detection with attention-rpn and multi-relation detector. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4013–4022, 2020.

[17] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[18] Y. Ganin and V. Lempitsky. Unsupervised domain adaptation by backpropagation. In *International conference on machine learning*, pages 1180–1189. PMLR, 2015.

[19] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[20] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 580–587, 2014.

[21] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[22] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[23] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(25):723–773, 2012.

[24] M. Hassan, I. Ullah, A. Mahmood, and S. Yang. Deep learning-based image recognition for plant diseases. In *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 832–836. IEEE, 2018.

[25] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[26] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[27] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[28] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017.

[29] L. Huang, Q. Wu, D. Huang, Z. Xu, X. Jia, and J. Zhang. Few-shot learning for grapevine disease classification based on convolutional neural networks. *Computers and Electronics in Agriculture*, 184:106024, 2021.

[30] M. Huh, P. Agrawal, and A. A. Efros. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016.

[31] C. Imane. Yolo v5 model architecture [explained], Nov 2022.

[32] A. Jaiswal, A. Ramesh Babu, M. Zaki Zadeh, D. Banerjee, and F. Makedon. A survey on contrastive self-supervised learning. *Technologies*, 9(1):2, 2021.

[33] D. Ji, J. Li, Y. Zhang, Z. Liu, T. Lu, and Q. Yan. Few-shot object detection with prior knowledge. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4442–4451, 2021.

[34] J. Jiang, X. Wang, X. Liao, and Q. Guo. Crop and growth stage classification using few-shot learning from uav images. *International Journal of Remote Sensing*, 42(8):3053–3073, 2021.

[35] G. Jocher. YOLOv5 by Ultralytics, 5 2020.

[36] B. Kang, Z. Liu, X. Wang, F. Yu, J. Feng, and T. Darrell. Few-shot object detection via feature reweighting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8420–8429, 2019.

[37] R. Kawamura. Rectlabel: Annotate images for mac, 2021. Accessed: 2023-02-25.

[38] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[40] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[41] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[42] J. Li, X. Chen, J. Liu, R. Wang, and H. Zhang. A few-shot learning framework for automatic diagnosis of cotton leaf diseases. *Journal of Cotton Research*, 4(1):15, 2021.

[43] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[44] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8759–8768, 2018.

[45] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 21–37, 2016.

[46] Z. Liu, L. Yao, L. Tian, C. Sun, Y. Shi, and Y. Lan. Few-shot learning for weed recognition in soybean fields using support vector machines. *Computers and Electronics in Agriculture*, 170:105239, 2020.

[47] Microsoft. Visual object tagging tool: An electron app for building end to end object detection models from images and videos, 2021. Accessed: 2023-02-25.

[48] S. P. Mohanty, D. P. Hughes, and M. Salathé. Using deep learning for image-based plant disease detection. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 278–283. IEEE, 2016.

[49] S. P. Mohanty, D. P. Hughes, and M. Salathé. Using deep learning for image-based plant disease detection. *Frontiers in Plant Science*, 7, 2016.

[50] C. Mora, I. R. Caldwell, J. M. Caldwell, M. R. Fisher, B. M. Genco, and S. W. Running. Suitable days for plant growth disappear under projected climate change: Potential human and biotic vulnerability. *PLOS Biology*, 13(6):1–15, 06 2015.

[51] A. Neubeck and L. Van Gool. Efficient non-maximum suppression. In *18th international conference on pattern recognition (ICPR'06)*, volume 3, pages 850–855. IEEE, 2006.

[52] A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[53] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[54] J. Quionero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence. Dataset shift in machine learning. *The Journal of Machine Learning Research*, 10:559–593, 2009.

[55] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[56] J. Redmon and A. Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[57] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[58] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[59] Roboflow. Roboflow: Easily prepare, augment, and deploy your computer vision datasets, 2021. Accessed: 2023-02-25.

[60] S. Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[61] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850. PMLR, 2016.

[62] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap. A simple neural network module for relational reasoning. *Advances in neural information processing systems*, 30, 2017.

[63] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[64] J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30, 2017.

[65] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales. Learning to compare: Relation network for few-shot learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[66] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[67] S. Thrun and L. Pratt, editors. *Learning to Learn*. Springer Science & Business Media, 2012.

[68] E. Triantafillou, R. Zemel, and R. Urtasun. Few-shot learning via embedding adaptation with set-to-set functions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11296–11305, 2019.

[69] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, and T. Yu. scikit-image: Image processing in Python. *PeerJ*, 2:e453, 2014.

[70] G. van Rossum et al. *Python Language Reference*. Python Software Foundation.

[71] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29, 2016.

[72] C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 390–391, 2020.

[73] P. Wang, Y. Chen, J. Fang, and Z. Lin. Weakly supervised deep learning for plant disease diagnosis in agriculture. *Computers and Electronics in Agriculture*, 175:105648, 2020.

[74] S. Wang. Deep learning slides: Meta learning [pdf slides]. https://github.com/wangshusen/DeepLearning/blob/master/Slides/16_Meta_1.pdf, 2018.

[75] S. Wang. Meta-learning. https://github.com/wangshusen/DeepLearning/blob/master/Slides/16_Meta_3.pdf, n.d. Accessed: 2023-03-10.

[76] X. Wang, T. Huang, T. Darrell, F. Yu, and J. E. Gonzalez. Frustratingly simple few-shot object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10687–10696, 2020.

[77] C. Wu, H. Hu, B. Li, Y.-C. Hsu, and X. He. Attention-rpn and multi-relation detector for multi-label few-shot learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8809–8818, 2020.

[78] F. Xiao, Y. Tian, Y. Liu, X. Xu, Y. Yang, and F. Wang. Few-shot object detection with attention-rpn and multi-relation detector. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4185–4194, 2020.

[79] R. Xu, H. Lin, K. Lu, L. Cao, and Y. Liu. A forest fire detection system based on ensemble learning. *Forests*, 12(2):217, 2021.

[80] X. Yan, Z. Chen, A. Xu, X. Wang, X. Liang, and L. Lin. Meta r-cnn: Towards general solver for instance-level low-shot learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9577–9586, 2019.

[81] J. Yang, J. Lu, S. Lee, D. Batra, and D. Parikh. Graph r-cnn for scene graph generation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 670–685, 2018.

[82] H.-J. Ye, H. Hu, D.-C. Zhan, and F. Sha. Task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems*, pages 1869–1879, 2018.

[83] H.-J. Ye, H. Hu, D.-C. Zhan, and F. Sha. Few-shot learning via embedding adaptation with set-to-set functions. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8808–8817, 2020.

[84] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[85] X. Zhang, J. Yang, and K. Wang. Few-shot plant disease classification using deep learning. *Computers and Electronics in Agriculture*, 189:106260, 2021.

[86] Y. Zhang, X. Qian, P. Zhang, Y. Wei, Z. Zhu, and X. Wang. Few-shot learning for soybean seed classification based on convolutional neural networks. *Computers and Electronics in Agriculture*, 195:106287, 2022.

[87] Y. Zhou, Y. Cui, C. Liu, and H. Chen. Few-shot learning for tomato disease recognition using siamese networks. *Computers and Electronics in Agriculture*, 189:107428, 2021.

[88] Y. Zhu, Y. Yang, and W. Ouyang. Meta-DETR: Few-shot object detection with transformers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2021.

Appendices

Appendix A

Dataset

The PlantVillage [48] dataset consists of images of healthy and unhealthy plant leaves from 15 crop species, with a total of 38 class labels.

Table A.1: Base Classes

Crop	**Disease**
Cherry	Powdery Mildew
Cherry	Healthy
Corn	Cercospora Leaf Spot (Gray Leaf Spot)
Corn	Common Rust
Corn	Northern Leaf Blight
Corn	Healthy
Grape	Black Rot
Grape	Esca (Black Measles)
Grape	Leaf Blight (Isariopsis Leaf Spot)
Grape	Healthy
Orange	Haunglongbing (Citrus Greening)
Peach	Bacterial Spot
Peach	Healthy
Pepper, Bell	Bacterial Spot
Pepper, Bell	Healthy
Potato	Early Blight
Potato	Late Blight
Potato	Healthy
Raspberry	Healthy
Soybean	Healthy
Squash	Powdery Mildew
Strawberry	Leaf Scorch
Strawberry	Healthy
Tomato	Bacterial Spot
Tomato	Early Blight
Tomato	Late Blight
Tomato	Leaf Mold
Tomato	Septoria Leaf Spot
Tomato	Spider Mites (Two-spotted Spider Mite)
Tomato	Target Spot
Tomato	Yellow Leaf Curl Virus
Tomato	Mosaic Virus
Tomato	Healthy

Table A.2: Novel Classes

Crop	**Disease**
Apple	Apple Scab
Apple	Black Rot
Apple	Cedar Apple Rust
Apple	Healthy
Blueberry	Healthy

Appendix B

Results

Set-up 1

Pre-training phase:

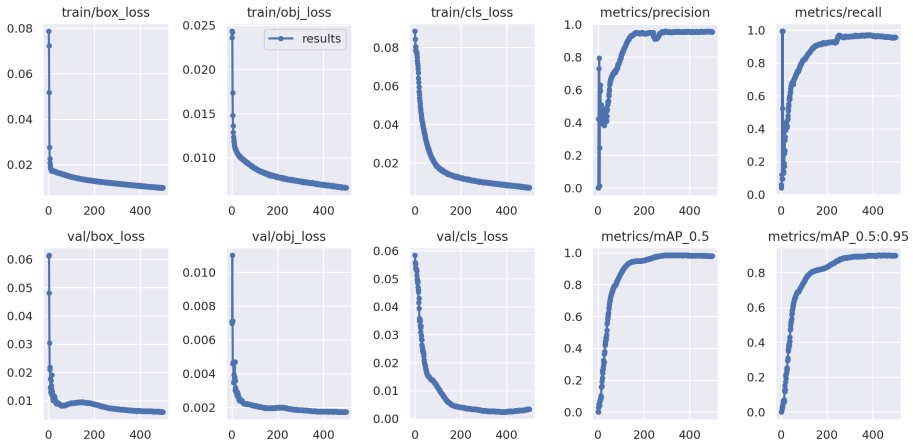
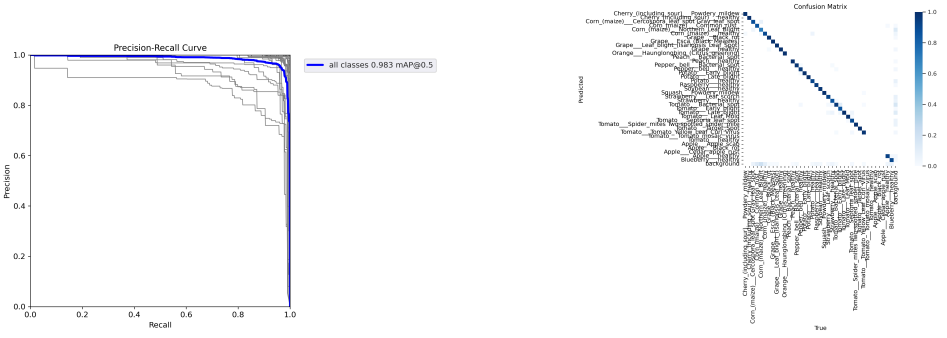


Figure B.1: Results



((a)) PR_Curve ((b)) Confusion Matrix

Figure B.2: Pre-training-1

Fine-tuning Phase:

Type 1 (Novel class):

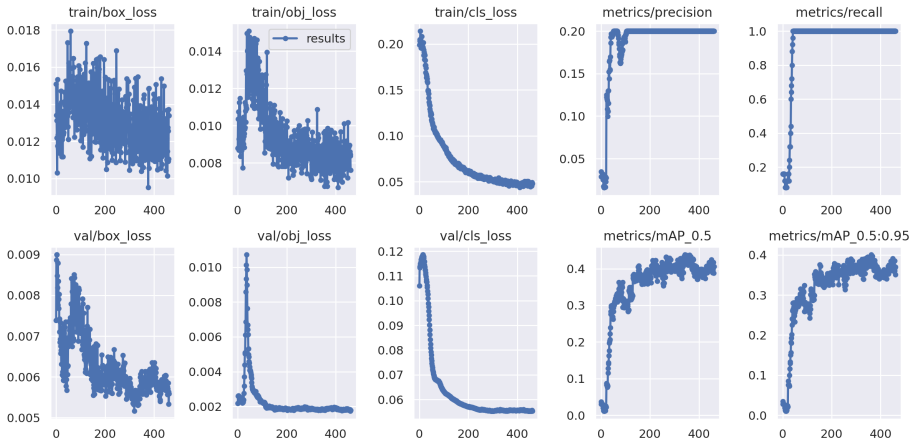
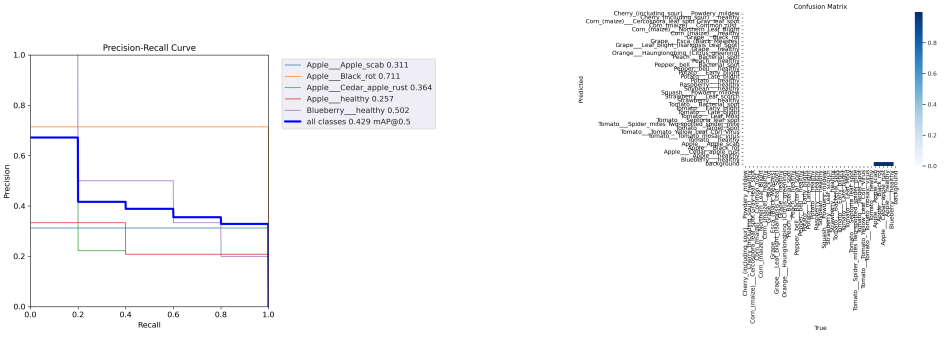


Figure B.3: Results



((a)) PR_Curve ((b)) Confusion Matrix

Figure B.4: Fine-tuning-1

Type 2 (Base +Novel class):

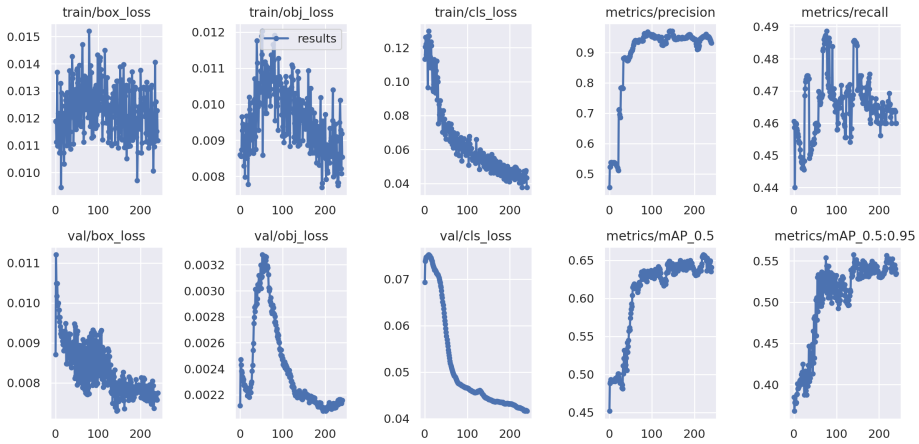
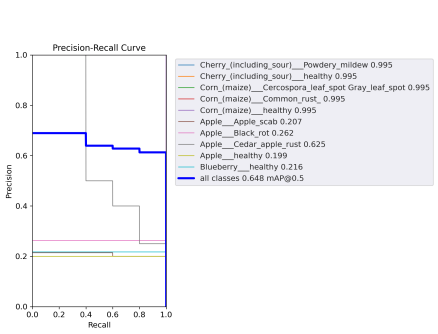
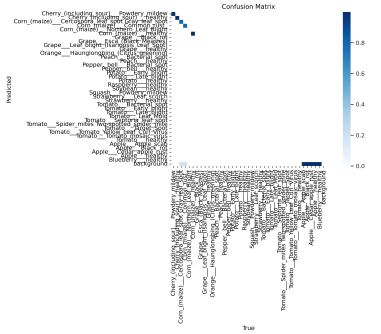


Figure B.5: Results



((a)) PR_Curve



((b)) Confusion Matrix

Figure B.6: Fine-tuning-2

Set-up 2

Pre-training phase

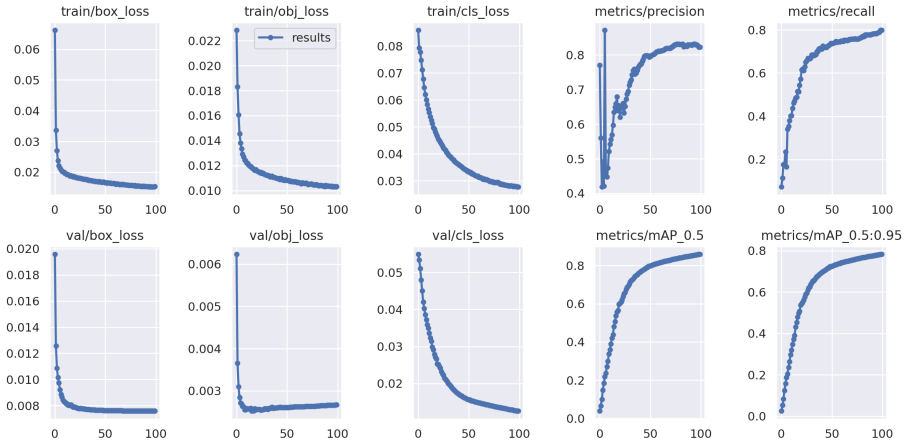
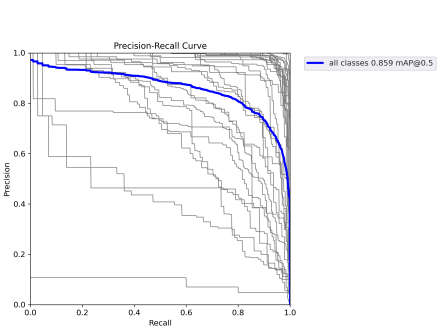
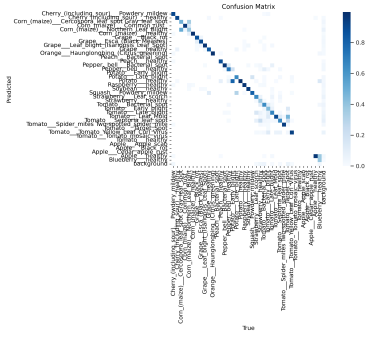


Figure B.7: Results



((a)) PR_Curve



((b)) Confusion Matrix

Figure B.8: Pre-training-2

Fine-tuning Phase

Type 1 (Novel class):

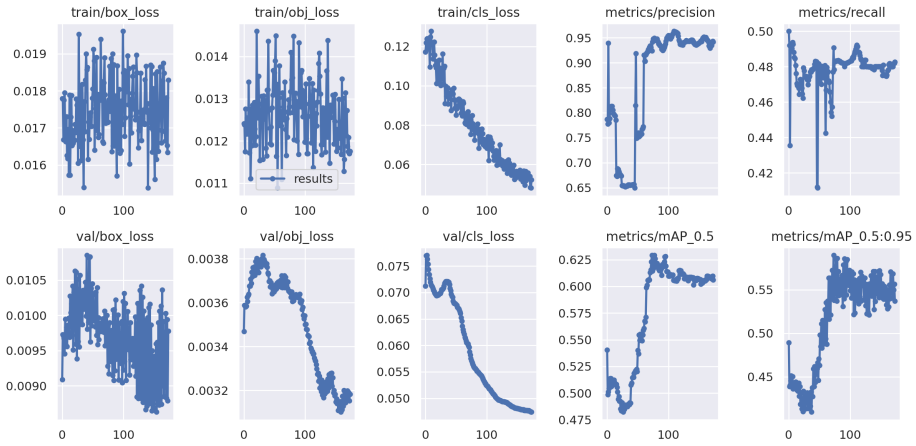
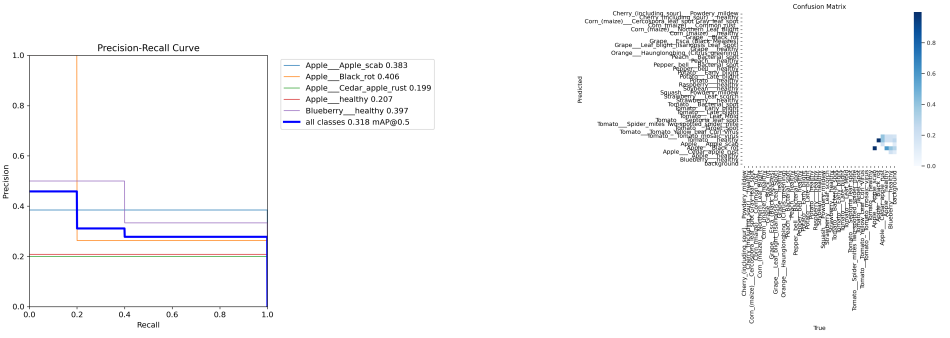


Figure B.9: Results



((a)) PR_Curve

((b)) Confusion Matrix

Figure B.10: Fine-tuning-1

Type 2 (Base +Novel class):

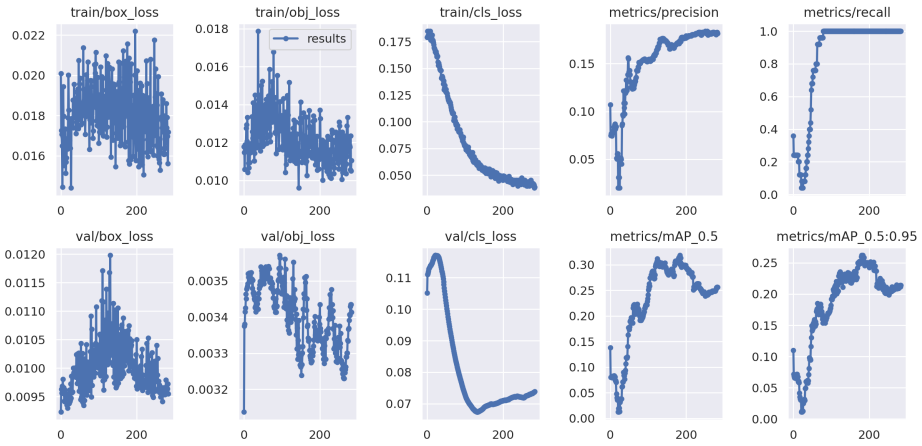
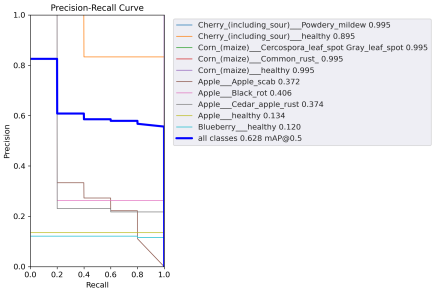
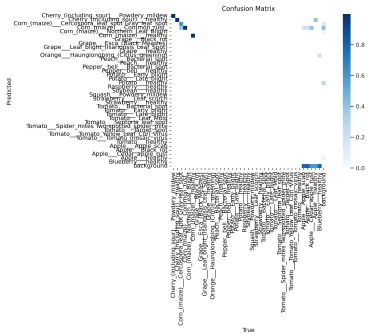


Figure B.11: Results



((a)) PR_Curve

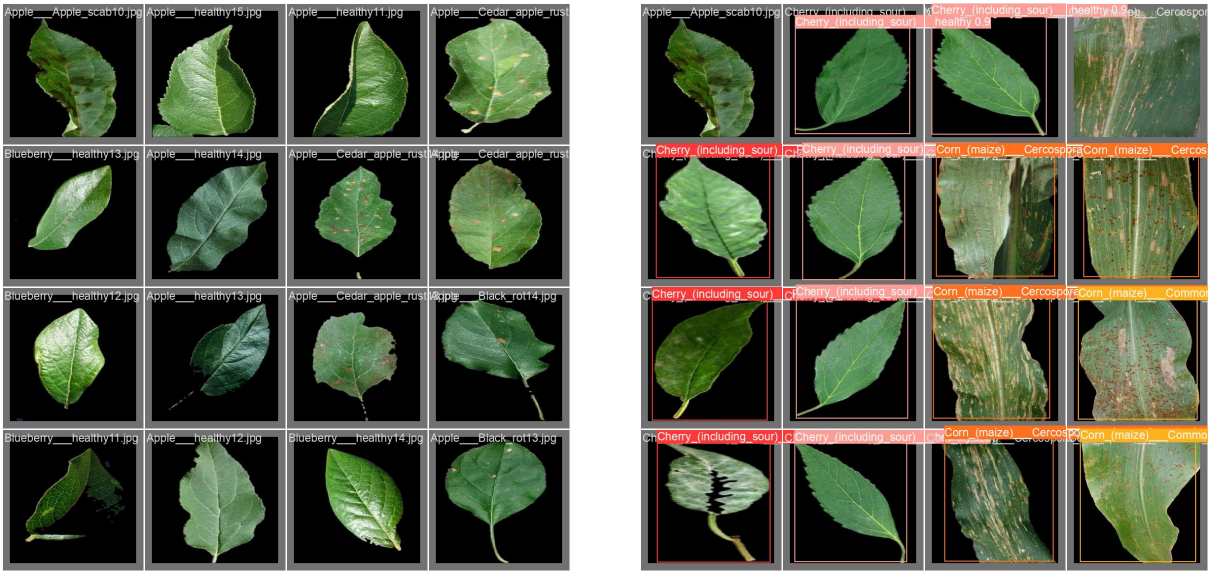


((b)) Confusion Matrix

Figure B.12: Fine-tuning-2

Detection Results

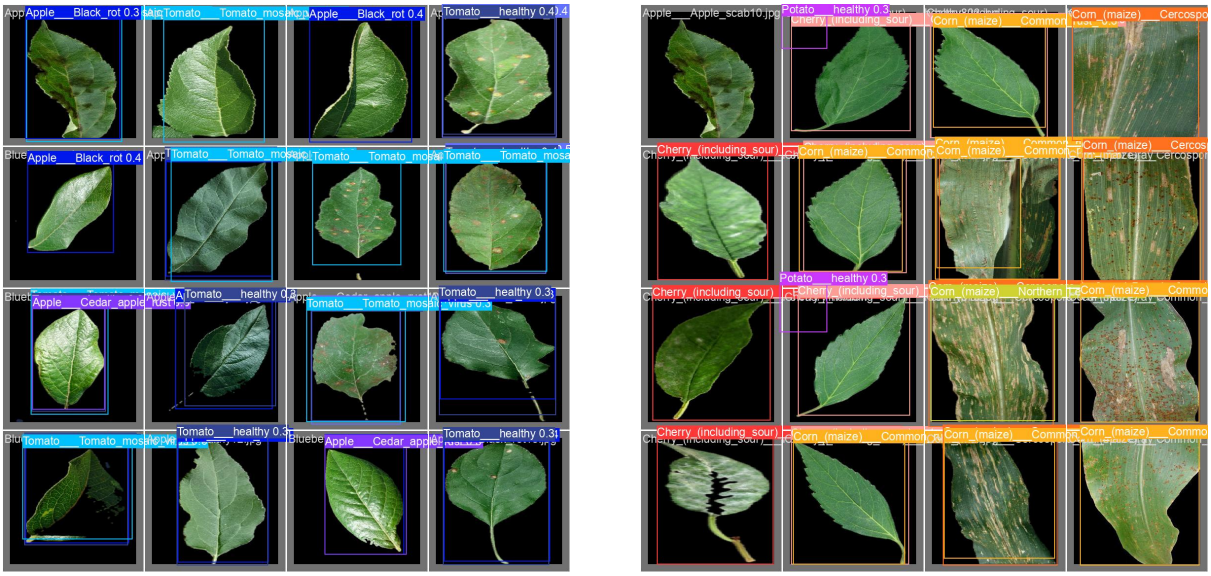
Set-up 1:



((a)) Novel Class ((b)) Base + Novel Class

Figure B.13: Detection Results_1

Set-up 2:



((a)) Novel Class ((b)) Base + Novel Class

Figure B.14: Detection Results_2