Informatics

Plant Detection and State Classification with Machine Learning

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Tobias Eidelpes, BSc
Matrikelnummer 01527193

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.-Prof. Dr. Horst Eidenberger

Wien, 20. Februar 2023

Tobias Eidelpes Horst Eidenberger

Informatics

Plant Detection and State Classification with Machine Learning

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Tobias Eidelpes, BSc
Registration Number 01527193

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.-Prof. Dr. Horst Eidenberger

Vienna, 20th February, 2023

Tobias Eidelpes	Horst Eidenberger

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Tobias Eidelpes, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 20. Februar 2023

Tobias Eidelpes

Danksagung

Ihr Text hier.

Acknowledgements

Enter your text here.

Kurzfassung

Ihr Text hier.

Abstract

Enter your text here.

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xv**

1 Introduction **1**
- 1.1 Motivation and Problem Statement 1
- 1.2 Methodological Approach 3
- 1.3 Thesis Structure 5

2 Theoretical Background **7**
- 2.1 Machine Learning 7
  - 2.1.1 Supervised Learning 8
  - 2.1.2 Artificial Neural Networks 9
  - 2.1.3 Activation Functions 11
    - Identity 11
    - Heaviside Step 11
    - Sigmoid 12
    - Rectified Linear Unit 12
    - Softmax 13
  - 2.1.4 Loss Function 13
  - 2.1.5 Backpropagation 14
- 2.2 Object Detection 15
  - 2.2.1 Traditional Methods 15
    - Viola-Jones Detector 15
    - HOG Detector 15
    - Deformable Part-Based Model 16
  - 2.2.2 Deep Learning Based Methods 16
  - 2.2.3 Two-Stage Detectors 17
    - R-Convolutional Neural Network (CNN) 17
    - SPP-net 18
    - Fast R-CNN 18

Faster R-CNN 18
Feature Pyramid Network 19
2.2.4 One-Stage Detectors 19
You Only Look Once 20
Single Shot MultiBox Detector 20
RetinaNet 20
2.3 Image Classification 21
2.3.1 Traditional Methods 21
2.3.2 Deep Learning Based Methods 22
LeNet-5 22
AlexNet 23
ZFNet 24
GoogLeNet 24
VGGNet 25
ResNet 25
DenseNet 26
MobileNet v3 26
2.4 Transfer Learning 27
2.5 Hyperparameter Optimization 28
2.5.1 Grid Search 29
2.5.2 Random Search 29
2.5.3 Evolution Strategies 30
2.6 Related Work 30

3 Prototype Design **35**
3.1 Requirements 35
3.2 Design 36
3.3 Selected Methods 37
3.3.1 You Only Look Once 37
You Only Look Once (YOLO)v2 39
YOLOv3 39
YOLOv4 40
YOLOv5 41
YOLOv6 41
YOLOv7 41
3.3.2 ResNet 42
3.3.3 Data Augmentation 42
3.3.4 Hyperparameter Optimization 43

4 Prototype Implementation **45**
4.1 Object Detection 45
4.2 Classification 48
4.3 Deployment 48

5 Evaluation **49**
5.1 Methodology 49
5.2 Results 49
5.2.1 Object Detection 49
Test Phase 50
Hyperparameter Optimization 50
5.2.2 Classification 53
Hyperparameter Optimization 54
Class Activation Maps 56
5.2.3 Aggregate Model 59
5.2.4 Non-optimized Model 60
5.2.5 Optimized Model 61
5.3 Discussion 62

6 Conclusion **63**
6.1 Future Work 63

List of Figures **65**

List of Tables **67**

Acronyms **69**

Bibliography **71**

CHAPTER 1

Introduction

Machine learning has seen an unprecedented rise in various research fields during the last few years. Large-scale distributed computing and advances in hardware manufacturing have allowed machine learning models to become more sophisticated and complex. Multi-billion parameter deep learning models show best-in-class performance in Natural Language Processing (NLP) [BMR+20], fast object detection [BWL20] and various classification tasks [ZHT22; AH22]. Agriculture is one of the areas which profits substantially from the automation possible with machine learning.

Large-scale as well as small local farmers are able to survey their fields and gardens with drones or stationary cameras to determine soil and plant condition as well as when to water or fertilize [RRL+20]. Machine learning models play an important role in that process because they allow automated decision-making in real time. While machine learning has been used in large-scale agriculture, it is also a valuable tool for household plants and gardens. By using machine learning to monitor and analyze plant conditions, homeowners can optimize their plant care and ensure their plants are healthy and thriving.

1.1 Motivation and Problem Statement

The challenges to implement an automated system for plant surveying are numerous. First, gathering data in the field requires a network of sensors which are linked to a central server for processing. Since communication between sensors is difficult without proper infrastructure, there is a high demand for processing the data on the sensor itself [MWL22]. Second, differences in local soil, plant and weather conditions require models to be optimized for these diverse inputs. Centrally trained models often lose the nuances present in the data because they have to provide actionable information for a larger area [Awa19]. Third, specialized methods such as hyper- or multispectral imaging in the field provide fine-grained information about the object of interest but come with substantial upfront costs and are of limited interest for gardeners.

To address all of the aforementioned problems, there is a need for an installation which is deployable by homeowners, gathers data using readily available hardware and performs computation on the device without a connection to a central server. The device should be able to visually determine whether the plants in its field of view need water or not and output its recommendation. The recommendation should then be used as a data point off of which homeowners can automatically water their plants with an automated watering system.

The aim of this work is to develop a prototype which can be deployed by gardeners to survey plants and recommend watering or not. To this end, a machine learning model will be trained to first identify the plants in the field of view and then to determine if the plants need water or not. The model should be suitable for edge devices equipped with a Tensor Processing Unit (TPU) or Graphics Processing Unit (GPU) but with otherwise limited processing capabilities. Examples of such systems include Google's Coral development board and the Nvidia Jetson series of single-board computers (SBCs). The model should make use of state-of-the-art algorithms from either classical machine learning or deep learning. The literature review will yield an appropriate machine learning method. Furthermore, the adaption of existing models (transfer learning) for object detection to the domain of plant recognition may provide higher performance than would otherwise be achievable within the time constraints.

The model will be deployed to the SBC and evaluated using established and well-known metrics from the field of machine learning. The evaluation will seek to answer the following questions:

1. *How well does the model work in theory and how well in practice?*

   We will measure the performance of our model with common metrics such as accuracy, F-score, Receiver Operating Characteristic (ROC) curve, Area Under the Curve (AUC), Intersection over Union (IOU) and various mean Average Precision (mAP) measures. These measurements will allow comparisons between our model and existing models. We expect the plant detection part of the model to achieve high scores on the test dataset. However, the classification of plants into stressed and non-stressed will likely prove to be more difficult. The model is limited to physiological markers of water stress and thus will have difficulties with plants which do not overtly display such features.

   Even though models may work well in theory, some do not easily transfer to practical applications. It is, therefore, important to examine if the model is suited for productive use in the field. The evaluation will contain a discussion about the model's transferability because theoretical performance does not automatically guarantee real-world performance due to different environmental conditions.

2. *What are possible reasons for it to work/not work?*

   Even if a model scores high on performance metrics, there might be a mismatch between how researchers think it achieves its goal and how it actually achieves its goal. The results have to be plausible and explainable with its inputs. Otherwise, there can be no confidence in the model's outputs. Conversely, if the model does not work, there must be a reason. We estimate that the curation of the dataset for the training and test phases will play a significant role. Explanations for model out- or underperformance are likely to be found in the structure and composition of the model's inputs.

3. *What are possible improvements to the system in the future?*

   The previous two questions will yield the data for possible improvements to the model and/or our approach. With the decision to include a plant detection step at the start, we hope to create consistent conditions for the stress classification. A downside to this approach is that errors during detection can be propagated through the system and result in adverse effects to overall performance. Although we estimate this problem to be negligible, additional feedback regarding our approach in this way might offer insight into potential improvements. If the model does not work as well as expected, which changes to the approach will yield a better result? Similarly to the previous question, the answer will likely lie in the dataset. A heavy focus on dataset construction and curation will ensure satisfactory model performance.

1.2 Methodological Approach

The methodological approach consists of the following steps:

1. **Literature Review**: The literature review informs the type of machine learning methods which are later applied during the implementation of the prototype.
2. **Dataset Curation**: After selecting the methods to use for the implementation, we have to create our own dataset or use existing ones, depending on availability.
3. **Model Training**: The selected models will be trained with the datasets curated in the previous step.
4. **Optimization**: The selected models will be optimized with respect to their parameters.
5. **Deployment to SBC**: The software prototype will be deployed to the SBC.
6. **Evaluation**: The models will be evaluated extensively and compared to other state-of-the-art systems. During evaluation, the author seeks to provide a basis for answering the research questions.

During the literature review, the search is centered around the terms *plant classification*, *plant state classification*, *plant detection*, *water stress detection*, *machine learning agriculture*, *crop machine learning* and *remote sensing*. These terms provide a solid basis for understanding the state of the art in plant detection and stress classification. We will use multiple search engines such as Google Scholar, Semantic Scholar, the ACM Digital Library, and IEEE Xplore. It is common to only publish research papers in preprint form in the data science and machine learning fields. For this reason, we will also reference arXiv.org for these papers. The work discovered in this way will also lead to further insights about the type of models which are commonly used.

In order to find and select appropriate datasets to train the models on, we will survey the existing big datasets for classes we can use. Datasets such as the Common Objects in Context (COCO) [LMB+15] and PASCAL Visual Object Classes (VOC) [EVW+10] contain the highly relevant class *Potted Plant*. By extracting only these classes from multiple datasets and concatenating them together, it is possible to create one unified dataset which only contains the classes necessary for training the model.

The training of the models will happen in an environment where more computational resources are available than what the SBC offers. We will deploy the final model with the Application Programming Interface (API) to the SBC after training and optimization. Furthermore, training will happen in tandem with a continuous evaluation process. After every iteration of the model, an evaluation run against the test set determines if there has been an improvement in performance. The results of the evaluation feed back into the parameter selection at the beginning of each training phase. Small changes to the training parameters, augmentations or structure of the model are followed by another test phase. The iterative nature of the development of the prototype increases the likelihood that the model's performance is not only locally maximal but also as close as possible to the global maximum.

In the final evaluation phase, we will measure the resulting model against the test set and evaluate its performance with common metrics. The aim is to first provide a solid basis of facts regarding the model(s). Second, the results will be discussed in detail. Third, we will cross-check the results with the hypotheses from section 1.1 and determine whether the aim of the work has been met, and—if not—give reasons for the rejection of all or part of the hypotheses.

Overall, the development of our application follows an evolutionary prototyping process [Dav92; SJJ07]. Instead of producing a full-fledged product from the start, development happens iteratively in phases. The main phases and their order for the prototype at hand are: model selection, implementation, and evaluation. The results of each phase—for example, which model has been selected—inform the decisions which have to be made in the next phase (implementation). In other words, every subsequent phase is dependent on the results of the previous phase. All three phases, in turn, constitute one iteration within the prototyping process. At the start of the next prototype, the results of the previous iteration determine the path forward.

The decision to use an evolutionary prototyping process follows in large part from the problem to be solved (as specified in section 1.1). Since the critical requirements have been established from the start, it is possible to build a solid prototype from the beginning by implementing only those features which are well-understood. The aim is to allow the developer to explore the problem further so that additional requirements which arise during development can be incorporated properly.

The prototyping process is embedded within the concepts of the *Scientific Method*. This thesis not only produces a prototype, but also explores the problem of plant detection and classification scientifically. Exploration of the problem requires making falsifiable hypotheses (see section 1.1), gathering empirical evidence (see section 5.2), and accepting or rejecting the initial hypotheses (see section 5.3). Empirical evidence is provided by measuring the model(s) against out-of-sample test sets. This provides the necessary foundation for acceptance or rejection of the hypotheses.

1.3 Thesis Structure

The first part of the thesis (chapter 2) contains the theoretical basis of the models which we use for the prototype. Chapter 3 goes into detail about the requirements for the prototype, the overall design and architecture of the recognition and classification pipeline, and the structure and unique properties of the selected models. Chapter 4 expands on how the datasets are used during training as well as how the prototype publishes its classification results. Chapter 5 shows the results of the testing phases as well as the performance of the aggregate model. Futhermore, the results are compared with the expectations and it is discussed whether they are explainable in the context of the task at hand as well as benchmark results from other datasets (COCO [LMB+15]). Chapter 6 concludes the thesis with a summary and an outlook on possible improvements and further research questions.

CHAPTER 2

Theoretical Background

This chapter is split into five parts. First, we introduce general machine learning concepts (section 2.1). Second, we provide a survey of object detection methods from early *traditional methods* to one-stage and two-stage deep learning based methods (section 2.2). Third, we go into detail about image classification in general and which approaches have been published in the literature (section 2.3). Fourth, we give a short explanation of transfer learning and its advantages and disadvantages (section 2.4). The chapter concludes with a section on hyperparameter optimization (section 2.5).

2.1 Machine Learning

The term machine learning was first used by Samuel [Sam59] in 1959 in the context of teaching a machine how to play the game Checkers. Mitchell [Mit97] defines learning in the context of programs as:

> A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. [Mit97, p.2]

In other words, if the aim is to learn to win at a game, the performance measure P is defined as the ability to win at that game. The tasks in T are playing the game multiple times, and the experience E is gained by letting the program play the game against itself.

Machine learning is thought to be a sub-field of Artificial Intelligence (AI). AI is a more general term for the scientific endeavour of creating things which possess the kind of intelligence we humans have. Since those things will not have been created *naturally*, their intelligence is termed *artificial.* Within the field of AI there have been other approaches than what is commonly referred to as machine learning today.

A major area of interest in the 1980s was the development of *expert systems*. These systems try to approach problem solving as a rational decision-making process. Starting from a knowledge base, which contains facts and rules about the world and the problem to be solved, the expert system applies an inference engine to arrive at a conclusion. An advantage of these systems is that they can often explain how they came to a particular conclusion, allowing humans to verify and judge the inference process. This kind of explainability is missing in the neural network based approaches of today. However, an expert system needs a significant base of facts and rules to be able to do any meaningful inference. Outside of specialized domains such as medical diagnosis, expert systems have always failed at commonsense reasoning.

Machine learning can be broadly divided into two distinct approaches: *supervised* and *unsupervised*. Supervised learning describes a process where the algorithm receives input values as well as their corresponding output values and tries to learn the function which maps inputs to outputs. This is called supervised learning because the model knows a target to map to. In unsupervised learning, in contrast, algorithms do not have access to labeled data or output values and therefore have to find patterns in the underlying inputs. There can be mixed approaches as in *semi-supervised* learning where a model receives a small amount of labeled data as an aid to better extract the patterns in the unlabeled data. Which type of learning to apply depends heavily on the problem at hand. Tasks such as image classification and speech recognition are good candidates for supervised learning. If a model is required to *generate* speech, text or images, an unsupervised approach makes more sense. We will go into detail about the general approach in supervised learning because it is used throughout this thesis when training the models.

2.1.1 Supervised Learning

The overall steps when training a model with labeled data are as follows:

1. Determine which type of problem is to be solved and select adequate training samples.

2. Gather enough training samples and obtain their corresponding targets (labels). This stage usually requires humans to create a body of ground truth with which the model can compare itself.

3. Select the type of representation of the inputs which is fed to the model. The representation heavily relies on the amount of data which the model can process in a reasonable amount of time. For speech recognition, for example, raw waveforms are rarely fed to any classifier. Instead, humans have to select a less granular and more meaningful representation of the waveforms such as Mel-frequency Cepstral Coefficients (MFCCs). Selecting the representation to feed to the model is also referred to as *feature selection* or *feature engineering*.

4. Select the structure of the model or algorithm and the learning function. Depending on the problem, possible choices are Support Vector Machines (SVMs), CNNs and many more.
5. Train the model on the training set.
6. Validate the results on out-of-sample data by computing common metrics and comparing the results to other approaches.
7. Optionally go back to 4. to select different algorithms or to train the model with different parameters or adjusted training sets. Depending on the results, one can also employ computational methods such as hyperparameter optimization to find a better combination of model parameters.

These steps are generally the same for every type of supervised or semi-supervised machine learning approach. The implementation for solving a particular problem differs depending on the type of problem, how much data is available, how much can reasonably be labeled and any other special requirements such as favoring speed over accuracy.

2.1.2 Artificial Neural Networks

Artificial neural networks are the building blocks of most state-of-the-art models in use today. The computer sciences have adopted the term from biology where it defines the complex structure in the human brain which allows us to experience and interact with the world around us. A neural network is necessarily composed of neurons which act as gatekeepers for the signals they receive. Depending on the inputs—electrochemical impulses, numbers, or other—the neuron *excites* and produces an output value if the right conditions are met. This output value travels via connections to other neurons and acts as an input on their side. Each neuron and connection between the neurons has an associated weight which changes when the network learns. The weights increase or decrease the signal from the neuron. The neuron itself only passes a signal on to its output connections if the conditions of its *activation function* have been met. This is typically a non-linear function. Multiple neurons are usually grouped together to form a *layer* within the network. Multiple layers are stacked one after the other with connections in-between to form a neural network. Layers between the input and output layers are commonly referred to as *hidden layers*. Figure 2.1 shows the structure of a three-layer fully-connected artificial neural network.

The earliest attempts at describing learning machines were by McCulloch and Pitts [MP43] with the idea of the *perceptron*. This idea was implemented in a more general sense by Rosenblatt [Ros57; Ros62] as a physical machine. At its core, the perceptron is the simplest artifical neural network with only one neuron in the center. The neuron takes all its inputs, aggregates them with a weighted sum and outputs 1 if the result is above some threshold θ and 0 if it is not (see equation 2.1). This function is called the

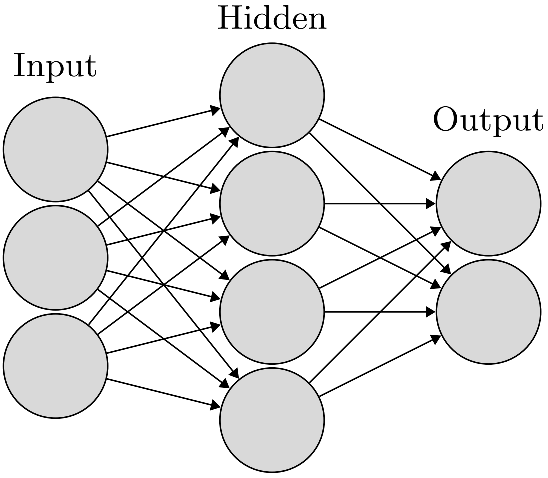

Figure 2.1: Structure of an artificial neural network. Information travels from left to right through the network using neurons and the connections between them.

activation function of a perceptron. A perceptron is a type of binary classifier which can only classify linearly separable variables.

$$y = \begin{cases} 1 \text{ if } \sum_{i=1}^{n} w_i \cdot x_i \geq \theta \\ 0 \text{ if } \sum_{i=1}^{n} w_i \cdot x_i < \theta \end{cases} \tag{2.1}$$

Due to the inherent limitations of perceptrons to only be able to classify linearly separable data, Multilayer Perceptrons (MLPs) are the bedrock of modern artifical neural networks. By adding an input layer, a hidden layer, and an output layer as well as requiring the activation function of each neuron to be non-linear, a MLP can classify also non-linear data. Every neuron in each layer is fully connected to all of the neurons in the next layer and it is the most straightforward case of a feedforward network. Figure 2.1 shows the skeleton of a MLP.

There are two types of artifical neural networks: feedforward and recurrent networks. Their names refer to the way information flows through the network. In a feedforward network, the information enters the network and flows only uni-directionally to the output nodes. In a recurrent network, information can also feed back into previous nodes. Which network is best used depends on the task at hand. Recurrent networks are usually necessary when *context* is needed. For example, if the underlying data to classify is a time series, individual data points have some relation to the previous and next points in the series. Maintaining a bit of state is beneficial because networks should be able to capture these dependencies. However, having additional functionality for feeding information back into previous neurons and layers comes with increased complexity. A feedforward network, as depicted in Figure 2.1, represents a simpler structure.

2.1.3 Activation Functions

Activation functions are the functions *inside* each neuron which receive inputs and produce an output value. The nature of these functions is that they need a certain amount of *excitation* from the inputs before they produce an output, hence the name *activation function*. Activation functions are either linear or non-linear. Linear functions are limited in their capabilities because they cannot approximate certain functions. For example, a perceptron, which uses a linear activation function, cannot approximate the XOR function [MP17]. Non-linear functions, however, are a requirement for neural networks to become *universal approximators* [HSW89]. We will introduce several activation functions which are used in the field of machine learning in the following sections. There exist many more than can be discussed within the scope of this thesis. However, the selection should give an overview of the most used and influential ones in the author's opinion.

Identity

The simplest activation function is the identity function. It is defined as

$$g(x) = x \tag{2.2}$$

If all layers in an artificial neural network use the identity activation function, the network is equivalent to a single-layer structure. The identity function is often used for layers which do not need an activation function per se, but require one to uphold consistency with the rest of the network structure.

Heaviside Step

The Heaviside step function, also known as the unit step function, is a mathematical function that is commonly used in control theory and signal processing to represent a signal that switches on at a specified time and stays on. The function is named after Oliver Heaviside, who introduced it in the late 19th century. It is defined as

$$H(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases} \quad . \tag{2.3}$$

In engineering applications, the Heaviside step function is used to describe functions whose values change abruptly at specified values of time t. We have already mentioned the Heaviside step function in section 2.1.2 when introducing the perceptron. It can only classify linearly separable variables when used in a neural network and is, therefore, not suitable for complex intra-data relationships. A major downside to using the Heaviside step function is that it is not differentiable at $x = 0$ and has a 0 derivative elsewhere. These properties make it unsuitable for use with gradient descent during backpropagation (section 2.1.5).

Sigmoid

The sigmoid activation function is one of the most important functions to introduce non-linearity into the outputs of a neuron. It is a special case of a logistic function and used synonymously with logistic function in machine learning. It is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

It has a characteristic S-shaped curve, mapping each input value to a number between 0 and 1, regardless of input size. This *squashing* property is particularly desirable for binary classification problems because the outputs can be interpreted as probabilities. Additionally to the squashing propery, it is also a saturating function because large values map to 1 and very small values to 0. If a learning algorithm has to update the weights in the network, saturated neurons are very inefficient and difficult to process because the outputs do not provide valuable information. In contrast to the Heaviside step function (section 2.1.3), it is differentiable which allows it to be used with gradient descent optimization algorithms. Unfortunately, the sigmoid function exacerbates the vanishing gradient problem, which makes it unsuitable for training deep neural networks.

Rectified Linear Unit

The Rectified Linear Unit (ReLU) function is defined as

$$f(x) = \max(0, x) = \begin{cases} x, x > 0 \\ 0, x \leq 0 \end{cases} \tag{2.5}$$

which means that it returns the input value if it is positive, and returns zero if it is negative. It was first introduced by Fukushima [Fuk69] in a modified form to construct a visual feature extractor. The ReLU function is nearly linear, and it thus preserves many of the properties that make linear models easy to optimize with gradient-based methods [GBC16]. In contrast to the sigmoid activation function, the ReLU function partially mitigates the vanishing gradient problem and is therefore suitable for training deep neural networks. Furthermore, the ReLU function is easier to calculate than sigmoid functions which allows networks to be trained more quickly. Even though it is not differentiable at 0, it is differentiable everywhere else and often used with gradient descent during optimization.

The ReLU function suffers from the dying ReLU problem, which can cause some neurons to become inactive. Large gradients, which are passed back through the network to update the weights, are typically the source of this. If many neurons are pushed into this state, the model's capability of learning new patterns is diminished. To address this problem, there are two possibilities. One solution is to make sure that the learning rate is not set too high, which reduces the problem but does not fully remove it. Another solution is to use one of the several variants of the ReLU function such as leaky ReLU, Exponential Linear Unit (ELU), and Sigmoid Linear Unit (SiLU).

In recent years, the ReLU function has become the most popular activation function for deep neural networks and is recommended as the default activation function in modern neural networks [GBC16]. Despite its limitations, the ReLU function has become an essential tool for deep learning practitioners and has contributed to the success of many state-of-the-art models in computer vision, natural language processing, and other domains.

Softmax

The softmax activation function is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes. It takes a vector of numbers, known as logits, and scales them into probabilities. The output of the softmax function is a vector with probabilities of each possible outcome, and the probabilities in the vector sum to one for all possible outcomes or classes. In mathematical terms, the function is defined as

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} i \text{ for } i = 1, \ldots, K \text{ and } \vec{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K \tag{2.6}$$

where the standard exponential function is applied to each value in the vector $\vec{z}$ and the result is normalized with the sum of the exponentials.

2.1.4 Loss Function

Loss functions play a fundamental role in machine learning, as they are used to evaluate the performance of a model and guide its training. The choice of loss function can significantly impact the accuracy and generalization of the model. There are various types of loss functions, each with its strengths and weaknesses, and the appropriate choice depends on the specific problem being addressed.

From the definition of a learning program from section 2.1, loss functions constitute the performance measure P against which the results of the learning program are measured. Only by minimizing the error obtained from the loss function and updating the weights within the network is it possible to gain experience E at carrying out a task T. How the weights are updated depends on the algorithm which is used during the *backward pass* to minimize the error. This type of procedure is referred to as *backpropagation* (see section 2.1.5).

One common type of loss function is the mean squared error (MSE) which is widely used in regression problems. The MSE is a popular choice because it is easy to compute and has a closed-form solution, making it efficient to optimize. It does have some limitations, however. For instance, it is sensitive to outliers, and it may not be appropriate for problems with non-normal distributions. MSE measures the average squared difference between predicted and actual values. It is calculated with

$$\text{MSE}_{\text{test}} = \frac{1}{m} \sum_i (\hat{y}^{(\text{test})} - y^{(\text{test})})_i^2 \tag{2.7}$$

where $\hat{y}^{(\text{test})}$ contains the predictions of the model on the test set and $y^{(\text{test})}$ refers to the target labels [GBC16]. It follows that, if $\hat{y}^{(\text{test})} = y^{(\text{test})}$, the error is 0 and the model has produced a perfect prediction.

We cannot, however, take the results of the error on the test set to update the weights during training because the test set must always contain only samples which the model has not seen before. If the model is trained to minimize the MSE on the test set and then evaluated against the same set, the results will be how well the model fits to the test set and not how well it generalizes. The goal, therefore, is to minimize the error on the training set and to compare the results against an evaluation on the test set. If the model achieves very low error rates on the training set but not on the test set, it is likely that the model is suffering from *overfitting*. Conversely, if the model does not achieve low error rates on the training set, it is likely that the model is suffering from *underfitting*.

Goodfellow, Bengio, and Courville [GBC16] writes on MSE: “MSE was popular in the 1980s and 1990s but was gradually replaced by cross-entropy losses and the principle of maximum likelihood as ideas spread between the statistics community and the machine learning community” [GBC16, p.222]. Cross-entropy measures the difference in information between two distinct probability distributions. Specifically, it gives a number on the average total amount of bits needed to represent a message or event from the first probability distribution in the second probability distribution. If there is the case of binary random variables, i.e. only two classes to classify exist, the measure is called binary cross-entropy. Cross-entropy loss is known to outperform MSE for classification tasks and allows the model to be trained faster [SSP03].

2.1.5 Backpropagation

So far, information only flows forward through the network whenever a prediction for a particular input should be made. In order for a neural network to learn, information about the computed loss has to flow backward through the network. Only then can the weights at the individual neurons be updated. This type of information flow is termed *backpropagation* [RHW86]. Backpropagation computes the gradient of a loss function with respect to the weights of a network for an input-output pair. The algorithm computes the gradient iteratively starting from the last layer and works its way backward through the network until it reaches the first layer.

Strictly speaking, backpropagation only computes the gradient, but does not determine how the gradient is used to learn the new weights. Once the backpropagation algorithm has computed the gradient, that gradient is passed to an algorithm which finds a local minimum of it. This step is usually performed by some variant of gradient descent [Cau47].

2.2 Object Detection

From facial detection to fully automated driving—object detection provides the basis for a wide variety of tasks within the computer vision world. While most implementations in the 1990s and early 2000s relied on cumbersome manual feature extraction, current methods almost exclusively leverage a deep learning based approach. This chapter gives an introduction to object detection, explains common problems researchers have faced and how they have been solved, and discusses the two main approaches to object detection via deep learning.

2.2.1 Traditional Methods

Before the advent of powerful GPUs, object detection was commonly done by manually extracting features from images and passing these features on to a classical machine learning algorithm. Early methods were generally far from being able to detect objects in real time.

Viola-Jones Detector

The first milestone was the face detector by Viola and Jones [VJ01; VJ01] which is able to perform face recognition on 384 by 288 pixel (grayscale) images with 15 fps on a 700 MHz Intel Pentium III processor. The authors use an integral image representation where every pixel is the summation of the pixels above and to the left of it. This representation allows them to quickly and efficiently calculate Haar-like features.

The Haar-like features are passed to a modified AdaBoost algorithm [FS95] which only selects the (presumably) most important features. At the end there is a cascading stage of classifiers where regions are only considered further if they are promising. Every additional classifier adds complexity, but once a classifier rejects a sub-window, the processing stops and the algorithm moves on to the next window. Despite their final structure containing 32 classifiers, the sliding-window approach is fast and achieves comparable results to the state of the art in 2001.

HOG Detector

The Histogram of Oriented Gradients (HOG) [DT05] is a feature descriptor used in computer vision and image processing to detect objects in images. It is a detector which detects shape like other methods such as Scale-Invariant Feature Transform (SIFT) [Low99]. The idea is to use the distribution of local intensity gradients or edge directions to describe an object. To this end, the authors divide the image into a grid of cells and calculate a histogram of edge orientations within each cell. Additionally, each histogram is normalized by taking a larger region and adjusting the local histograms based on the larger region's intensity levels. The resulting blocks of normalized gradients are evenly spaced out across the image with some overlap. These patches are then passed as a feature vector to a classifier.

Dalal and Triggs [DT05] successfully use the HOG with a linear SVM for classification to detect humans in images. They work with images of 64 by 128 pixels and make sure that the image contains a margin of 16 pixels around the person. Decreasing the border by either enlarging the person or reducing the overall image size results in worse performance. Unfortunately, their method is far from being able to process images in real time—a 320 by 240 image takes roughly a second to process.

Deformable Part-Based Model

Deformable Part-Based Models (DPMs) [FMR08] were the winners of the VOC challenge in the years 2007, 2008, and 2009. The method is heavily based on the previously discussed HOG since it also uses HOG descriptors internally. The authors addition is the idea of learning how to decompose objects during training and classifying/detecting the decomposed parts during inference. The HOG descriptors are computed on different scales to form a HOG feature pyramid. Coarse features are more easily identified at the top of the pyramid while details are present at the lower end of the pyramid. The coarse features are obtained by calculating the histograms over fairly large areas, whereas smaller image patches are used for the detailed levels. A root filter works on the coarse levels by detecting general features of the object of interest. If the goal is to detect a face, for example, the root filter detects the contours of the face. Smaller part filters provide additional information about the individual parts of the object. For the face example, these filters capture information about the eyes, mouth and nose.

The idea of detecting detail at different scales is not unlike what happens with the later CNNs. The individual layers of a CNN often describe higher level features in the earlier layers and provide additional lower level information as the network increases in depth. Girshick et al. [GID$^+$15] argue that DPMs *are* in fact CNNs because they can be formulated as CNNs by unrolling each step of the algorithm into a corresponding CNN layer.

2.2.2 Deep Learning Based Methods

After the publication of the DPM, the field of object detection did not make significant advances regarding speed or accuracy. Only the (re-)introduction of CNNs by Krizhevsky, Sutskever, and Hinton [KSH12] with their AlexNet architecture and their subsequent win of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 gave the field a new influx of ideas. The availability of the 12 million labeled images in the ImageNet dataset [DDS$^+$09] allowed a shift from focusing on better methods to being able to use more data to train models. Earlier models had difficulties with making use of the large dataset since training was unfeasible. AlexNet, however, provided an architecture which was able to be trained on two GPUs within 6 days. For an in depth overview of AlexNet see section 2.3.2. Object detection networks from 2014 onward either follow a *one-stage* or *two-stage* detection approach. The following sections go into detail about each model category.

2.2.3 Two-Stage Detectors

As their name implies, two-stage detectors consist of two stages which together form a complete object detection pipeline. Commonly, the first stage extracts Regions of Interest (ROIs) which might contain relevant objects to detect. The second stage operates on the extracted ROIs and returns a vector of class probabilities. Since the computation in the second stage is performed for every ROI, two-stage detectors are often not as efficient as one-stage detectors.

R-CNN

Girshick et al. [GDD$^+$14] were the first to propose using feature representations of CNNs for object detection. Their approach consists of generating around 2000 region proposals and passing these on to a CNN for feature extraction. The fixed-length feature vector is used as input for a linear SVM which classifies the region. They name their method R-CNN, where the R stands for region.

R-CNN uses selective search to generate region proposals [UvdSG$^+$13].The authors use selective search's *fast mode* to generate the 2000 proposals and warp (i.e. aspect ratios are not retained) each proposal into the image dimensions required by the CNN. The CNN, which matches the architecture of AlexNet [KSH12], generates a 4096-dimensional feature vector and each feature vector is scored by a linear SVM for each class. Scored regions are selected/discarded by comparing each region to other regions within the same class and rejecting them if there exists another region with a higher score and greater IOU than a threshold. The linear SVM classifiers are trained to only label a region as positive if the overlap, as measured by IOU, is above 0.3.

While the approach of generating region proposals is not new, using a CNN purely for feature extraction is. Unfortunately, R-CNN is far from being able to operate in real time. The authors report that it takes 13 s/image on a GPU and 53 s/image on a Central Processing Unit (CPU) to generate the region proposals and feature vector. In some sense, these processing times are a step backward from the DPMs introduced in section 2.2.1. However, the authors showed that CNNs can function perfectly well as feature extractors, even if their processing performance is not yet up to par with traditional methods. Furthermore, R-CNN crushes DPMs on the VOC 2007 challenge with a mAP of 58.5% [GDD$^+$14] versus 33.7% (DPM-v5 [GFM; FGM$^+$10]). This was enough to spark renewed interest in CNNs and—with better availability of large data sets and GPU processing capabilities—opened the way for further research in that direction.

SPP-net

A year after the publication of R-CNN, He et al. [HZR$^+$15] introduce the concept of Spatial Pyramid Pooling (SPP) to allow CNNs to accept arbitrarily sized instead of fixed-size input images. They name their method SPP-net and it outputs a fixed-length feature vector of the input image.

SPP layers operate in-between the convolutional and fully-connected layers of a CNN. Since the fully-connected layers require fixed-size inputs but the convolutional layers do not, SPP layers aggregate the information from convolutional layers and pass the resulting fixed-size outputs to the fully-connected layers. This approach allows only passing the full image through the convolutional layers once and calculating features with the SPP layer from these results. This avoids the redundant computations for each ROI present in R-CNN and provides a speedup of 24-102 times while achieving even better metrics on the VOC 2007 data set at a mAP of 59.2%.

Fast R-CNN

Fast R-CNN was proposed by Girshick [Gir15] to fix the three main problems R-CNN and SPP-net have. The first problem is that the training for both models is multi-stage. R-CNN finetunes the convolutional network which is responsible for feature extraction and then trains SVMs to classify the feature vectors. The third stage consists of training the bounding box regressors. The second problem is the training time which is on the order of multiple days for deep convolutional networks. The third problem is the processing time per image which is (depending on the convolutional network) upwards of 13 s/image.

Fast R-CNN deals with these problems by having an architecture which allows it to take in images and object proposals at once and process them simultaneously to arrive at the results. The outputs of the network are the class an object proposal belongs to and 4 scalar values representing the bounding box of the object. Unfortunately, this approach still requires a separate object proposal generator such as selective search [UvdSG$^+$13].

Faster R-CNN

Faster R-CNN [RHG$^+$15; RHG$^+$17]—as the name implies—is yet another improvement building on R-CNN, SPP-net and Fast R-CNN. Since the bottleneck in performance with previous approaches has been the object proposal generator, the authors of Faster R-CNN introduce a Region Proposal Network (RPN) to predict bounding boxes and objectness in one step. As with previous networks, the proposals are then passed to the detection network.

RPNs work by using the already present convolutional features in Fast R-CNN and adding additional layers on top to also regress bounding boxes and objectness scores per location. Instead of relying on a pyramid structure such as with SPP-net (see section 2.2.3), RPNs use *anchor boxes* as a basis for the bounding box regressor. These anchor boxes are predefined for various scales and aspect ratios and serve as starting points for the regressor to properly fit a bounding box around an object.

The RPN makes object proposal generation inexpensive and possible on GPUs. The whole network operates on an almost real time scale by being able to process 5 images/s and maintaining high state-of-the-art mAP values of 73.2% (VOC 2007). If the detection network is switched from VGGNet [LD15] to ZF-Net [ZF14], Faster R-CNN is able to achieve 17 images/s, albeit at a lower mAP of 59.9%.

Feature Pyramid Network

Feature Pyramid Networks (FPNs) were first introduced by Lin et al. [LDG⁺17] to use the hierarchical pyramid structure inherent in CNNs to compute feature maps on different scales. Previously, detectors were only using the features of the top most (coarse) layers because it was computationally too expensive to use lower (fine-grained) layers. By leveraging feature maps on different scales, FPNs are able to better detect small objects because predictions are made independently on all levels. FPNs are an important building block of many state-of-the-art object detectors.

A FPN first computes the feature pyramid bottom-up with a scaling step of two. The lower levels capture less semantic information than the higher levels, but include more spatial information due to the higher granularity. In a second step, the FPN upsamples the higher levels such that the dimensions of two consecutive layers are the same. The upsampled top layer is merged with the layer beneath it via element-wise addition and convolved with a one by one convolutional layer to reduce channel dimensions and to smooth out potential artifacts introduced during the upsampling step. The results of that operation constitute the new *top layer* and the process continues with the layer below it until the finest resolution feature map is generated. In this way, the features of the different layers at different scales are fused to obtain a feature map with high semantic information but also high spatial information.

Lin et al. [LDG⁺17] report results on COCO with a mAP@0.5 of 59.1% with a Faster R-CNN structure and a ResNet-101 backbone. Their submission does not include any specific improvements such as hard negative mining [SGG16] or data augmentation.

2.2.4 One-Stage Detectors

One-stage detectors, in contrast to two-stage detectors, combine the proposal generation and detection tasks into one neural network such that all objects can be retrieved in a single step. Since the proposal generation in two-stage detectors is a costly operation and usually the bottleneck, one-stage detectors are significantly faster overall. Their speeds allow them to be deployed to low-resource devices such as mobile phones while still providing real time object detection. Unfortunately, their detection accuracy trailed the two-stage approaches for years, especially for small and/or dense objects.

You Only Look Once

YOLO was the first one-stage detector introduced by Redmon et al. [RDG$^+$16]. It divides each image into regions and predicts bounding boxes and classes of objects simultaneously. This allows it to be extremely fast at up to 155 fps with a mAP of 52.7% on VOC 2007. The accuracy results were not state of the art at the time because the architecture trades localization accuracy for speed, especially for small objects. These issues have been gradually dealt with in later versions of YOLO as well as in other one-stage detectors such as Single Shot MultiBox Detector (SSD). Since a later version of YOLO is used in this work, we refer to section 3.3.1 for a thorough account of its architecture.

Single Shot MultiBox Detector

SSD was proposed by Liu et al. [LAE$^+$16] and functions similarly to YOLO in that it does not need an extra proposal generation step, but instead detects and classifies objects in one go. The aim of one-stage detectors is to be considerably faster and at least as accurate as two-stage detectors. While YOLO paved the way for one-stage detectors, the detection accuracy is significantly lower than state-of-the-art two-stage detection approaches such as Faster RCNN. SSD combines generating detections on multiple scales and an end-to-end architecture to achieve high accuracy as well as high speed.

SSD is based on a standard CNN such as VGG16 [LD15] and adds additional feature layers to the network. The CNN, which the detector is using to extract features, has its last fully-connected layer removed such that the output of the CNN is a scaled down representation of the input image. The extra layers are intended to capture features at different scales and compare them during training to a range of default anchor boxes. This idea comes from MultiBox [EST$^+$14], but is implemented in SSD with a slight twist: during matching of default boxes to the ground truth, boxes with a Jaccard overlap (IOU) of less than 0.5 are discarded. In one-stage detector terms, the feature extractor is the *backbone* whereas the extra layers constitute the *head* of the network. The outputs of the extra layers contain features for smaller regions with higher spatial information. Making use of these additional feature maps is what sets SSD apart from YOLO and results in SSD being able to detect smaller and denser objects as well.

The authors report results on VOC 2007 for their SSD300 and SSD512 model varieties. The number refers to the size of the input images. SSD300 outperforms Fast R-CNN by 1.1 percentage points (mAP 66.9% vs 68%). SSD512 outperforms Faster R-CNN by 1.7% mAP. If trained on the VOC 2007, 2012 and COCO train sets, SSD512 achieves a mAP of 81.5% on the VOC 2007 test set. SSD's speed is at 46 fps which, although lower than Fast YOLO's 155 fps, is still in real time. Furthermore, SSD has a mAP which is almost 22% higher than Fast YOLO.

RetinaNet

One-stage detectors before 2017 always trailed the accuracy of top two-stage detectors on common and difficult benchmark data sets such as COCO. Lin et al. [LGG$^+$17] investigated what the culprit for the lower accuracy scores could be and found that the severe class imbalance between foreground and background instances is the problem. They introduce a novel loss function called *Focal Loss* which replaces the standard cross-entropy loss. Focal loss down-weights the importance of easy negative examples during training and instead focuses on instances which are harder but provide more information.

Focal loss is based on cross-entropy loss but includes a scaling factor which decreases while the classification confidence increases. In other words, if the confidence that an object belongs to a particular class is already high, focal loss outputs a small value such that the weight updates during backpropagation are only marginally affected by the current example. The model can thus focus on examples which are harder to achieve a good confidence score on.

Lin et al. [LGG$^+$17] implement their focal loss with a simple one-stage detector called *RetinaNet*. It makes use of previous advances in object detection and classification by including a FPN on top of a ResNet [HZR$^+$16] as the backbone and using anchors for the different levels in the feature pyramid. Attached to the backbone are two subnetworks which classify anchor boxes and regress them to the ground truth boxes. The results are that the RetinaNet-101-500 version (with an input size of 500 px) achieves a mAP of 34.4% at a speed of around 11 fps on the COCO data set.

2.3 Image Classification

Image classification, in contrast to object detection, is a slightly easier task because there is no requirement to localize objects in the image. Instead, image classification operates always on the image as a whole rather than individual parts of it. As has been demonstrated in the last chapter, object detection methods often rely on advances in image classification to accurately detect objects. After objects have been localized, we humans want to know what kind of object it is and that is where image classification methods become useful.

This section goes into detail about various image classification methods. We first give a short summary on how image classification was commonly done before CNNs became the de facto standard. Afterwards, we will introduce common and influential approaches leveraging CNNs and discuss problems and solutions for training large networks.

2.3.1 Traditional Methods

Similarly to early object detection algorithms, traditional methods rely on manual feature extraction and subsequent classification with classical algorithms. Passing raw images to the algorithms is often not feasible due to the immense information contained in just one image. Furthermore, a raw image contains a signal to noise ratio which is too low for a computer to successfully learn properties about the image. Instead, humans—with the aid of image processing methods—have to select a lower-dimensional representation of the input image and then pass this representation to a classifier. This process of manually reducing the dimensions and complexity of an image to the part which is *relevant* is termed *feature engineering*.

Manual feature engineering requires selecting an appropriate representation for the task at hand. For example, if the task is to classify images which show an object with a special texture, a feature engineer will likely select an image representation which clearly pulls the texture into the foreground. In other words, engineers help the classifier by preprocessing the image such that the most discriminative features are easily visible. The methods with which an image representation is created is called *feature descriptor*.

In line with the different ways objects can present themselves on images, there have been many feature descriptors proposed. Most of the feature descriptors used in object detection are also used in image classification (see HOG and SIFT from section 2.2.1) because their representational power is useful in both domains.

2.3.2 Deep Learning Based Methods

Manual feature engineering is a double-edged sword. Although it allows to have a high amount of control, it also necessitates the engineer to select a meaningful representation for training the downstream classifier. Often, humans make unconscious assumptions about the problem to be solved as well as the available data and how best to extract features. These assumptions can have a detrimental effect on classification accuracy later on because the best-performing feature descriptor lies outside of the engineer's purview. Therefore, instead of manually preparing feature vectors for the classifier, researchers turned to allowing an Artificial Neural Network (ANN) to recognize and extract the most relevant aspects of an image on its own, without human intervention. Attention is thus mostly given to the structure of the ANN and less to the preparation of inputs.

The idea of automatic generation of feature maps via ANNs gave rise to CNNs. Early CNNs [LBD+89] were mostly discarded for practical applications because they require much more data during training than traditional methods and also more processing power during inference. Passing 224 by 224 pixel images to a CNN, as is common today, was simply not feasible if one wanted a reasonable inference time. With the development of GPUs and supporting software such as the Compute Unified Device Architecture (CUDA) toolkit, it was possible to perform many computations in parallel. The architecture of CNNs lends itself well to parallel processing and thus CNNs slowly but surely overtook other image classification methods.

LeNet-5

LeNet-5, developed and described by Lecun et al. [LBB+98], laid the foundation of CNNs as we still use them today. The basic structure of convolutional layers with pooling layers in-between and one or more fully-connected layers at the end has been iterated on many times since then. LeCun et al. [LBD+89] introduced the first version of LeNet when describing their system for automatic handwritten zip code recognition. They applied backpropagation with Stochastic Gradient Descent (SGD) and used the scaled hyperbolic tangent as the activation function. The error function with which the weights are updated is MSE.

The architecture of LeNet-5 is composed of two convolutional layers, two pooling layers and a dense block of three fully-connected layers. The input image is a grayscale image of 32 by 32 pixels. The first convolutional layer generates six feature maps, each with a scale of 28 by 28 pixels. Each feature map is fed to a pooling layer which effectively downsamples the image by a factor of two. By aggregating each two by two area in the feature map via averaging, the authors are more likely to obtain relative (to each other) instead of absolute positions of the features. To make up for the loss in spatial resolution, the following convolutional layer increases the amount of feature maps to 16 which aims to increase the richness of the learned representations. Another pooling layer follows which reduces the size of each of the 16 feature maps to five by five pixels. A dense block of three fully-connected layers of 120, 84 and 10 neurons respectively serves as the actual classifier in the network. The last layer uses the euclidean Radial Basis Function (RBF) to compute the class an image belongs to (0-9 digits).

The performance of LeNet-5 was measured on the Modified National Institute of Standards and Technology (MNIST) database which consists of 70000 labeled images of handwritten digits. The MSE on the test set is 0.95%. This result is impressive considering that character recognition with a CNN had not been done before. However, standard machine learning methods of the time, such as manual feature engineering and SVMs, achieved a similar error rate, even though they are much more memory-intensive. LeNet-5 was conceived to take advantage of the (then) large MNIST database. Since there were not many data sets available at the time, especially with more samples than in the MNIST database, CNNs were not widely used even after their viability had been demonstrated by Lecun et al. [LBB+98]. Only in 2012 Krizhevsky, Sutskever, and Hinton [KSH12] reintroduced CNNs (see section 2.2.2) and since then most state-of-the-art image classification methods have used them.

AlexNet

AlexNet's main contributions are the use of ReLUs, training on multiple GPUs, Local Response Normalization (LRN) and overlapping pooling [KSH12]. As mentioned in section 2.1.3, ReLUs introduce non-linearity into the network. Instead of using the traditional non-linear activation function tanh, where the output is bounded between -1 and 1, ReLUs allow the output layers to grow as high as training requires it. Normalization before an activation function is usually used to prevent the neuron from saturating, as would be the case with tanh. Even though ReLUs do not suffer from saturation, the authors found that LRN reduces the top-1 error rate by 1.4% [KSH12]. Overlapping pooling, in contrast to regular pooling, does not easily accept the dominant pixel values per window. By smoothing out the pooled information, bias is reduced and networks are slightly more resilient to overfitting. Overlapping pooling reduces the top-1 error rate by

0.4% [KSH12]. In aggregate, these improvements result in a top-5 error rate of below 25% at 16.4%.

These results demonstrated that CNNs can extract highly relevant feature representations from images. While AlexNet was only concerned with the classification of images, it did not take long for researchers to apply CNNs to the problem of object detection.

ZFNet

ZFNet's [ZF14] contributions to the image classification field are twofold. First, the authors develop a way to visualize the internals of a CNN with the use of *deconvolution* techniques. Second, with the added knowledge gained from looking *inside* a CNN, they improve AlexNet's structure. The deconvolution technique is essentially the reverse operation of a CNN layer. Instead of pooling (downsampling) the results of the layer, Zeiler and Fergus [ZF14] *unpool* the max-pooled values by recording the maximum positions of the maximum value per kernel. The maximum values are then put back into each two by two area (depending on the kernel size). This process loses information because a max-pooling layer is not invertible. The subsequent ReLU function can be easily inverted because negative values are squashed to zero and and positive values are retained. The final deconvolution operation concerns the convolutional layer itself. In order to *reconstruct* the original spatial dimensions (before convolution), a transposed convolution is performed. This process reverses the downsampling which happens during convolution.

With these techniques in place, the authors visualize the first and second layers of the feature maps present in AlexNet. They identify multiple problems with their structure such as aliasing artifacts and a mix of low and high frequency information without any mid frequencies. These results indicate that the filter size in AlexNet is too large at 11 by 11 and the authors reduce it to seven by seven. Additionally, they modify the original stride of four to two. These two changes result in an improvement in the top-5 error rate of 1.6% over their own replicated AlexNet result of 18.1%.

GoogLeNet

GoogLeNet, also known as Inception v1, was proposed by Szegedy et al. [SLJ$^+$15] to increase the depth of the network without introducing too much additional complexity. Since the relevant parts of an image can often be of different sizes, but kernels within convolutional layers are fixed, there is a mismatch between what can realistically be detected by the layers and what is present in the data set. Therefore, the authors propose to perform multiple convolutions with different kernel sizes and concatenating them together before sending the result to the next layer. Unfortunately, three by three and five by five kernel sizes within a convolutional layer can make the network too expensive to train. The authors add one by one convolutions to the outputs of the previous layer before passing the result to the three by three and five by five convolutions. The one by one convolutions have the effect that the channels of the inputs (feature maps) are reduced and are thus easier to process by the subsequent larger filters.

GoogLeNet consists of nine Inception modules stacked one after the other and a *stem* with convolutions at the beginning as well as two auxiliary classifiers which help retain the gradient during backpropagation. The auxiliary classifiers are only used during training. The authors submitted multiple model versions to the 2004 ILSVRC and their ensemble prediction model consisting of 7 GoogleNets achieved a top-5 error rate of 6.67%, which resulted in first place.

VGGNet

In the quest for ever-more layers and deeper networks, Simonyan and Zisserman [SZ15] propose an architecture which is based on small-resolution kernels (receptive fields) for each convolutional layer. They make extensive use of stacked three by three kernels and one by one convolutions with ReLUs in-between to decrease the number of parameters. Their choice relies on the fact that two three by three convolutional layers have an effective receptive field of one five by five layer. The advantage is that they introduce additional non-linearities by having two ReLUs instead of only one. The authors provide five different networks with increasing number of parameters based on these principles. The smallest network has a depth of eight convolutional layers and three fully-connected layers for the head (11 in total). The largest network has 16 convolutional and three fully-connected layers (19 in total). The fully-connected layers are the same for each architecture, only the layout of the convolutional layers varies.

The deepest network with 19 layers achieves a top-5 error rate on ILSVRC 2014 of 9%. If trained with different image scales in the range of $S \in [256, 512]$, the same network achieves a top-5 error rate of 8% (test set at scale 256). By combining their two largest architectures and multi-crop as well as dense evaluation, they achieve an ensemble top-5 error rate of 6.8%, while their best single network with multi-crop and dense evaluation results in 7%, thus beating the single-net submission of GoogLeNet (see section 2.3.2) by 0.9%.

ResNet

The 22-layer structure of GoogLeNet [SLJ$^+$15] and the 19-layer structure of VGGNet [SZ15] showed that *going deeper* is beneficial for achieving better classification performance. However, the authors of VGGNet already note that stacking even more layers does not lead to better performance because the model is *saturated*. He et al. [HZR$^+$16] provide a solution to the vanishing gradient as well as the degradation problem by introducing *skip connections* to the network. They call their resulting network architecture *ResNet* and since it is used in this work, we will give a more detailed account of its structure in section 3.3.2.

DenseNet

The authors of DenseNet [HLV+17] go one step further than ResNets by connecting every convolutional layer to every other layer in the chain. Previously, each layer was connected in sequence with the one before and the one after it. Residual connections establish a link between the previous layer and the next one, but still do not always propagate enough information forward. These *shortcut connections* from earlier layers to later layers are thus only taking place in an episodic way for short sections in the chain. DenseNets are structured in a way such that every layer receives the feature map of every previous layer as input. In ResNets, information from previous layers is added on to the next layer via element-wise addition. DenseNets concatenate the features of the previous layers. The number of feature maps per layer has to be kept low so that the subsequent layers can still process their inputs. Otherwise, the last layer in each dense block would receive too many channels which increases computational complexity.

The authors construct their network from multiple dense blocks which are connected via a batch normalization layer, a one by one convolutional layer and a two by two pooling layer to reduce the spatial resolution for the next dense block. Each dense block consists of a Batch Normalization (BN) layer, a ReLU layer and a three by three convolutional layer. In order to keep the number of feature maps low, the authors introduce a *growth rate* k as a hyperparameter. The growth rate can be as low as $k = 4$ and still allow the network to learn highly relevant representations.

In their experiments, the authors evaluate different combinations of dense blocks and growth rates against ImageNet. Their DenseNet-161 ($k = 48$) achieves a top-5 error rate with single-crop of 6.15% and with multi-crop 5.3%. Their DenseNet-BC variant requires only one third of the amount of parameters of a ResNet-101 network to achieve the same test error on the CIFAR-10 dataset.

MobileNet v3

MobileNet v3 by Howard et al. [HSC+19] is the third iteration of the original MobileNet architecture [HZC+17]. MobileNets use depthwise separable convolution instead of regular convolution. In the latter, the kernel in each convolutional layer is applied to all channels of the input simultaneously. Depthwise convolution applies the kernel to each channel separately instead and the output is then convolved in a second layer with a one by one kernel over all channels. The second step is also called a *pointwise convolution* because it squeezes the number of channels per one by one input field into n output channels.

The effect of using depthwise separable convolutions is that the amount of computation needed is severely reduced compared to standard convolutions. A standard convolutional layer with a kernel size of $D_K \times D_K$, an output feature map size of $D_F \times D_F$, M input channels and N output channels has a computational cost of

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F. \tag{2.8}$$

A depthwise separable convolution, however, has a computational cost of

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F. \tag{2.9}$$

The first summand refers to the cost of the depthwise convolution and added to it is the cost for the pointwise convolution. The authors demonstrate that the reduction in computational cost is

$$\frac{1}{N} + \frac{1}{D_K^2} \tag{2.10}$$

which—at a kernel size of three by three—results in a smaller computational cost of between eight to nine times. MobileNet v2 [SHZ⁺18] introduced *inverted residuals* and *linear bottlenecks* and MobileNet v3 [HSC+19] brought *squeeze and excitation layers* among other improvements. These concepts led to better classification accuracy at the same or smaller model size. The authors evaluate a large and a small variant of MobileNet v3 on Imagenet on single-core phone processors and achieve a top-1 accuracy of 75.2% and 67.4% respectively.

2.4 Transfer Learning

Transfer learning refers to the application of a learning algorithm to a target domain by utilizing knowledge already learned from a different source domain [ZQD+21]. The learned representations from the source domain are thus *transferred* to solve a related problem in another domain. Transfer learning works because semantically meaningful information an algorithm has learned from a (large) data set is often meaningful in other contexts as well, even though the *new problem* is not exactly the same problem for which the original model had been trained for. An analogy to day-to-day life as humans can be drawn with sports. Intuitively, skills learned during soccer such as ball control, improved endurance and strategic thinking are often also useful in other ball sports. Someone who is adept at certain kinds of sports will likely be able to pick up similar types much faster.

In mathematical terms, Pan and Yang [PY10] define transfer learning as:

> [PY10, p.1347] Given a source domain $\mathcal{D}_S$ and learning task $\mathcal{T}_S$, a target domain $\mathcal{D}_T$ and learning task $\mathcal{T}_T$, transfer learning aims to help improve the learning of the target predictive function $f_T(\cdot)$ in $\mathcal{D}_T$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$.

In the machine learning world, collecting and labeling data for training a model is often time consuming, expensive and sometimes not possible. Deep learning based models especially require substantial amounts of data to be able to robustly classify images or solve other tasks. Semi-supervised or unsupervised (see section 2.1) learning approaches can partially mitigate this problem, but having accurate ground truth data is usually a requirement nonetheless. Through the publication of large labeled data sets such as via the ILSVRCs, a basis for (pre-)training exists from which the model can be optimized for downstream tasks.

Transfer learning is not a panacea, however. Care has to be taken to only use models which have been pretrained in a source domain which is similar to the target domain in terms of feature space. While this may seem to be an easy task, it is often not known in advance if transfer learning is the correct approach. Furthermore, choosing whether to only remove the fully-connected layers at the end of a pretrained model or to fine-tune all parameters introduces at least one additional hyperparameter. These decisions have to be made by comparing the source domain with the target domain, how much data in the target domain is available, how much computational resources are available and observing which layers are responsible for which features. Since earlier layers usually contain low-level and later layers high-level information, resetting the weights of the last few layers or replacing them with different ones entirely is also an option.

To summarize, while transfer learning is an effective tool and is likely a major factor in the proliferation of deep learning based models, not all domains are suited for it. The additional decisions which have to be made as a result of using transfer learning can introduce more complexity than would otherwise be necessary for a particular problem. It does, however, allow researchers to get started quickly and to iterate faster because popular network architectures pretrained on Imagenet are integrated into the major machine learning frameworks. Transfer learning is used extensively in this work to train a classifier as well as an object detection model.

2.5 Hyperparameter Optimization

While a network is learning, the parameters of its layers are updated. These parameters are *learnable* in the sense that changing them should bring the model closer to solving a problem. Updating these parameters happens during the learning/training phase. Hyperparameters, on the other hand, are not included in the learning process because they are fixed before the model starts to train. They are fixed because hyperparameters concern the structure, architecture and learning parameters of the model and without having those in place, a model cannot start training.

Model designers have to carefully define values for a wide range of hyperparameters. Which hyperparameters have to be set is determined by the type of model which is being used. A SVM, for example, has a penalty parameter C which indicates to the network how lenient it should be when misclassifying training examples. The type of kernel to use is also a hyperparameter for any SVM and can only be answered by looking at the distribution of the underlying data. In neural networks the range of hyperparameters is even greater because every part of the network architecture such as how many layers to stack, which layers to stack, which kernel sizes to use in each CNN layer and which activation function(s) to use in-between the layers is a parameter which can be altered.

Finding the best combination of some or all of the available hyperparameters is called *hyperparameter tuning.*

Hyperparameter tuning can be and is often done manually by researchers where they select values which *have been known to work well.* This approach—while it works to some extent—is not optimal because adhering to *best practice* precludes parameter configurations which would be closer to optimality for a given data set. Furthermore, manual tuning requires a deep understanding of the model itself and how each parameter influences it. Biases present in a researcher's understanding are detrimental to finding optimal hyperparameters and the amount of possible combinations can quickly get intractable. Instead, automated methods to search the hyperparameter space offer an unbiased and more efficient approach to hyperparameter tuning. This type of algorithmic search is called *hyperparameter optimization.*

2.5.1 Grid Search

There are multiple possible strategies to opt for when optimizing hyperparameters. The straightforward approach is to do grid search. In grid search, all hyperparameters are discretized and all possible combinations mapped to a search space. The search space is then sampled for configurations at evenly spaced points and the resulting vectors of hyperparameter values are evaluated. For example, if a model has seven hyperparameters and three of those can take on a continuous value, these three variables have to be discretized. In practical terms this means that the model engineer chooses suitable discrete values for said hyperparameters. Once all hyperparameters are discrete, all possible combinations of the hyperparameters are evaluated. If each of the seven hyperparameters has three discrete values, the number of possible combinations is

$$3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 = 3^7 = 2187. \tag{2.11}$$

For this example, evaluating 2187 possible combinations can already be intractable depending on the time required for each run. Further, grid search requires that the resolution of the grid is determined beforehand. If the points on the grid (combinations) are spaced too far apart, the chance of finding a global optimum is lower than if the grid is dense. However, a dense grid results in a higher number of possible combinations and thus more time is required for an exhaustive search. Additionally, grid search suffers from the *curse of dimensionality* because the number of evaluations scales exponentially with the number of hyperparameters.

2.5.2 Random Search

Random search [PDD$^+$09] is an alternative to grid search which often provides configurations which are similar or better in the same amount of time than ones obtained with grid search [BB12]. Random search performs especially well in high-dimensional environments because the hyperparameter response surface is often of *low effective dimensionality*

[BB12]. That is, a low number of hyperparameters disproportionately affects the performance of the resulting model and the rest has a negligible effect. We use random search in this work to improve the hyperparameters of our classification model.

2.5.3 Evolution Strategies

Evolution strategies follow a population-based model where the search strategy starts from initial random configurations and evolves the hyperparameters through *mutation* and *crossover*. Mutation randomly changes the value of a hyperparameter and crossover creates a new configuration by mixing the values of two configurations. Hyperparameter optimization with evolutionary strategies roughly goes through the following stages [BBL$^+$23].

1. Set the hyperparameters to random initial values and create a starting population of configurations.
2. Evaluate each configuration.
3. Rank all configurations according to a fitness function.
4. The best-performing configurations are selected as *parents*.
5. Child configurations are created from the parent configurations by mutation and crossover.
6. Evaluate the child configurations.
7. Go to step three and repeat the process until a termination condition is reached.

This strategy is more efficient than grid search or random search, but requires a substantial amount of iterations for good solutions and can thus be too expensive for hyperparameter optimization [BBL$^+$23]. We use an evolution strategy based on a genetic algorithm in this work to optimize the hyperparameters of our object detection model.

2.6 Related Work

The literature on machine learning in agriculture is broadly divided into four main areas: livestock management, soil management, water management, and crop management [BTD$^+$21]. Of those four, water management only makes up about 10% of all surveyed papers during the years 2018–2020. This highlights the potential for research in this area to have a high real-world impact. Besides agriculture, algorithmic approaches to watering house plants have not been studied at all to the best of our knowledge. Related work thus mostly focuses on a small selection of plants which are used for agricultural purposes. Nevertheless, the methods presented in those works are of interest for our own work.

Su et al. [SCL+20] used traditional feature extraction and preprocessing techniques to train various machine learning models for classifying water stress for a wheat field. They took top-down images of the field using an Unmanned Aerial Vehicle (UAV), segmented wheat pixels from background pixels and constructed features based on spectral intensities and color indices. The features are fed into a SVM with a Gaussian kernel and optimized using Bayesian optimization. Their results of 92.8% accuracy show that classical machine learning approaches can offer high classification scores if meaningful features are chosen. One disadvantage is that feature extraction is often a tedious task involving trial and error (see section 2.3.1). Advantages are the small data set and the short training time (3 s) required to obtain a good result.

Similarly, López-García et al. [LIM+22] investigated the potential for UAVs to determine water stress for vineyards using RGB and multispectral imaging. The measurements of the UAV were taken at 80 m with a common off-the-shelf Advanced Photo System type-C (APS-C) sensor. At the same time, stem water measurements were taken with a pressure chamber to be able to evaluate the performance of an ANN against the ground truth. The RGB images were used to calculate the Green Canopy Cover (GCC) which was also fed to the model as input. The model achieves a high determination coefficient R^2 of 0.98 for the 2018 season on RGB data with a relative error of $RE = 10.84\,\%$. However, their results do not transfer well to the other seasons under survey (2019 and 2020).

Zhuang et al. [ZWJ+17] showed that water stress in maize can be detected early on and, therefore, still provide actionable information before the plants succumb to drought. They installed a camera which took 640 by 480 pixel RGB images every two hours. A simple linear classifier (SVM) segmented the image into foreground and background using the green color channel. The authors constructed a 14-dimensional feature space consisting of color and texture features. A Gradient Boosted Decision Tree (GBDT) model classified the images into water stressed and non-stressed and achieved an accuracy of 90.39 %. Remarkably, the classification was not significantly impacted by illumination changes throughout the day.

An et al. [ALL+19] used the ResNet50 model (see section 2.3.2) as a basis for transfer learning and achieved high classification scores (ca. 95%) on maize. Their model was fed with 640 by 480 pixel images of maize from three different viewpoints and across three different growth phases. The images were converted to grayscale which turned out to slightly lower classification accuracy. Their results also highlight the superiority of Deep Convolutional Neural Networkss (DCNNs) compared to manual feature extraction and GBDTs.

Chandel et al. [CCR+21] investigated deep learning models in depth by comparing three well-known CNNs. The models under scrutiny were AlexNet (see section 2.3.2), GoogLeNet (see section 2.3.2), and Inception v3. Each model was trained with a dataset containing images of maize, okra, and soybean at different stages of growth and under stress and no stress. The researchers did not include an object detection step before image classification and compiled a fairly small dataset of 1200 images. Of the three models, GoogLeNet beat the other two with a sizable lead at a classification accuracy of >94% for all three types of crop. The authors attribute its success to its inherently deeper structure and application of multiple convolutional layers at different stages. Unfortunately, all of the images were taken at the same 45° ± 5° angle and it stands to reason that the models would perform significantly worse on images taken under different conditions.

Ramos-Giraldo et al. [RRL$^+$20] detected water stress in soybean and corn crops with a pretrained model based on DenseNet-121 (see section 2.3.2). Low-cost cameras deployed in the field provided the training data over a 70-day period. They achieved a classification accuracy for the degree of wilting of 88%.

In a later study, the same authors [RRM$^+$20] deployed their machine learning model in the field to test it for production use. They installed multiple Raspberry Pis with attached Raspberry Pi Cameras which took images in 30 min intervals. The authors had difficulties with cameras not working and power supply issues. Furthermore, running the model on the resource-constrained RPis proved difficult and they had to port their TensorFlow model to a TensorFlow Lite model. This conversion lowered their classification scores slightly since it was sometimes off by one water stress level. Nevertheless, their architecture allowed for reasonably high classification scores on corn and soybean with a low-cost setup.

Azimi, Kaur, and Gandhi [AKG20] demonstrate the efficacy of deep learning models versus classical machine learning models on chickpea plants. The authors created their own dataset in a laboratory setting for stressed and non-stressed plants. They acquired 8000 images at eight different angles in total. For the classical machine learning models, they extracted feature vectors using SIFT and HOG. The features are fed into three classical machine learning models: SVM, k-Nearest Neighbors (k-NN), and a Decision Tree (DT) using the Classification and Regression Tree (CART) algorithm. On the deep learning side, they used their own CNN architecture and the pretrained ResNet-18 (see section 2.3.2) model. The accuracy scores for the classical models was in the range of 60 % to 73 % with the SVM outperforming the two others. The CNN achieved higher scores at 72 % to 78 % and ResNet-18 achieved the highest scores at 82 % to 86 %. The results clearly show the superiority of deep learning over classical machine learning. A downside of their approach lies in the collection of the images. The background in all images was uniformly white and the plants were prominently placed in the center. It should, therefore, not be assumed that the same classification scores can be achieved on plants in the field with messy and noisy backgrounds as well as illumination changes and so forth.

Venal, Fajardo, and Hernandez [VFH19] combine a standard CNN architecture with a SVM for classification. The CNN acts as a feature extractor and instead of using the last fully-connected layers of an off-the-shelf CNN, they replace them with a SVM. They use this classifier to determine which biotic or abiotic stresses soybeans suffer from. Their data set consists of 65184 64 by 64 RGB images of which around 40000 were used for training and 6000 for testing. All images show a close-up of a soybean leaf. Their CNN architecture makes use of three Inception modules (see section 2.3.2) with Squeeze-Excitation (SE) blocks and BN layers in-between. Their model achieves an average F_1-score of 97% and an average accuracy of 97.11% on the test set. Overall, the hybrid structure of their model is promising, but it is not clear why only using the CNN as a feature extractor provides better results than using it also for classification.

Aversano, Bernardi, and Cimitile [ABC22] perform water stress classification on images of tomato crops obtained with a UAV. Their data set consists of 6600 thermal and 6600 optimal images which have been segmented using spectral clustering. They use two VGG-19 networks (see section 2.3.2) which extract features from the thermal (network one) and optical (network two) images. Both feature extractors are merged together via a fully-connected and softmax layer to predict one of three classes: water excess, well-watered and water deficit. The authors select three hyperparameters (image resolution, optimization algorithm and batch size) and optimize them for accuracy. The best classifier works with a resolution of 512 px, SGD and a batch size of 32. This configuration achieves an accuracy of 80.5% and an F_1-score of 79.4% on the validation set. To test whether the optical or thermal images are more relevant for classification, the authors conduct an ablation study. The results show that the network with the optical images alone achieves an F_1-score of 74% while only using the thermal images gives an F_1-score of 62%.

A significant problem in the detection of water stress is posed by the evolution of indicators across time. Since physiological features such as leaf wilting progress as time passes, the additional time domain has to be taken into account. To make use of these spatiotemporal patterns, Azimi, Wadhawan, and Gandhi [AWG21] propose the application of a CNN Long Short-Term Memory Network (CNN-LSTM) architecture. The model was trained on chickpea plants and achieves a robust classification accuracy of >97%.

All of the previously mentioned studies solely focus on either one specific type of plant or on a small number of them. Furthermore, the researchers construct their datasets in homogeneous environments which often do not mimic real-world conditions. Finally, there exist no studies on common household or garden plants. This fact may be attributed to the propensity for funding to come from the agricultural sector. It is thus desirable to explore how plants other than crops show water stress and if there is additional information to be gained from them.

CHAPTER 3

Prototype Design

1. Closely examine the used models (YOLOv7 and ResNet) regarding their structure as well as unique features. Additionally, list the augmentations which were done during training of the object detector. Finally, elaborate on the process of hyperparameter optimization (train/val structure, metrics, genetic evolution and random search).

Estimated 10 pages for this chapter.

3.1 Requirements

The basic requirements for the prototype have been introduced in section 1.1 and stem from the research questions defined in the same section. The aim of this work is to detect household plants, classify them into water-stressed or healthy, and to continuously publish the results via a Representational State Transfer (REST) API. To this end, a portable SBC such as the Nvidia Jetson Nano stores the trained models locally and uses them for inference on images which are periodically taken with an attached camera.

The prototype is thus required to be running the models on its own without help from a central server or other computational resource. However, because the results are published via a REST service, internet access is necessary to be able to retrieve the predictions.

Other functional requirements are that the inference on the device for both models does not take too long (i.e. not longer than a few seconds per image). Even though plants are not known to grow extremely rapidly from one minute to the next, keeping the inference time low results in a more resource efficient prototype. As such, it is possible to run the device off of a battery which completes the self-contained nature of the prototype.

From an evaluation perspective, the models are required to attain a reasonable level of accuracy. It is difficult to determine said level beforehand, but considering the task as

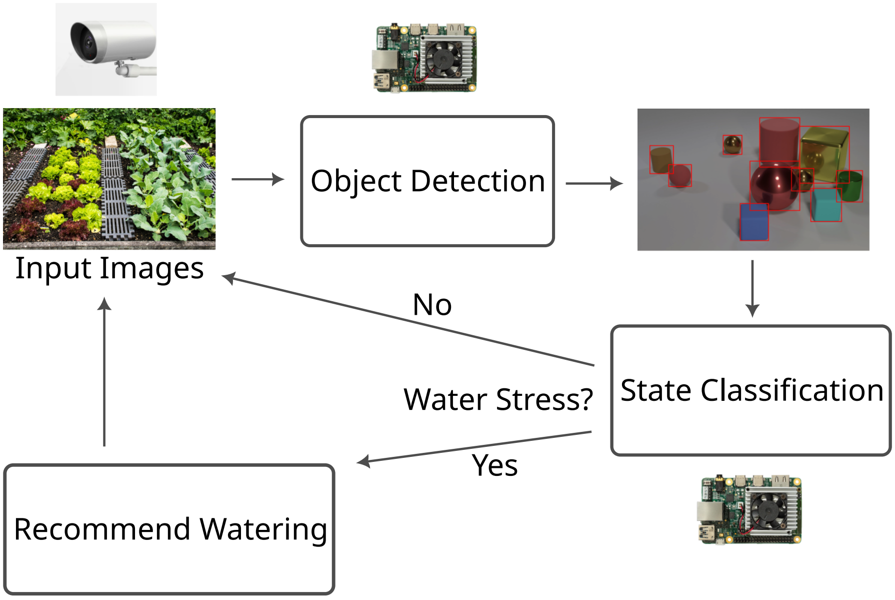

Figure 3.1: Methodological approach for the prototype. The prototype will run in a loop which starts at the top left corner. First, the camera attached to the prototype takes images of plants. These images are passed to the models running on the prototype. The first model generates bounding boxes for all detected plants. The bounding boxes are used to cut out the individual plants and pass them to the state classifier in sequence. The classifier outputs a probability score indicating the amount of stress the plant is experiencing. After a set amount of time, the camera takes a picture again and the process continues indefinitely.

well as general object detection and classification benchmarks such as COCO [LMB+15], we expect a mAP of around 40% and precision and recall values of 70%.

3.2 Design

Figure 3.1 shows the overall processing loop which happens on the device. The camera is directly attached to the Nvidia Jetson Nano via a Camera Serial Interface (CSI) cable. Since the cable is quite rigid, the camera must be mounted on a small *stand* such as a tripod. Images coming in from the camera are then passed to the object detection model running on the Nvidia Jetson Nano. The model detects all plants in the image and returns the coordinates of a bounding box per plant. These coordinates are used to *cut out* each plant from the original image. The cutout is then passed to the second model running on the Nvidia Jetson Nano which determines if the plant is water-stressed or not. The percentage values of the prediction are mapped to a scale between one and ten, where ten indicates that the plant is in a very dire state. This number is available via a REST endpoint with additional information such as current time as well as how long it has been since the state has been better than three. The endpoint publishes this information for every plant which has been detected.

The water stress prediction itself consists of two stages. First, plants are detected and, second, each individual plant is classified. This two-stage approach lends itself well to a two-stage model structure. Since the first stage is an object detection task, we employ an object detection model and pass the individual plant images to a second model—the classifier.

While most object detection models could be trained to determine the difference between water-stressed and healthy, the reason for this two-stage design lies in the availability of data. To our knowledge, there are no sufficiently large enough data sets available which contain labeling information for water-stressed and healthy. Instead, most data sets only classify common objects such as plane, person, car, bicycle, and so forth (e.g. COCO [LMB$^+$15]). However, the classes *plant* and *houseplant* are present in most data sets and provide the basis for our object detection model. The size of these data sets allows us to train the object detection model with a large number of samples which would have been unfeasible to label on our own. The classifier is then trained with a smaller data set which only comprises individual plants and their associated classification (*stressed* or *healthy*).

Both data sets (object detection and classification) only allow us to train and validate each model separately. A third data set is needed to evaluate the detection/classification pipeline as a whole. To this end, we construct our own data set where all plants per image are labeled with bounding boxes as well as the classes *stressed* or *healthy*. This data set is small in comparison to the one with which the object detection model is trained, but suffices because it is only used for evaluation. Labeling each sample in the evaluation data set manually is still a laborious task which is why each image is *preannotated* by the already existing object detection and classification model. The task of labeling thus becomes a task of manually correcting the annotations which have been generated by the models.

3.3 Selected Methods

In the following sections we will go into detail about the two selected architectures for our prototype. The object detector we chose—YOLOv7—is part of a larger family of models which all function similarly, but have undergone substantial changes from version to version. In order to understand the used model, we trace the improvements to the YOLO family from version one to version seven. For the classification stage, we have opted for a ResNet architecture which is also described in detail.

3.3.1 You Only Look Once

The YOLO family of object detection models started in 2015 when [RDG$^+$16] published the first version. Since then there have been up to 16 updated versions depending on how one counts. The original YOLO model marked a shift from two-stage detectors to one-stage detectors as is evident in its name. Two-stage detectors (see section 2.2.3) rely on a proposal generation step and then subsequent rejection or approval of each proposal to detect objects. Generating proposals, however, is an expensive procedure which limits the amount of object detections per second. YOLO dispenses with the extra proposal generation step and instead provides a unified *one-stage* detection approach.

The first version of YOLO [RDG$^+$16] framed object detection as a single regression problem which allows the model to directly infer bounding boxes with class probabilities from image pixels. This approach has the added benefit that YOLO sees an entire image at once, allowing it to capture more contextual information than with sliding window or region proposal methods. However, YOLO still divides an image into regions which are called *grid cells*, but this is just a simple operation and does not rely on external algorithms such as selective search [UvdSG$^+$13]. The number of bounding box proposals within YOLO is much lower than with selective search as well (98 versus 2000 per image).

The architecture of YOLO is similar to GoogleNet (see section 2.3.2), but the authors do not use inception modules directly. The network contains 24 convolutional layers in total where most three by three layers are fed a reduced output from a one by one layer. This approach reduces complexity substantially—as has been demonstrated with GoogleNet. Every block of convolutional layers is followed by a two by two maxpool layer for downsampling. The model expects an input image of size 448 by 448 pixels, but has been pretrained on ImageNet with half that resolution (i.e. 224 by 224 pixels). After the convolutional layers, the authors add two fully-connected layers to produce an output of size $7 \times 7 \times 30$. This output tensor is chosen because the VOC data set has 20 classes C and each grid cell produces two bounding boxes B where each bounding box is described by x, y, w, h and the confidence. With a grid size of $S = 7$, the output is thus $S \times S \times (B \cdot 5 + C) = 7 \times 7 \times 30$.

Each grid cell is responsible for a detected object if the object's center coordinates (x, y) fall within the bounds of the cell. Furthermore, every cell can only predict *one* object which leads to problems with images of dense objects. In that case, a finer grid size is needed. The w and h of a bounding box is relative to the image as a whole which allows the bounding box to span more than one grid cell.

Since the authors frame object detection as a regression problem of bounding box coordinates (center point (x, y), width w, and height h), object probabilities per box, and class probabilities, they develop a loss function which is a sum of five parts. The first part describes the regression for the bounding box center coordinates (sum of squared differences), the second part the width and height of the box, the third part the confidence of there being an object in a box, the fourth part the confidence if there is no actual object in the box, and the fifth part the individual class probabilities (see equation 3.1). The two constants λ_{coord} and λ_{noobj} are weighting factors which increase the loss from bounding box coordinate predictions and decrease the loss from confidence predictions for boxes without objects. These are set to $\lambda_{\text{coord}} = 5$ and $\lambda_{\text{noobj}} = 0.5$.

$$
\begin{aligned}
\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \Big[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \Big] \\
+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \Big[\Big(\sqrt{w_i} - \sqrt{\hat{w}_i} \Big)^2 \Big] \\
+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \Big[C_i - \hat{C}_i \Big]^2 \\
+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \Big[C_i - \hat{C}_i \Big]^2 \\
+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \Big[p_{i(c)} - \hat{p}_{i(c)} \Big]^2
\end{aligned}
\tag{3.1}
$$

The original YOLO model has a few limitations. It only predicts one class per bounding box and can only accommodate two bounding boxes per grid cell. YOLO thus has problems detecting small and dense objects. The most severe problem, however, is the localization accuracy. The loss function treats errors in small bounding boxes similarly to errors in big bounding boxes even though small errors have a higher impact on small bounding boxes than big ones. This results in a more lenient loss function for IOUs of small bounding boxes and, therefore, worse localization.

YOLOv2

YOLOv2 [RF17] incorporates multiple improvements such as BN layers, higher resolution inputs, a fully-convolutional architecture, anchor boxes, dimension priors, and multi-scale training. Of particular interest is the use of anchor boxes to localize bounding boxes. Instead of regressing arbitrary bounding box sizes, YOLOv2 predicts the bounding box offsets from a set of predefined boxes which are called *anchor boxes*. The authors note that finding a good set of prior anchor boxes by hand is error-prone and suggest finding them via k-means clustering (dimension priors). They select five anchor boxes per grid cell which still results in high recall, but does not introduce too much complexity.

These additional details result in an improved mAP of 78.6% on the VOC 2007 data set compared to 63.4% of the previous YOLO version. YOLOv2 still maintains a fast detection rate at 40 fps (mAP 78.6%) and up to 91 fps (mAP 69%).

YOLOv3

YOLOv3 [RF18] provided additional updates to the YOLOv2 model. To be competitive with the deeper network structures of state-of-the-art models at the time, the authors introduce a deeper feature extractor called Darknet-53. It makes use of the residual connections popularized by ResNet [HZR+16] (see section 2.3.2). Darknet-53 is more accurate than Darknet-19 and compares to ResNet-101, but can process more images per second (78 fps versus 53 fps). The activation function throughout the network is still leaky ReLU, as in earlier versions.

YOLOv3 uses multi-scale predictions to achieve better detection ratios across object sizes. Inspired by FPNs (see section 2.2.3), YOLOv3 uses predictions at different scales from the feature extractor and combines them to form a final prediction. Combining the features from multiple scales is often done in the *neck* of the object detection architecture.

Around the time of the publication of YOLOv3, researchers started to use the terminology *backbone*, *neck* and *head* to describe the architecture of object detection models. The feature extractor (Darknet-53 in this case) is the *backbone* and provides the feature maps which are aggregated in the *neck* and passed to the *head* which outputs the final predictions. In some cases there are additional postprocessing steps in the head such as Non Maximum Suppression (NMS) to eliminate duplicate or suboptimal detections.

While YOLOv2 had problems detecting small objects, YOLOv3 performs much better on them (Average Precision (AP) of 18.3% versus 5% on COCO). The authors note, however, that the new model sometimes has comparatively worse results with larger objects. The reasons for this behavior are unknown. Additionally, YOLOv3 is still lagging behind other detectors when it comes to accurately localizing objects. The COCO evaluation metric was changed from the previous $\mathrm{AP}_{0.5}$ to the mAP between 0.5 to 0.95 which penalizes detectors which do not achieve close to perfect IOU scores. This change highlights YOLOv3's weakness in that area.

YOLOv4

Keeping in line with the aim of carefully balancing accuracy and speed of detection, Bochkovskiy, Wang, and Liao [BWL20] publish the fourth version of YOLO. The authors investigate the use of what they term *bag of freebies*—methods which increase training time while increasing inference accuracy without sacrificing inference speed. A prominent example of such methods is data augmentation (see section 3.3.3). Specifically, the authors propose to use mosaic augmentation which lowers the need for large mini-batch sizes. They also use new features such as weighted residual connections [SGZ16], a modified Spatial Attention Module (SAM) [WPL+18], a modified Path Aggregation Network (PANet) [LQQ+18] for the neck, Complete Intersection over Union (CIoU) loss [ZWL+20] for the detector and the Mish activation function [Mis20].

Taken together, these additional improvements yield a mAP of 43.5% on the COCO test set while maintaining a speed of above 30 fps on modern GPUs. YOLOv4 was the first version which provided results on all scales (S, M, L) that were better than almost all other detectors at the time without sacrificing speed.

YOLOv5

The author of YOLOv5 [Joc20] ported the code from YOLOv4 from the Darknet framework to PyTorch which facilitated better interoperability with other Python utilities. New in this version is the pretraining algorithm called AutoAnchor which adjusts the anchor boxes based on the data set at hand. This version also implements a genetic algorithm for hyperparameter optimization (see section 2.5.3) which is used in our work as well.

Version 5 comes in multiple architectures of various complexity. The smallest—and therefore fastest—version is called YOLOv5n where the *n* stands for *nano*. Additional versions with increasing parameters are YOLOv5s (small), YOLOv5m (medium), YOLOv5l (large), and YOLOv5x (extra large). The smaller models are intended to be used in resource constrained environments such as edge devices, but come with a cost in accuracy. Conversely, the larger models are for tasks where high accuracy is paramount and enough computational resources are available. The YOLOv5x model achieves a mAP of 50.7% on the COCO test data set.

YOLOv6

The authors of YOLOv6 [LLJ⁺22] use a new backbone based on RepVGG [DZM⁺21] which they call EfficientRep. They also use different losses for classification (Varifocal loss [ZWD⁺21]) and bounding box regression (Scylla Intersection over Union (SIoU) [Gev22]/Generalized Intersection over Union (GIoU) [RTG⁺19]). YOLOv6 is made available in eight scaled version of which the largest achieves a mAP of 57.2% on the COCO test set.

YOLOv7

At the time of implementation of our own plant detector, YOLOv7 [WBL22] was the newest version within the YOLO family. Similarly to YOLOv4, it introduces more trainable bag of freebies which do not impact inference time. The improvements include the use of Extended Efficient Layer Aggregation Networks (E-ELANs) (based on Efficient Layer Aggregation Networks (ELANs) [WLY22]), joint depth and width model scaling techniques, reparameterization on module level, and an auxiliary head—similarly to GoogleNet (see section 2.3.2)—which assists during training. The model does not use a pretrained backbone, it is instead trained from scratch on the COCO data set. These changes result in much smaller model sizes compared to YOLOv4 and a mAP of 56.8% with a detection speed of over 30 fps.

We use YOLOv7 in our own work during the plant detection stage because it was the fastest and most accurate object detector at the time of implementation.

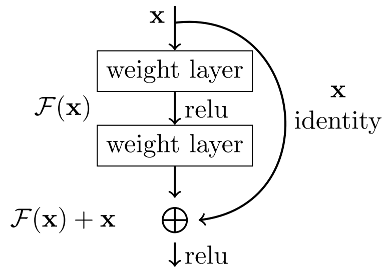

Figure 3.2: Residual connections: information from previous layers flows into subsequent layers before the activation function is applied. The symbol $\bigoplus$ represents simple element-wise addition. Figure redrawn from He et al. [HZR+16].

3.3.2 ResNet

Early research [BSF94; GB10] already demonstrated that the vanishing/exploding gradient problem with standard gradient descent and random initialization adversely affects convergence during training and results in worse performance than would be otherwise achievable with the same architecture. If a neural network is trained with gradient descent by the application of the chain rule (backpropagation), weight updates are passed from the later layers back through the network to the early layers. Unfortunately, with some activation functions (notably tanh), the gradient can be very small and decreases exponentially the further it passes through the network. The effect being that the early layers do not receive any weight updates which can stop the learning process entirely.

There are multiple potential solutions to the vanishing gradient problem. Different weight initialization schemes [GB10; SA15] as well as batch normalization layers [IS15] can help mitigate the problem. The most effective solution yet, however, was proposed as *residual connections* by He et al. [HZR+16]. Instead of connecting each layer only to the previous and next layer in a sequential way, the authors add the input of the previous layer to the output of the next layer. This is achieved through the aforementioned residual or skip connections (see figure 3.2).

3.3.3 Data Augmentation

Go over the data augmentation methods which are used during training for the object detector:

- HSV-hue
- HSV-saturation
- HSV-value
- translation

- scaling
- inversion (left-right)
- mosaic

Estimated 1 page for this section.

3.3.4 Hyperparameter Optimization

Go into detail about the process used to optimize the detection and classification models, what the training set looks like and how a best-performing model was selected on the basis of the metrics.

Estimated 2 pages for this section.

CHAPTER 4

Prototype Implementation

4.1 Object Detection

Describe how the object detection model was trained and what the training set looks like. Include a section on hyperparameter optimization and go into detail about how the detector was optimized.

The object detection model was trained for 300 epochs on 79204 images with 284130 ground truth labels. The weights from the best-performing epoch were saved. The model's fitness for each epoch is calculated as the weighted average of mAP@0.5 and mAP@0.5:0.95:

$$f_{epoch} = 0.1 \cdot \mathsf{mAP@0.5} + 0.9 \cdot \mathsf{mAP@0.5{:}0.95} \tag{4.1}$$

Figure 4.1 shows the model's fitness over the training period of 300 epochs. The gray vertical line indicates the maximum fitness of 0.61 at epoch 133. The weights of that epoch were frozen to be the final model parameters. Since the fitness metric assigns the mAP at the higher range the overwhelming weight, the mAP@0.5 starts to decrease after epoch 30, but the mAP@0.5:0.95 picks up the slack until the maximum fitness at epoch 133. This is an indication that the model achieves good performance early on and continues to gain higher confidence values until performance deteriorates due to overfitting.

Overall precision and recall per epoch are shown in figure 4.2. The values indicate that neither precision nor recall change materially during training. In fact, precision starts to decrease from the beginning, while recall experiences a barely noticeable increase. Taken together with the box and object loss from figure 4.3, we speculate that the pre-trained model already generalizes well to plant detection because one of the categories in the COCO [LMB+15] dataset is *potted plant*. Any further training solely impacts the

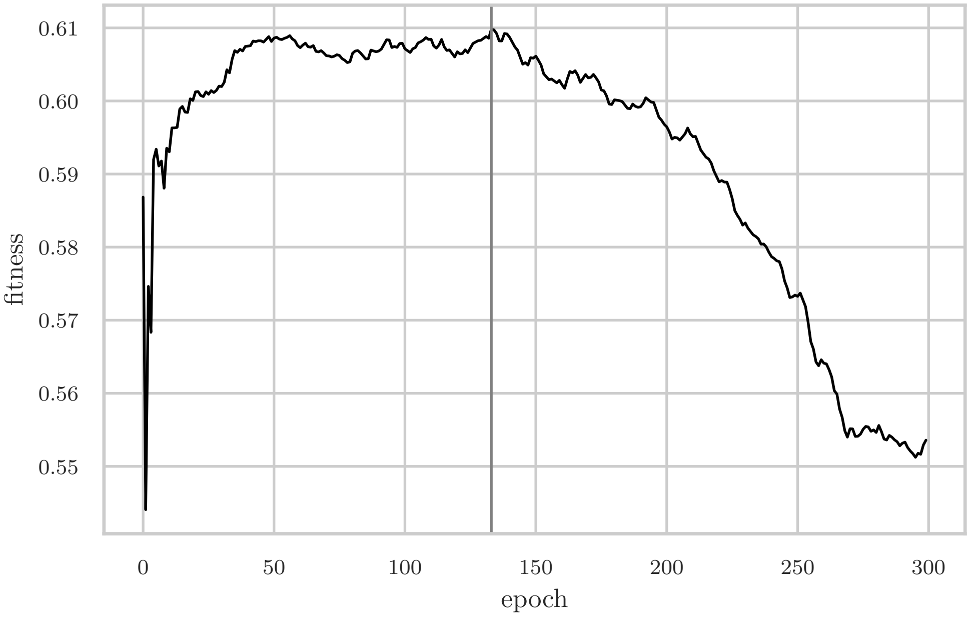

Figure 4.1: Object detection model fitness for each epoch calculated as in equation 4.1. The vertical gray line at 133 marks the epoch with the highest fitness.

confidence of detection, but does not lead to higher detection rates. This conclusion is supported by the increasing `mAP@0.5:0.95` until epoch 133.

Further culprits for the flat precision and recall values may be found in bad ground truth data. The labels from the Open Images Dataset (OID) are sometimes not fine-grained enough. Images which contain multiple individual—often overlapping—plants are labeled with one large bounding box instead of multiple smaller ones. The model recognizes the individual plants and returns tighter bounding boxes even if that is not what is specified in the ground truth. Therefore, it is prudent to limit the training phase to relatively few epochs in order to not penalize the more accurate detections of the model. The smaller bounding boxes make more sense considering the fact that the cutout is passed to the classifier in a later stage. Smaller bounding boxes help the classifier to only focus on one plant at a time and to not get distracted by multiple plants in potentially different stages of wilting.

The box loss decreases slightly during training which indicates that the bounding boxes become tighter around objects of interest. With increasing training time, however, the object loss increases, indicating that less and less plants are present in the predicted bounding boxes. It is likely that overfitting is a cause for the increasing object loss from epoch 40 onward. Since the best weights as measured by fitness are found at epoch 133 and the object loss accelerates from that point, epoch 133 is probably the correct cutoff before overfitting occurs.

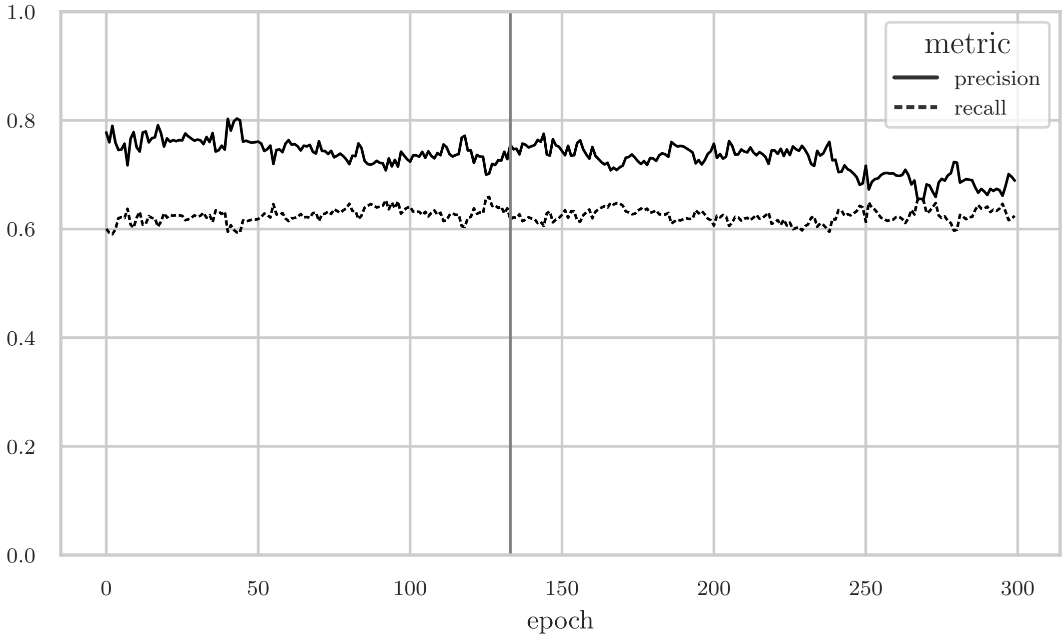

Figure 4.2: Overall precision and recall during training for each epoch. The vertical gray line at 133 marks the epoch with the highest fitness.

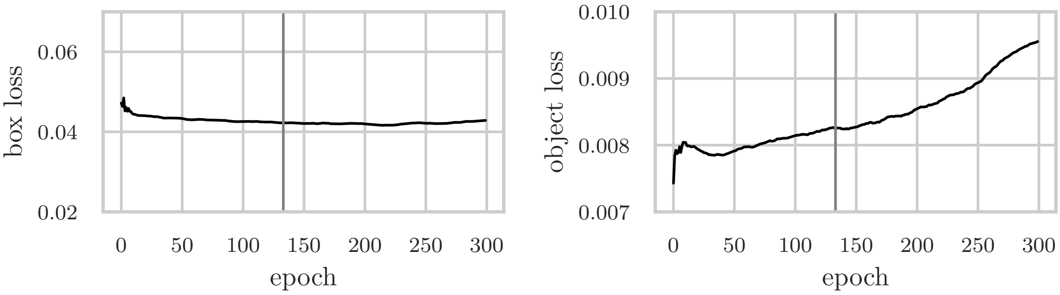

Figure 4.3: Box and object loss measured against the validation set of 3091 images and 4092 ground truth labels. The class loss is omitted because there is only one class in the dataset and the loss is therefore always zero.

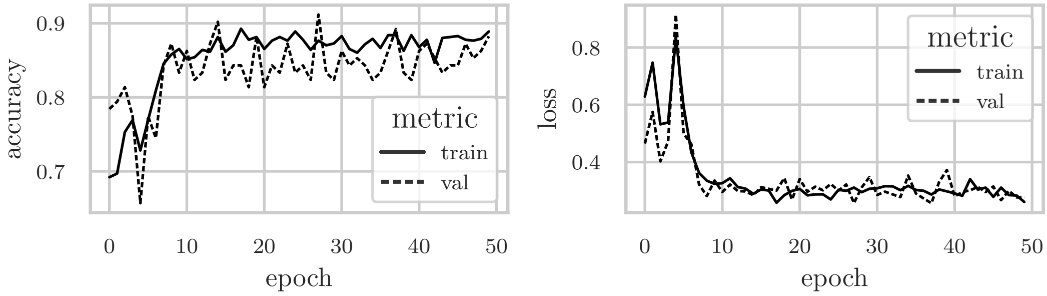

Figure 4.4: Accuracy and loss during training of the classifier. The model converges quickly, but additional epochs do not cause validation loss to increase, which would indicate overfitting. The maximum validation accuracy of 0.9118 is achieved at epoch 27.

Estimated 2 pages for this section.

4.2 Classification

Describe how the classification model was trained and what the training set looks like. Include a subsection hyperparameter optimization and go into detail about how the classifier was optimized.

The dataset was split 85/15 into training and validation sets. The images in the training set were augmented with a random crop to arrive at the expected image dimensions of 224 pixels. Additionally, the training images were modified with a random horizontal flip to increase the variation in the set and to train a rotation invariant classifier. All images, regardless of their membership in the training or validation set, were normalized with the mean and standard deviation of the ImageNet [DDS$^+$09] dataset, which the original Residual Neural Network (ResNet) model was pre-trained with. Training was done for 50 epochs and the best-performing model as measured by validation accuracy was selected as the final version.

Figure 4.4 shows accuracy and loss on the training and validation sets. There is a clear upwards trend until epoch 20 when validation accuracy and loss stabilize at around 0.84 and 0.3, respectively. The quick convergence and resistance to overfitting can be attributed to the model already having robust feature extraction capabilities.

Estimated 2 pages for this section.

4.3 Deployment

Describe the Jetson Nano, how the model is deployed to the device and how it reports its results (REST API).

Estimated 2 pages for this section.

CHAPTER 5

Evaluation

The following sections contain a detailed evaluation of the model in various scenarios. First, we present metrics from the training phases of the constituent models. Second, we employ methods from the field of Explainable Artificial Intelligence (XAI) such as Gradient-weighted Class Activation Mapping (Grad-CAM) to get a better understanding of the models' abstractions. Finally, we turn to the models' aggregate performance on the test set.

5.1 Methodology

Go over the evaluation methodology by explaining the test datasets, where they come from, and how they're structured. Explain how the testing phase was done and which metrics are employed to compare the models to the SOTA.

Estimated 2 pages for this section.

5.2 Results

Systematically go over the results from the testing phase(s), show the plots and metrics, and explain what they contain.

Estimated 4 pages for this section.

5.2.1 Object Detection

The following parapraph should probably go into section 4.1.

The object detection model was pre-trained on the COCO [LMB$^+$15] dataset and fine-tuned with data from the OID [KRA$^+$20] in its sixth version. Since the full OID dataset contains considerably more classes and samples than would be feasibly trainable on a small cluster of GPUs, only images from the two classes *Plant* and *Houseplant* have been downloaded. The samples from the Houseplant class are merged into the Plant class because the distinction between the two is not necessary for our model. Furthermore, the OID contains not only bounding box annotations for object detection tasks, but also instance segmentations, classification labels and more. These are not needed for our purposes and are omitted as well. In total, the dataset consists of 91479 images with a roughly 85/5/10 split for training, validation and testing, respectively.

Test Phase

Of the 91479 images around 10% were used for the test phase. These images contain a total of 12238 ground truth labels. Table 5.1 shows precision, recall and the harmonic mean of both (F_1-score). The results indicate that the model errs on the side of sensitivity because recall is higher than precision. Although some detections are not labeled as plants in the dataset, if there is a labeled plant in the ground truth data, the chance is high that it will be detected. This behavior is in line with how the model's detections are handled in practice. The detections are drawn on the original image and the user is able to check the bounding boxes visually. If there are wrong detections, the user can ignore them and focus on the relevant ones instead. A higher recall will thus serve the user's needs better than a high precision.

	Precision	Recall	F_1-score	Support
Plant	0.547571	0.737866	0.628633	12238.0

Table 5.1: Precision, recall and F_1-score for the object detection model.

Figure 5.1 shows the AP for the IOU thresholds of 0.5 and 0.95. Predicted bounding boxes with an IOU of less than 0.5 are not taken into account for the precision and recall values of table 5.1. The lower the detection threshold, the more plants are detected. Conversely, a higher detection threshold leaves potential plants undetected. The precision-recall curves confirm this behavior because the area under the curve for the threshold of 0.5 is higher than for the threshold of 0.95 (0.66 versus 0.41). These values are combined in COCO's [LMB$^+$15] main evaluation metric which is the AP averaged across the IOU thresholds from 0.5 to 0.95 in 0.05 steps. This value is then averaged across all classes and called mAP. The object detection model achieves a state-of-the-art mAP of 0.5727 for the *Plant* class.

Hyperparameter Optimization

This section should be moved to the hyperparameter optimization section in the development chapter (section 4.1).

To further improve the object detection performance, we perform hyper-parameter optimization using a genetic algorithm. Evolution of the hyper-parameters starts from

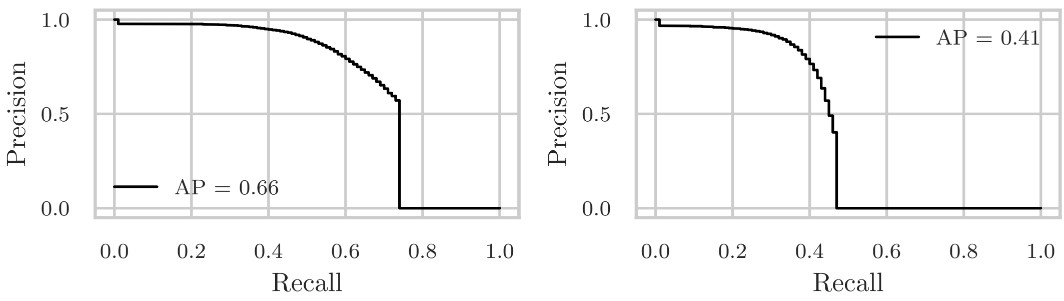

Figure 5.1: Precision-recall curves for IOU thresholds of 0.5 and 0.95. The AP of a specific threshold is defined as the area under the precision-recall curve of that threshold. The mAP across IOU thresholds from 0.5 to 0.95 in 0.05 steps `mAP@0.5:0.95` is 0.5727.

the initial 30 default values provided by the authors of YOLO. Of those 30 values, 26 are allowed to mutate. During each generation, there is an 80% chance that a mutation occurs with a variance of 0.04. To determine which generation should be the parent of the new mutation, all previous generations are ordered by fitness in decreasing order. At most five top generations are selected and one of them is chosen at random. Better generations have a higher chance of being selected as the selection is weighted by fitness. The parameters of that chosen generation are then mutated with the aforementioned probability and variance. Each generation is trained for three epochs and the fitness of the best epoch is recorded.

In total, we ran 87 iterations of which the 34^{th} generation provides the best fitness of 0.6076. Due to time constraints, it was not possible to train each generation for more epochs or to run more iterations in total. We assume that the performance of the first few epochs is a reasonable proxy for model performance overall. The optimized version of the object detection model is then trained for 70 epochs using the parameters of the 34^{th} generation.

Figure 5.2 shows the model's fitness during training for each epoch. After the highest fitness of 0.6172 at epoch 27, the performance quickly declines and shows that further training would likely not yield improved results. The model converges to its highest fitness much earlier than the non-optimized version, which indicates that the adjusted parameters provide a better starting point in general. Furthermore, the maximum fitness is 0.74% higher than in the non-optimized version.

Figure 5.3 shows precision and recall for the optimized model during training. Similarly to the non-optimized model from figure 4.2, both metrics do not change materially during training. Precision is slightly higher than in the non-optimized version and recall hovers at the same levels.

The box and object loss during training is pictured in figure 5.4. Both losses start from a lower level which suggests that the initial optimized parameters allow the model to converge quicker. The object loss exhibits a similar slope to the non-optimized model in

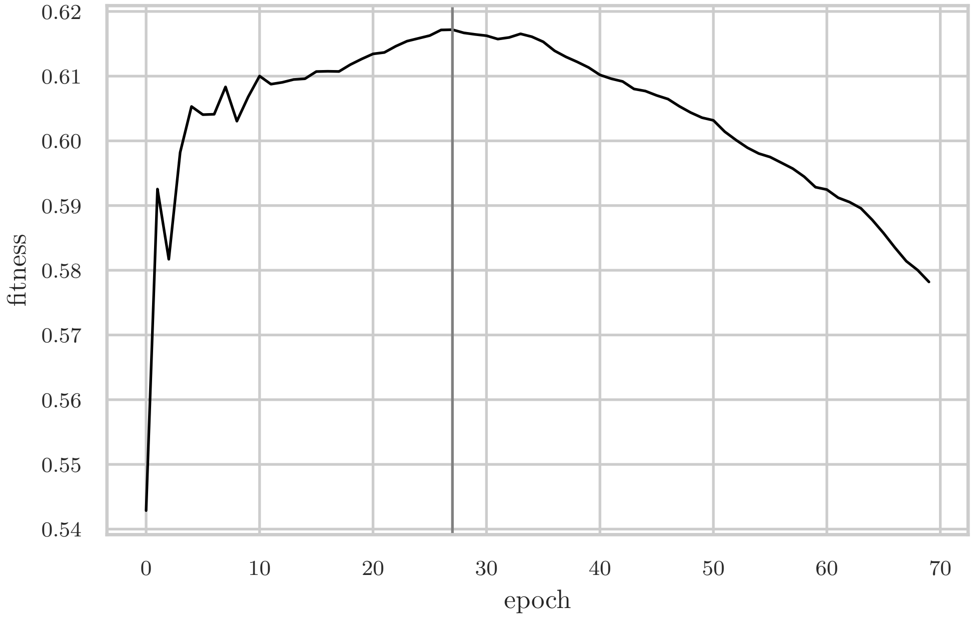

Figure 5.2: Object detection model fitness for each epoch calculated as in equation 4.1. The vertical gray line at 27 marks the epoch with the highest fitness of 0.6172.

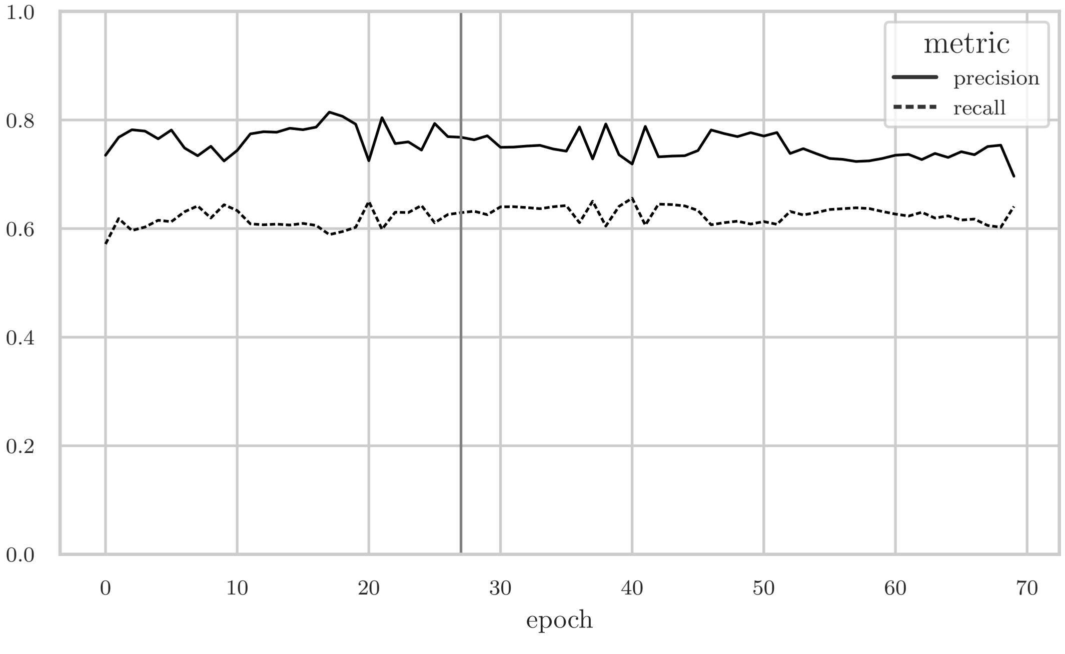

Figure 5.3: Overall precision and recall during training for each epoch of the optimized model. The vertical gray line at 27 marks the epoch with the highest fitness.

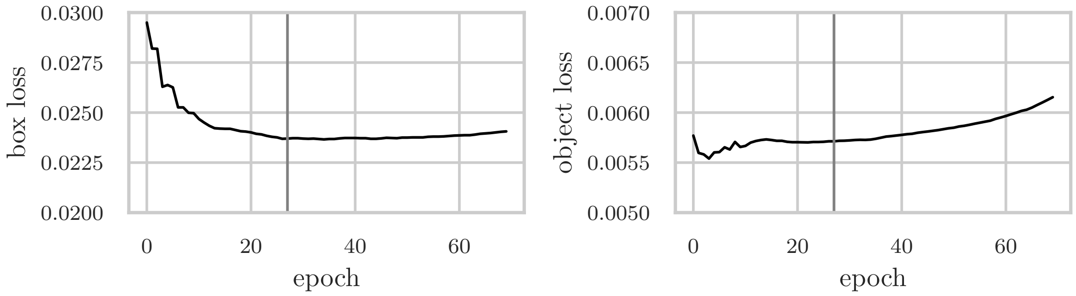

Figure 5.4: Box and object loss measured against the validation set of 3091 images and 4092 ground truth labels. The class loss is omitted because there is only one class in the dataset and the loss is therefore always zero.

figure 4.3. The vertical gray line again marks epoch 27 with the highest fitness. The box loss reaches its lower limit at that point and the object loss starts to increase again after epoch 27.

	Precision	Recall	F_1-score	Support
Plant	0.633358	0.702811	0.666279	12238.0

Table 5.2: Precision, recall and F_1-score for the optimized object detection model.

Turning to the evaluation of the optimized model on the test dataset, table 5.2 shows precision, recall and the F_1-score for the optimized model. Comparing these metrics with the non-optimized version from table 5.1, precision is significantly higher by more than 8.5%. Recall, however, is 3.5% lower. The F_1-score is higher by more than 3.7% which indicates that the optimized model is better overall despite the lower recall. We feel that the lower recall value is a suitable trade off for the substantially higher precision considering that the non-optimized model's precision is quite low at 0.55.

The precision-recall curves in figure 5.5 for the optimized model show that the model draws looser bounding boxes than the optimized model. The AP for both IOU thresholds of 0.5 and 0.95 is lower indicating worse performance. It is likely that more iterations during evolution would help increase the AP values as well. Even though the precision and recall values from table 5.2 are better, the mAP@0.5:0.95 is lower by 1.8%.

5.2.2 Classification

The classifier receives cutouts from the object detection model and determines whether the image shows a stressed plant or not. To achieve this goal, we trained a ResNet [HZR$^+$16] on a dataset of 452 images of healthy and 452 stressed plants. We chose the ResNet architecture due to its popularity and ease of implementation as well as its consistently high performance on various classification tasks. While its classification speed

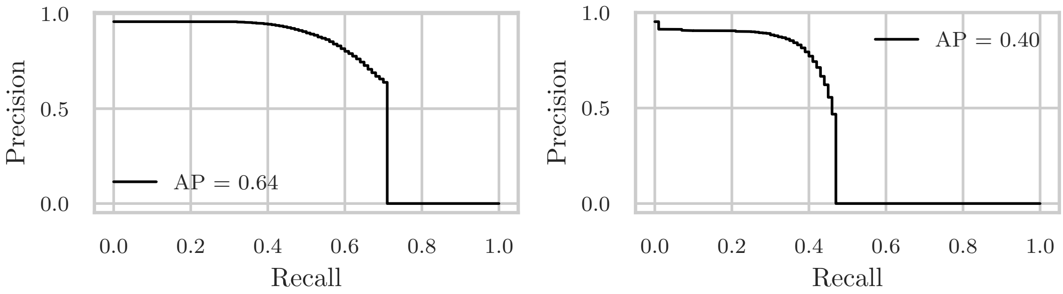

Figure 5.5: Precision-recall curves for IOU thresholds of 0.5 and 0.95. The AP of a specific threshold is defined as the area under the precision-recall curve of that threshold. The mAP across IOU thresholds from 0.5 to 0.95 in 0.05 steps `mAP@0.5:0.95` is 0.5546.

in comparison with networks optimized for mobile and edge devices (e.g. MobileNet) is significantly lower, the deeper structure and the additional parameters are necessary for the fairly complex task at hand. Furthermore, the generous time budget for object detection *and* classification allows for more accurate results at the expense of speed. The architecture allows for multiple different structures, depending on the amount of layers. The smallest one has 18 and the largest 152 layers with 34, 50 and 101 in-between. The larger networks have better accuracy in general, but come with trade-offs regarding training and inference time as well as required space. The 50 layer architecture (ResNet50) is adequate for our use case.

Hyperparameter Optimization

This section should be moved to the hyperparameter optimization section in the development chapter (section 4.2).

In order to improve the aforementioned accuracy values, we perform hyper-parameter optimization across a wide range of parameters. Table 5.3 lists the hyper-parameters and their possible values. Since the number of all combinations of values is 11520 and each combination is trained for 10 epochs with a training time of approximately six minutes per combination, exhausting the search space would take 48 days. Due to time limitations, we have chosen to not search exhaustively but to pick random combinations instead. Random search works surprisingly well—especially compared to grid search—in a number of domains, one of which is hyper-parameter optimization [BB12].

The random search was run for 138 iterations which equates to a 75% probability that the best solution lies within 1% of the theoretical maximum (5.1). Figure 5.6 shows three of the eight parameters and their impact on a high F_1-score. SGD has less variation in its results than Adam [KB17] and manages to provide eight out of the ten best results. The number of epochs to train for was chosen based on the observation that almost all configurations converge well before reaching the tenth epoch. The assumption that a

Parameter	Values
optimizer	adam, sgd
batch size	4, 8, 16, 32, 64
learning rate	0.0001, 0.0003, 0.001, 0.003, 0.01, 0.1
step size	2, 3, 5, 7
gamma	0.1, 0.5
beta one	0.9, 0.99
beta two	0.5, 0.9, 0.99, 0.999
eps	0.00000001, 0.1, 1

Table 5.3: Hyper-parameters and their possible values during optimization.

Optimizer	Batch Size	Learning Rate	Step Size
SGD	64	0.01	5

Table 5.4: Chosen hyper-parameters for the final, improved model. The difference to the parameters listed in Table 5.3 comes as a result of choosing SGD over Adam. The missing four parameters are only required for Adam and not SGD.

training run with ten epochs provides a good proxy for final performance is supported by the quick convergence of validation accuracy and loss in figure 4.4.

$$1 - (1 - 0.01)^{138} \approx 0.75 \tag{5.1}$$

Table 5.4 lists the final hyper-parameters which were chosen to train the improved model. In order to confirm that the model does not suffer from overfitting or is a product of chance due to a coincidentally advantageous train/test split, we perform stratified 10-fold cross validation on the dataset. Each fold contains 90% training and 10% test data and was trained for 25 epochs. Figure 5.7 shows the performance of the epoch with the highest F_1-score of each fold as measured against the test split. The mean ROC curve provides a robust metric for a classifier's performance because it averages out the variability of the evaluation. Each fold manages to achieve at least an AUC of 0.94, while the best fold reaches 0.98. The mean ROC has an AUC of 0.96 with a standard deviation of 0.02. These results indicate that the model is accurately predicting the correct class and is robust against variations in the training set.

The classifier shows good performance so far, but care has to be taken to not overfit the model to the training set. Comparing the F_1-score during training with the F_1-score during testing gives insight into when the model tries to increase its performance during training at the expense of generalizability. Figure 5.8 shows the F_1-scores of each epoch and fold. The classifier converges quickly to 1 for the training set at which point it experiences a slight drop in generalizability. Training the model for at most five epochs is

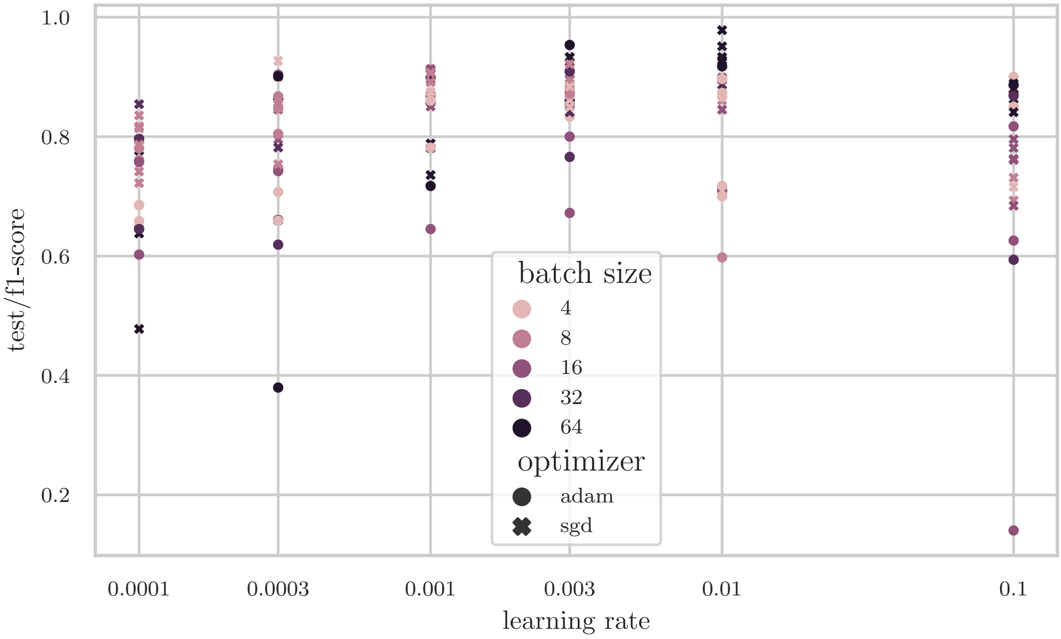

Figure 5.6: This figure shows three of the eight hyper-parameters and their performance measured by the F_1-score during 138 trials. Differently colored markers show the batch size with darker colors representing a larger batch size. The type of marker (circle or cross) shows which optimizer was used. The x-axis shows the learning rate on a logarithmic scale. In general, a learning rate between 0.003 and 0.01 results in more robust and better F_1-scores. Larger batch sizes more often lead to better performance as well. As for the type of optimizer, SGD produced the best iteration with an F_1-score of 0.9783. Adam tends to require more customization of its parameters than SGD to achieve good results.

sufficient because there are generally no improvements afterwards. The best-performing epoch for each fold is between the second and fourth epoch which is just before the model achieves an F_1-score of 1 on the training set.

Class Activation Maps

Neural networks are notorious for their black-box behavior, where it is possible to observe the inputs and the corresponding outputs, but the stage in-between stays hidden from view. Models are continuously developed and deployed to aid in human decision-making and sometimes supplant it. It is, therefore, crucial to obtain some amount of interpretability of what the model does *inside* to be able to explain why a decision was made in a certain way. The research field of XAI gained significance during the last few years because of the development of new methods to peek inside these black boxes.

One such method, Class Activation Mapping (CAM) [ZKL$^+$15], is a popular tool to produce visual explanations for decisions made by CNNs. Convolutional layers essentially

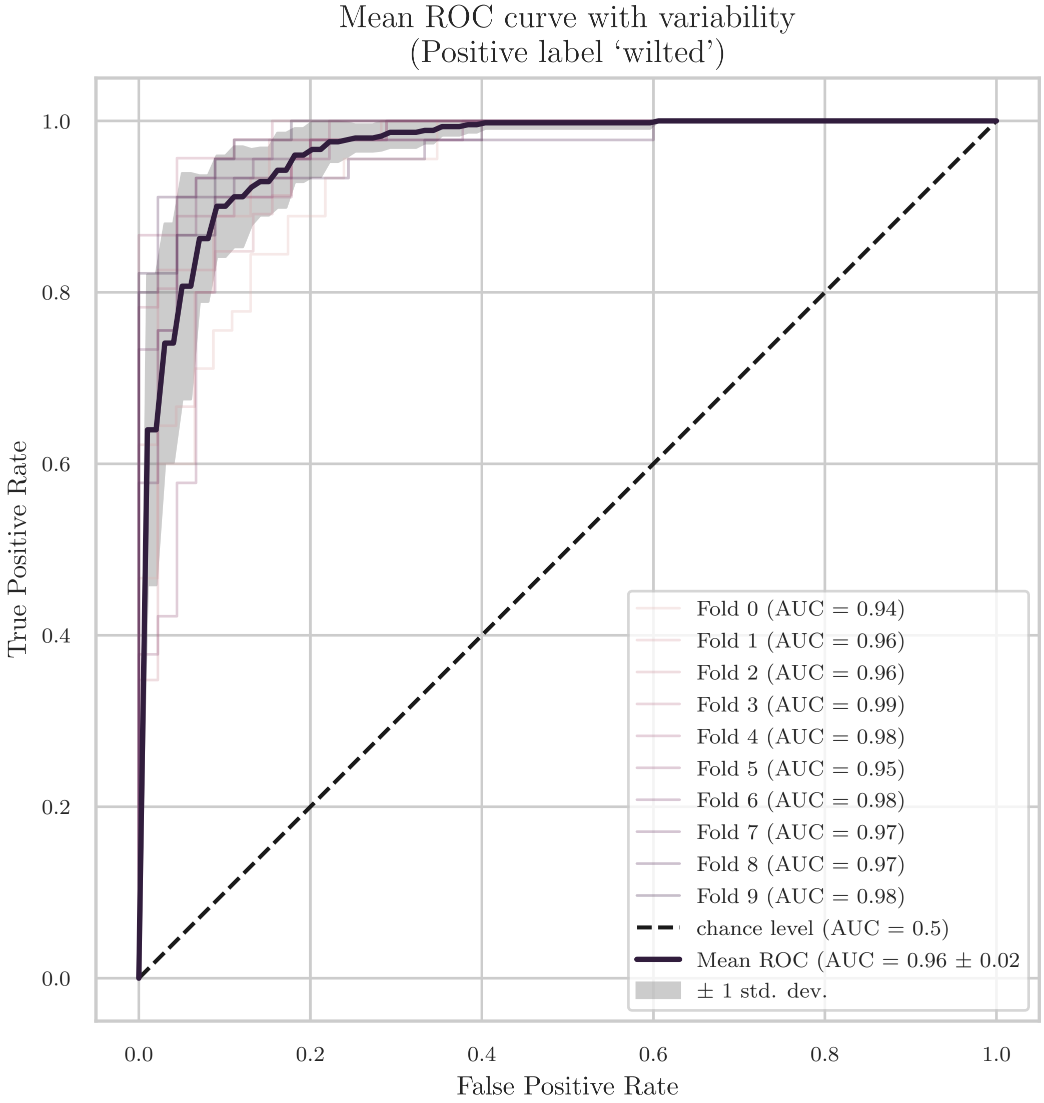

Figure 5.7: This plot shows the ROC curve for the epoch with the highest F_1-score of each fold as well as the AUC. To get a less variable performance metric of the classifier, the mean ROC curve is shown as a thick line and the variability is shown in gray. The overall mean AUC is 0.96 with a standard deviation of 0.02. The best-performing fold reaches an AUC of 0.99 and the worst an AUC of 0.94. The black dashed line indicates the performance of a classifier which picks classes at random (AUC = 0.5). The shapes of the ROC curves show that the classifier performs well and is robust against variations in the training set.

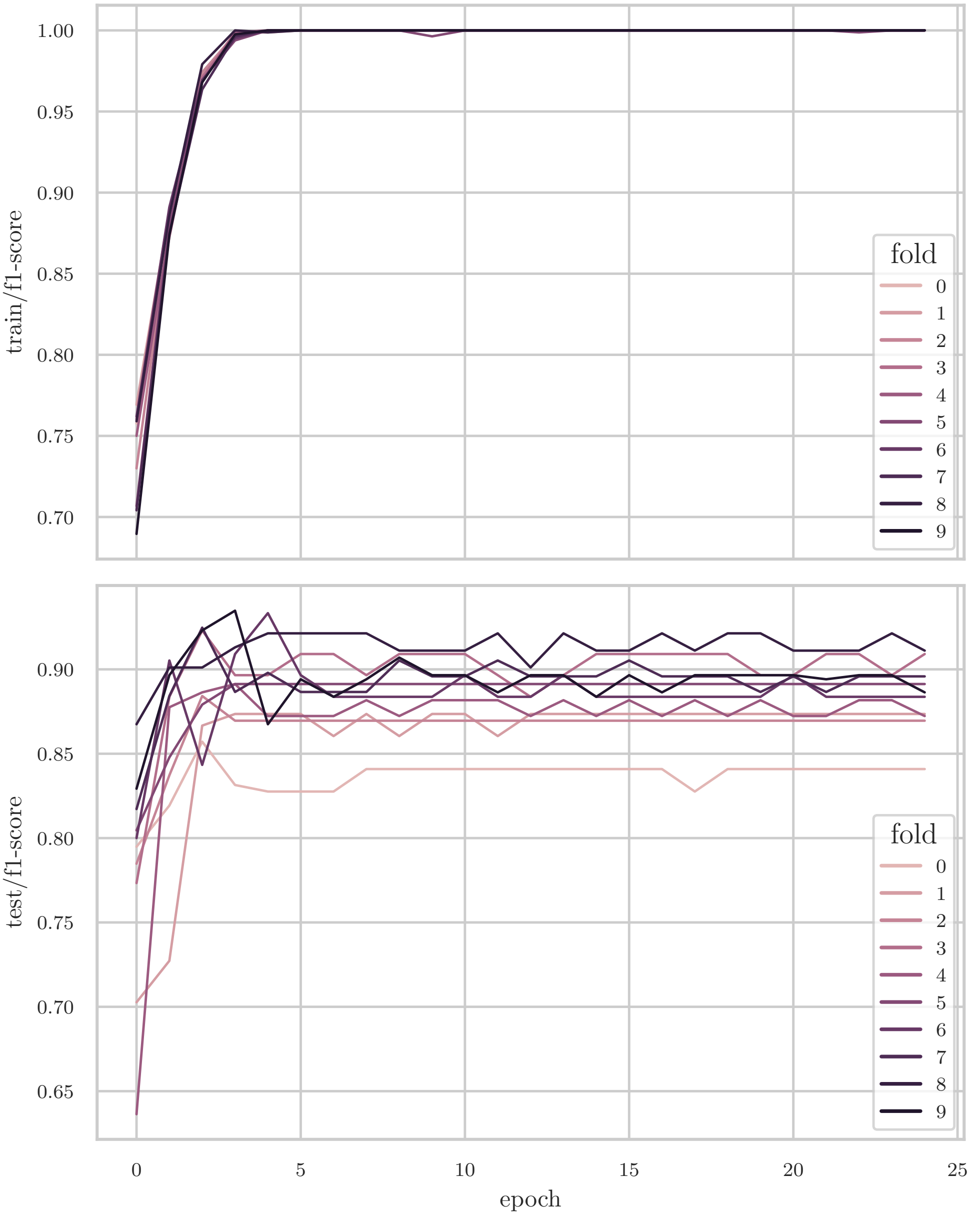

Figure 5.8: These plots show the F_1-score during training as well as testing for each of the folds. The classifier converges to 1 by the third epoch during the training phase, which might indicate overfitting. However, the performance during testing increases until epoch three in most cases and then stabilizes at approximately 2-3% lower than the best epoch. We believe that the third, or in some cases fourth, epoch is detrimental to performance and results in overfitting, because the model achieves an F_1-score of 1 for the training set, but that gain does not transfer to the test set. Early stopping during training alleviates this problem.

function as object detectors as long as no fully-connected layers perform the classification. This ability to localize regions of interest, which play a significant role in the type of class the model predicts, can be retained until the last layer and used to generate activation maps for the predictions.

A more recent approach to generating a CAM via gradients is proposed by Selvaraju et al. [SCD$^+$20]. Their Grad-CAM approach works by computing the gradient of the feature maps of the last convolutional layer with respect to the specified class. The last layer is chosen because the authors find that "[...] Grad-CAM maps become progressively worse as we move to earlier convolutional layers as they have smaller receptive fields and only focus on less semantic local features." [SCD$^+$20, p.5]

Turning to our classifier, figure 5.9 shows the CAMs for *healthy* and *stressed*. While the regions of interest for the *healthy* class lie on the healthy plant, the *stressed* plant is barely considered and mostly rendered as background information (blue). Conversely, when asked to explain the inputs to the *stressed* classification, the regions of interest predominantly stay on the thirsty as opposed to the healthy plant. In fact, the large hanging leaves play a significant role in determining the class the image belongs to. This is an additional data point confirming that the model focuses on the semantically meaningful parts of the image during classification.

5.2.3 Aggregate Model

In this section we turn to the evaluation of the aggregate model. We have confirmed the performance of the constituent models: the object detection and the classification model. It remains to evaluate the complete pipeline from gathering detections of potential plants in an image and forwarding them to the classifier to obtaining the results as either healthy or stressed with their associated confidence scores.

The test set contains 640 images which were obtained from a google search using the terms *thirsty plant*, *wilted plant* and *stressed plant*. Images which clearly show one or multiple plants with some amount of visible stress were added to the dataset. Care was taken to include plants with various degrees of stress and in various locations and lighting conditions. The search not only provided images of stressed plants, but also of healthy plants due to articles, which describe how to care for plants, having a banner image of healthy plants. The dataset is biased towards potted plants which are commonly put on display in western households. Furthermore, many plants, such as succulents, are sought after for home environments because of their ease of maintenance. Due to their inclusion in the dataset and how they exhibit water stress, the test set nevertheless contains a wide variety of scenarios.

After collecting the images, the aggregate model was run on them to obtain initial bounding boxes and classifications for ground truth labeling. Letting the model do the work beforehand and then correcting the labels allowed to include more images in the test set because they could be labeled more easily. Additionally, going over the detections and classifications provided a comprehensive view on how the models work and what

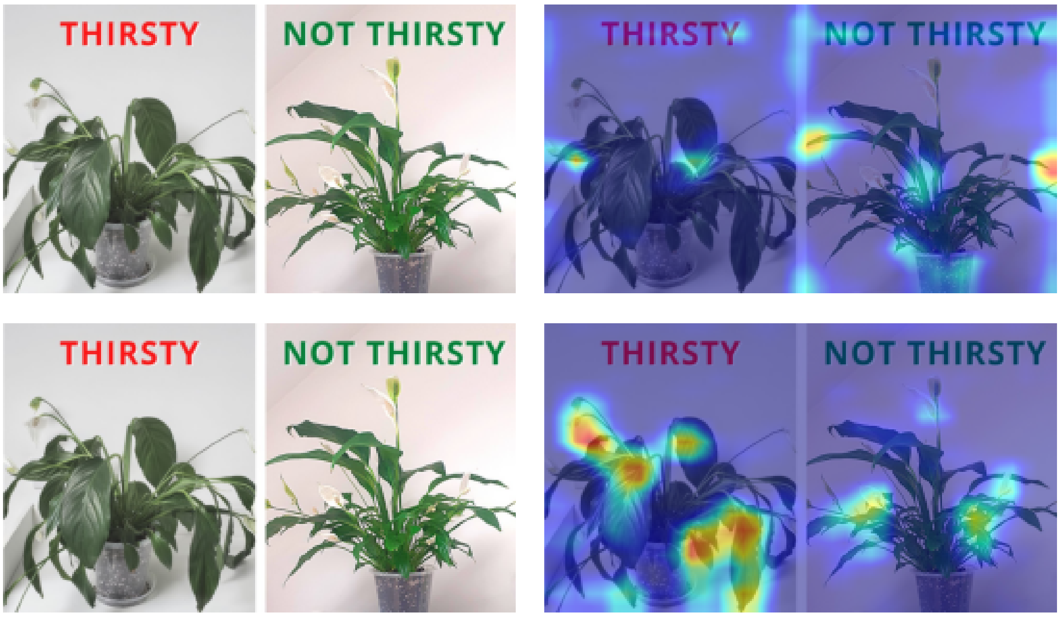

Figure 5.9: The top left image shows the original image of the same plant in a stressed (left) and healthy (right) state. In the top right image, the CAM for the class *healthy* is laid over the original image. The classifier draws its conclusion mainly from the healthy plant, which is indicated by the red hot spots around the tips of the plant. The bottom right image shows the CAM for the *stressed* class. The classifier focuses on the hanging leaves of the thirsty plant. The image was classified as *stressed* with a confidence of 70%.

	precision	recall	F_1-score	support
Healthy	0.665	0.554	0.604	766
Stressed	0.639	0.502	0.562	494
micro avg	0.655	0.533	0.588	1260
macro avg	0.652	0.528	0.583	1260
weighted avg	0.655	0.533	0.588	1260

Table 5.5: Precision, recall and F_1-score for the aggregate model.

their weaknesses and strengths are. After the labels have been corrected, the ground truth of the test set contains 766 bounding boxes of healthy plants and 494 of stressed plants.

5.2.4 Non-optimized Model

Table 5.5 shows precision, recall and the F_1-score for both classes *Healthy* and *Stressed*. Precision is higher than recall for both classes and the F_1-score is at 0.59. Unfortunately,

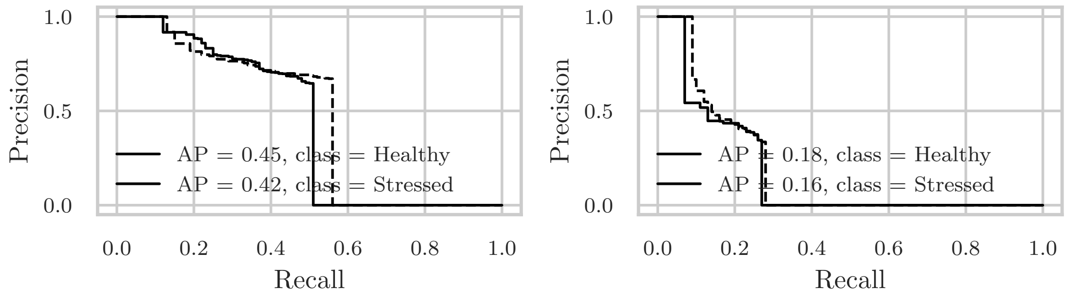

Figure 5.10: Precision-recall curves for IOU thresholds of 0.5 and 0.95. The AP of a specific threshold is defined as the area under the precision-recall curve of that threshold. The mAP across IOU thresholds from 0.5 to 0.95 in 0.05 steps `mAP@0.5:0.95` is 0.3581.

	precision	recall	F_1-score	support
Healthy	0.711	0.555	0.623	766
Stressed	0.570	0.623	0.596	494
micro avg	0.644	0.582	0.611	1260
macro avg	0.641	0.589	0.609	1260
weighted avg	0.656	0.582	0.612	1260

Table 5.6: Precision, recall and F_1-score for the optimized aggregate model.

these values do not take the accuracy of bounding boxes into account and thus have only limited expressive power.

Figure 5.10 shows the precision and recall curves for both classes at different IOU thresholds. The left plot shows the AP for each class at the threshold of 0.5 and the right one at 0.95. The mAP is 0.3581 and calculated across all classes as the median of the IOU thresholds from 0.5 to 0.95 in 0.05 steps. The cliffs at around 0.6 (left) and 0.3 (right) happen at a detection threshold of 0.5. The classifier's last layer is a softmax layer which necessarily transforms the input into a probability of showing either a healthy or stressed plant. If the probability of an image showing a healthy plant is below 0.5, it is no longer classified as healthy but as stressed. The threshold for discriminating the two classes lies at the 0.5 value and is therefore the cutoff for either class.

5.2.5 Optimized Model

So far the metrics shown in table 5.5 are obtained with the non-optimized versions of both the object detection and classification model. Hyper-parameter optimization of the classifier led to significant model improvements, while the object detector has improved precision but lower recall and slightly lower mAP values. To evaluate the final aggregate model which consists of the individual optimized models, we run the same test described in section 5.2.3.

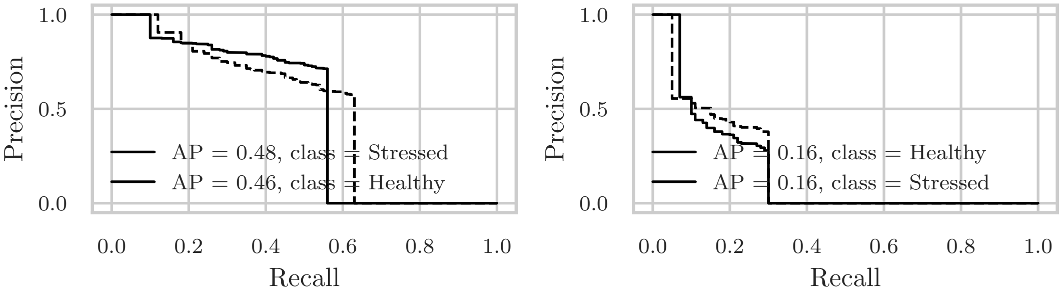

Figure 5.11: Precision-recall curves for IOU thresholds of 0.5 and 0.95. The AP of a specific threshold is defined as the area under the precision-recall curve of that threshold. The mAP across IOU thresholds from 0.5 to 0.95 in 0.05 steps `mAP@0.5:0.95` is 0.3838.

Table 5.6 shows precision, recall and F_1-score for the optimized model on the same test dataset of 640 images. All of the metrics are better for the optimized model. In particular, precision for the healthy class could be improved significantly while recall remains at the same level. This results in a better F_1-score for the healthy class. Precision for the stressed class is lower with the optimized model, but recall is significantly higher (0.502 vs. 0.623). The higher recall results in a 3% gain for the F_1-score in the stressed class. Overall, precision is the same but recall has improved significantly, which also results in a noticeable improvement for the average F_1-score across both classes.

Figure 5.11 confirms the performance increase of the optimized model established in table 5.6. The `mAP@0.5` is higher for both classes, indicating that the model better detects plants in general. The `mAP@0.95` is slightly lower for the healthy class, which means that the confidence for the healthy class is slightly lower compared to the non-optimized model. The result is that more plants are correctly detected and classified overall, but the confidence scores tend to be lower with the optimized model. The `mAP@0.5:0.95` could be improved by about 0.025.

5.3 Discussion

Pull out discussion parts from current results chapter (5.2) and add a section about achievement of the aim of the work discussed in motivation and problem statement section (1.2).

Estimated 2 pages for this chapter.

CHAPTER 6

Conclusion

Conclude the thesis with a short recap of the results and the discussion. Establish whether the research questions from section 1.2 can be answered successfully.

Estimated 2 pages for this chapter.

6.1 Future Work

Suggest further research directions regarding the approach. Give an outlook on further possibilities in this research field with respect to object detection and plant classification.

Estimated 1 page for this section

List of Figures

2.1 Structure of an artificial neural network 10

3.1 Methodological approach for the prototype. 36
3.2 Residual connection 42

4.1 Object detection fitness per epoch. 46
4.2 Object detection precision and recall during training. 47
4.3 Object detection box and object loss. 47
4.4 Classifier accuracy and loss during training. 48

5.1 Object detection AP@0.5 and AP@0.95. 51
5.2 Optimized object detection fitness per epoch. 52
5.3 Hyper-parameter optimized object detection precision and recall during training. 52
5.4 Hyper-parameter optimized object detection box and object loss. 53
5.5 Hyper-parameter optimized object detection AP@0.5 and AP@0.95. . . . 54
5.6 Classifier hyper-parameter optimization results. 56
5.7 Mean ROC and variability of hyper-parameter-optimized model. 57
5.8 F_1-score of stratified 10-fold cross validation. 58
5.9 Classifier CAMs. 60
5.10 Aggregate model AP@0.5 and AP@0.95. 61
5.11 Optimized aggregate model AP@0.5 and AP@0.95. 62

List of Tables

5.1 Precision, recall and F_1-score for the object detection model. 50
5.2 Precision, recall and F_1-score for the optimized object detection model. . 53
5.3 Hyper-parameters and their possible values during optimization. 55
5.4 Hyper-parameters for the optimized classifier. 55
5.5 Precision, recall and F_1-score for the aggregate model. 60
5.6 Precision, recall and F_1-score for the optimized aggregate model. 61

Acronyms

AI Artificial Intelligence. 8

ANN Artificial Neural Network. 23

AP Average Precision. 38, 39, 41, 42, 49, 50

API Application Programming Interface. 4

AUC Area Under the Curve. 2, 43, 45

CAM Class Activation Mapping. 44, 47, 48, 53

CNN Convolutional Neural Network. xv, xvi, 9, 17–24, 44

COCO Common Objects in Context. 4, 5, 20, 21

CPU Central Processing Unit. 18

CUDA Compute Unified Device Architecture. 23

DPM Deformable Part-Based Model. 16–18

ELU Exponential Linear Unit. 13

FPN Feature Pyramid Network. 19, 21

GPU Graphics Processing Unit. 2, 15, 17–19, 23, 24, 38

Grad-CAM Gradient-weighted Class Activation Mapping. 37, 47

HOG Histogram of Oriented Gradients. 16, 22

. 17

IOU Intersection over Union. 2, 18, 21, 38, 39, 41, 42, 49, 50

LRN Local Response Normalization. 24

mAP mean Average Precision. 2, 18–21, 38, 39, 42, 49, 50
MFCC Mel-frequency Cepstral Coefficient. 9
MLP Multilayer Perceptron. 11
MNIST Modified National Institute of Standards and Technology. 24
MSE mean squared error. 14, 15, 23, 24

OID Open Images Dataset. 34, 37, 38

RBF Radial Basis Function. 23
ReLU Rectified Linear Unit. 12, 13, 24
ResNet Residual Neural Network. 36, 41, 42
ROC Receiver Operating Characteristic. 2, 43, 45, 53
ROI Region of Interest. 17, 18
RPN Region Proposal Network. 19

SBC single-board computer. 2, 4
SGD Stochastic Gradient Descent. 23, 42–44
SIFT Scale-Invariant Feature Transform. 16, 22
SiLU Sigmoid Linear Unit. 13
SPP Spatial Pyramid Pooling. 18, 19
SSD Single Shot MultiBox Detector. 20, 21
SVM Support Vector Machine. 9, 16–18, 24

TPU Tensor Processing Unit. 2

VOC PASCAL Visual Object Classes. 4, 16, 18–21

XAI Explainable Artificial Intelligence. 37, 44

YOLO You Only Look Once. 20, 21

Bibliography

[ABC22] Lerina Aversano, Mario Luca Bernardi, and Marta Cimitile. Water stress classification using Convolutional Deep Neural Networks. *JUCS - Journal of Universal Computer Science*, 28(3):311–328, 3, March 28, 2022. DOI: `10.3897/jucs.80733`.

[AH22] Omar El Ariss and Kaoning Hu. ResNet-based Parkinson's Disease Classification. *IEEE Transactions on Artificial Intelligence*:1–11, 2022. DOI: `10.1109/TAI.2022.3193651`.

[AKG20] Shiva Azimi, Taranjit Kaur, and Tapan K Gandhi. Water Stress Identification in Chickpea Plant Shoot Images Using Deep Learning. In *2020 IEEE 17th India Council International Conference (INDICON)*. 2020 IEEE 17th India Council International Conference (INDICON), pages 1–7, December 2020. DOI: `10.1109/INDICON49873.2020.9342388`.

[ALL$^+$19] Jiangyong An, Wanyi Li, Maosong Li, Sanrong Cui, and Huanran Yue. Identification and Classification of Maize Drought Stress Using Deep Convolutional Neural Network. *Symmetry*, 11(2):256, February 2019. DOI: `10.3390/sym11020256`.

[Awa19] Mohamad M. Awad. Toward Precision in Crop Yield Estimation Using Remote Sensing and Optimization Techniques. *Agriculture*, 9(3):54, March 2019. DOI: `10.3390/agriculture9030054`.

[AWG21] Shiva Azimi, Rohan Wadhawan, and Tapan K. Gandhi. Intelligent Monitoring of Stress Induced by Water Deficiency in Plants Using Deep Learning. *IEEE Transactions on Instrumentation and Measurement*, 70:1–13, 2021. DOI: `10.1109/TIM.2021.3111994`.

[BB12] James Bergstra and Yoshua Bengio. Random Search for Hyper-Parameter Optimization. *The Journal of Machine Learning Research*, 13:281–305, null, February 1, 2012.

[BBL+23] Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, Difan Deng, and Marius Lindauer. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *WIREs Data Mining and Knowledge Discovery*, 13(2):e1484, 2023. DOI: `10.1002/widm.1484`.

[BMR+20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models Are Few-Shot Learners. July 22, 2020. DOI: `10.48550/arXiv.2005.14165`.

[BSF94] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, March 1994. DOI: `10.1109/72.279181`.

[BTD+21] Lefteris Benos, Aristotelis C. Tagarakis, Georgios Dolias, Remigio Berruto, Dimitrios Kateris, and Dionysis Bochtis. Machine Learning in Agriculture: A Comprehensive Updated Review. *Sensors*, 21(11):3758, January 2021. DOI: `10.3390/s21113758`.

[BWL20] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection. April 22, 2020. DOI: `10.48550/arXiv.2004.10934`. preprint.

[Cau47] M. Augustine Cauchy. Méthode génerale pour la résolution des systèmes d'équations simultanées. *Comptes rendus hebdomadaires des séances de l'Académie des sciences*, 25:399–402, October 18, 1847.

[CCR+21] Narendra Singh Chandel, Subir Kumar Chakraborty, Yogesh Anand Rajwade, Kumkum Dubey, Mukesh K. Tiwari, and Dilip Jat. Identifying Crop Water Stress Using Deep Learning Models. *Neural Computing and Applications*, 33(10):5353–5367, May 1, 2021. DOI: `10.1007/s00521-020-05325-4`.

[Dav92] A.M. Davis. Operational prototyping: a new development approach. *IEEE Software*, 9(5):70–78, September 1992. DOI: `10.1109/52.156899`.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009 IEEE Conference on Computer Vision and Pattern Recognition, pages 248–255, June 2009. DOI: `10.1109/CVPR.2009.5206848`.

[DT05] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), volume 1, 886–893 vol. 1, June 2005. DOI: `10.1109/CVPR.2005.177`.

[DZM$^+$21] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. RepVGG: Making VGG-style ConvNets Great Again. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 13728–13737, June 2021. DOI: `10.1109/CVPR46437.2021.01352`.

[EST$^+$14] Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov. Scalable Object Detection Using Deep Neural Networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014 IEEE Conference on Computer Vision and Pattern Recognition, pages 2155–2162, June 2014. DOI: `10.1109/CVPR.2014.276`.

[EVW$^+$10] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, 88(2):303–338, June 1, 2010. DOI: `10.1007/s11263-009-0275-4`.

[FGM$^+$10] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. Object Detection with Discriminatively Trained Part-Based Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, September 2010. DOI: `10.1109/TPAMI.2009.167`.

[FMR08] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 2008 IEEE Conference on Computer Vision and Pattern Recognition, pages 1–8, June 2008. DOI: `10.1109/CVPR.2008.4587597`.

[FS95] Yoav Freund and Robert E. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In Paul Vitányi, editor, *Computational Learning Theory*, Lecture Notes in Computer Science, pages 23–37, Berlin, Heidelberg. Springer, 1995. DOI: `10.1007/3-540-59119-2_166`.

[Fuk69] Kunihiko Fukushima. Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4):322–333, October 1969. DOI: `10.1109/TSSC.1969.300225`.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pages 249–256. JMLR Workshop and Conference Proceedings, March 31, 2010. URL: `https://proceedings.mlr.press/v9/glorot10a.html` (visited on 11/08/2023).

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, November 10, 2016. 801 pages.

[GDD$^+$14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014 IEEE Conference on Computer Vision and Pattern Recognition, pages 580–587, June 2014. DOI: `10.1109/CVPR.2014.81`.

[Gev22] Zhora Gevorgyan. SIoU Loss: More Powerful Learning for Bounding Box Regression. May 25, 2022. DOI: `10.48550/arXiv.2205.12740`. preprint.

[GFM] Ross B. Girshick, Pedro F. Felzenszwalb, and David McAllester. Discriminatively Trained Deformable Part Models (Release 5). URL: `https://www.rossgirshick.info/latent/` (visited on 10/26/2023).

[GID$^+$15] Ross Girshick, Forrest Iandola, Trevor Darrell, and Jitendra Malik. Deformable part models are convolutional neural networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 437–446, June 2015. DOI: `10.1109/CVPR.2015.7298641`.

[Gir15] Ross Girshick. Fast R-CNN. In *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015 IEEE International Conference on Computer Vision (ICCV), pages 1440–1448, December 2015. DOI: `10.1109/ICCV.2015.169`.

[HLV$^+$17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2261–2269, July 2017. DOI: `10.1109/CVPR.2017.243`.

[HSC$^+$19] Andrew Howard, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, Yukun Zhu, Ruoming Pang, Hartwig Adam, and Quoc Le. Searching for MobileNetV3. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pages 1314–1324, October 2019. DOI: `10.1109/ICCV.2019.00140`.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, January 1, 1989. DOI: `10.1016/0893-6080(89)90020-8`.

[HZC$^+$17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. April 16, 2017. DOI: `10.48550/arXiv.1704.04861`. preprint.

[HZR$^+$15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1904–1916, September 2015. DOI: `10.1109/TPAMI.2015.2389824`.

[HZR$^+$16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 770–778, June 2016. DOI: `10.1109/CVPR.2016.90`.

[IS15] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning*. International Conference on Machine Learning, pages 448–456. PMLR, June 1, 2015. URL: `https://proceedings.mlr.press/v37/ioffe15.html` (visited on 11/09/2023).

[Joc20] Glenn Jocher. YOLOv5 by Ultralytics, version 7.0, May 2020. DOI: `10.5281/zenodo.3908559`.

[KB17] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. January 29, 2017. DOI: `10.48550/arXiv.1412.6980`.

[KRA$^+$20] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, Tom Duerig, and Vittorio Ferrari. The Open Images Dataset V4: Unified Image Classification, Object Detection, and Visual Relationship Detection at Scale. *International Journal of Computer Vision*, 128(7):1956–1981, July 2020. DOI: `10.1007/s11263-020-01316-z`.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL: `https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html` (visited on 10/22/2023).

[LAE$^+$16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single Shot MultiBox Detector. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, Lecture Notes in Computer Science, pages 21–37, Cham. Springer International Publishing, 2016. DOI: `10.1007/978-3-319-46448-0_2`.

[LBB$^+$98] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998. DOI: `10.1109/5.726791`.

[LBD$^+$89] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, December 1, 1989. DOI: `10.1162/neco.1989.1.4.541`.

[LD15] Shuying Liu and Weihong Deng. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*. 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR), pages 730–734, November 2015. DOI: `10.1109/ACPR.2015.7486599`.

[LDG$^+$17] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 936–944, July 2017. DOI: `10.1109/CVPR.2017.106`.

[LGG$^+$17] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal Loss for Dense Object Detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017 IEEE International Conference on Computer Vision (ICCV), pages 2999–3007, October 2017. DOI: `10.1109/ICCV.2017.324`.

[LIM$^+$22] Patricia López-García, Diego Intrigliolo, Miguel A. Moreno, Alejandro Martínez-Moreno, José Fernando Ortega, Eva Pilar Pérez-Álvarez, and Rocío Ballesteros. Machine Learning-Based Processing of Multispectral and RGB UAV Imagery for the Multitemporal Monitoring of Vineyard Water Status. *Agronomy*, 12(9):2122, September 2022. DOI: `10.3390/agronomy12092122`.

[LLJ$^+$22] Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, Yiduo Li, Bo Zhang, Yufei Liang, Linyuan Zhou, Xiaoming Xu, Xiangxiang Chu, Xiaoming Wei, and Xiaolin Wei. YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications. September 7, 2022. DOI: `10.48550/arXiv.2209.02976`. preprint.

[LMB+15] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common Objects in Context. February 20, 2015. DOI: `10.48550/arXiv.1405.0312`.

[Low99] David G. Lowe. Object Recognition from Local Scale-Invariant Features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, page 1150, USA. IEEE Computer Society, September 20, 1999.

[LQQ+18] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path Aggregation Network for Instance Segmentation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 8759–8768, June 2018. DOI: `10.1109/CVPR.2018.00913`.

[Mis20] Diganta Misra. Mish: A Self Regularized Non-Monotonic Activation Function. August 13, 2020. DOI: `10.48550/arXiv.1908.08681`. preprint.

[Mit97] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., USA, 1st edition, February 1997. 432 pages.

[MP17] Marvin Minsky and Seymour A. Papert. *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, September 22, 2017. DOI: `10.7551/mitpress/11301.001.0001`.

[MP43] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, December 1, 1943. DOI: `10.1007/BF02478259`.

[MWL22] Patrick McEnroe, Shen Wang, and Madhusanka Liyanage. A Survey on the Convergence of Edge Computing and AI for UAVs: Opportunities and Challenges. *IEEE Internet of Things Journal*, 9(17):15435–15459, September 2022. DOI: `10.1109/JIOT.2022.3176400`.

[PDD+09] Nicolas Pinto, David Doukhan, James J. DiCarlo, and David D. Cox. A High-Throughput Screening Approach to Discovering Good Forms of Biologically Inspired Visual Representation. *PLOS Computational Biology*, 5(11):e1000579, November 26, 2009. DOI: `10.1371/journal.pcbi.1000579`.

[PY10] Sinno Jialin Pan and Qiang Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, October 2010. DOI: `10.1109/TKDE.2009.191`.

[RDG+16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 779–788, June 2016. DOI: `10.1109/CVPR.2016.91`.

[RF17] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 6517–6525, July 2017. DOI: `10.1109/CVPR.2017.690`.

[RF18] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. April 8, 2018. DOI: `10.48550/arXiv.1804.02767`. preprint.

[RHG+15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL: `https://proceedings.neurips.cc/paper/2015/hash/14bfa6bb14875e45bba028a21ed38046-Abstract.html` (visited on 10/27/2023).

[RHG+17] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, June 2017. DOI: `10.1109/TPAMI.2016.2577031`.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 6088, October 1986. DOI: `10.1038/323533a0`.

[Ros57] Frank Rosenblatt. The Perceptron: A Perceiving and Recognizing Automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, NY, January 1957.

[Ros62] Frank Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan Books, 1962. 648 pages.

[RRL+20] Paula Ramos-Giraldo, Chris Reberg-Horton, Anna M. Locke, Steven Mirsky, and Edgar Lobaton. Drought Stress Detection Using Low-Cost Computer Vision Systems and Machine Learning Techniques. *IT Professional*, 22(3):27–29, May 2020. DOI: `10.1109/MITP.2020.2986103`.

[RRM+20] Paula Ramos-Giraldo, S. Chris Reberg-Horton, Steven Mirsky, Edgar Lobaton, Anna M. Locke, Esleyther Henriquez, Ane Zuniga, and Artem Minin. Low-Cost Smart Camera System for Water Stress Detection in Crops. In *2020 IEEE SENSORS.* 2020 IEEE SENSORS, pages 1–4, October 2020. DOI: `10.1109/SENSORS47125.2020.9278744`.

[RTG+19] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 658–666, June 2019. DOI: `10.1109/CVPR.2019.00075`.

[SA15] David Sussillo and L. F. Abbott. Random Walk Initialization for Training Very Deep Feedforward Networks. February 27, 2015. DOI: `10.48550/arXiv.1412.6558`. preprint.

[Sam59] A. L. Samuel. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 3(3):210–229, July 1959. DOI: `10.1147/rd.33.0210`.

[SCD+20] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. *International Journal of Computer Vision*, 128(2):336–359, February 2020. DOI: `10.1007/s11263-019-01228-7`.

[SCL+20] Jinya Su, Matthew Coombes, Cunjia Liu, Yongchao Zhu, Xingyang Song, Shibo Fang, Lei Guo, and Wen-Hua Chen. Machine Learning-Based Crop Drought Mapping System by UAV Remote Sensing RGB Imagery. *Unmanned Systems*, 08(01):71–83, January 2020. DOI: `10.1142/S2301385020500053`.

[SGG16] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training Region-Based Object Detectors with Online Hard Example Mining. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 761–769, June 2016. DOI: `10.1109/CVPR.2016.89`.

[SGZ16] Falong Shen, Rui Gan, and Gang Zeng. Weighted residuals for very deep networks. In *2016 3rd International Conference on Systems and Informatics (ICSAI)*. 2016 3rd International Conference on Systems and Informatics (ICSAI), pages 936–941, November 2016. DOI: `10.1109/ICSAI.2016.7811085`.

[SHZ+18] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 4510–4520, June 2018. DOI: `10.1109/CVPR.2018.00474`.

[SJJ07] Prototyping Tools and Techniques. In Andrew Sears, Julie A. Jacko, and Julie A. Jacko, editors, *The Human-Computer Interaction Handbook*, pages 1043–1066. CRC Press, September 19, 2007. DOI: `10.1201/9781410615862-66`.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1–9, June 2015. DOI: `10.1109/CVPR.2015.7298594`.

[SSP03] P.Y. Simard, D. Steinkraus, and J.C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.* Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings. Pages 958–963, August 2003. DOI: `10.1109/ICDAR.2003.1227801`.

[SZ15] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. April 10, 2015. DOI: `10.48550/arXiv.1409.1556`. preprint.

[UvdSG+13] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective Search for Object Recognition. *International Journal of Computer Vision*, 104(2):154–171, September 1, 2013. DOI: `10.1007/s11263-013-0620-5`.

[VFH19] Maria Cecilia A. Venal, Arnel C. Fajardo, and Alexander A. Hernandez. Plant Stress Classification for Smart Agriculture utilizing Convolutional Neural Network - Support Vector Machine. In *2019 International Conference on ICT for Smart Society (ICISS).* 2019 International Conference on ICT for Smart Society (ICISS), volume 7, pages 1–5, November 2019. DOI: `10.1109/ICISS48059.2019.8969799`.

[VJ01] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001.* Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, volume 1, pages I–I, December 2001. DOI: `10.1109/CVPR.2001.990517`.

[WBL22] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. July 6, 2022. DOI: `10.48550/arXiv.2207.02696`. preprint.

[WLY22] Chien-Yao Wang, Hong-Yuan Mark Liao, and I.-Hau Yeh. Designing Network Design Strategies Through Gradient Path Analysis. November 9, 2022. DOI: `10.48550/arXiv.2211.04800`. preprint.

[WPL+18] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. CBAM: Convolutional Block Attention Module. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, Lecture Notes in Computer Science, pages 3–19, Cham. Springer International Publishing, 2018. DOI: `10.1007/978-3-030-01234-2_1`.

[ZF14] Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, Lecture Notes in Computer Science, pages 818–833, Cham. Springer International Publishing, 2014. DOI: `10.1007/978-3-319-10590-1_53`.

[ZHT22] Yiwei Zhong, Baojin Huang, and Chaowei Tang. Classification of Cassava Leaf Disease Based on a Non-Balanced Dataset Using Transformer-Embedded ResNet. *Agriculture*, 12(9):1360, September 2022. DOI: `10.3390/agriculture12091360`.

[ZKL+15] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning Deep Features for Discriminative Localization. December 13, 2015. DOI: `10.48550/arXiv.1512.04150`.

[ZQD+21] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A Comprehensive Survey on Transfer Learning. *Proceedings of the IEEE*, 109(1):43–76, January 2021. DOI: `10.1109/JPROC.2020.3004555`.

[ZWD+21] Haoyang Zhang, Ying Wang, Feras Dayoub, and Niko Sünderhauf. VarifocalNet: An IoU-aware Dense Object Detector. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 8510–8519, June 2021. DOI: `10.1109/CVPR46437.2021.00841`.

[ZWJ+17] Shuo Zhuang, Ping Wang, Boran Jiang, Maosong Li, and Zhihong Gong. Early Detection of Water Stress in Maize Based on Digital Images. *Computers and Electronics in Agriculture*, 140:461–468, August 1, 2017. DOI: `10.1016/j.compag.2017.06.022`.

[ZWL+20] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(07):12993–13000, 07, April 3, 2020. DOI: `10.1609/aaai.v34i07.6999`.