NOVA IMS Information Management School

MDSAA

Master Degree Program in

Data Science and Advanced Analytics

Transformers in Object Detection

A comparative analysis of the effectiveness of transformer-based neural networks in Object Detection tasks

Matias Marques Condessa de Sousa Neves

Dissertation

presented as partial requirement for obtaining the Master Degree Program in Data Science and Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

TRANSFORMERS IN OBJECT DETECTION

by

Matias Marques Condessa de Sousa Neves

Dissertation report presented as partial requirement for obtaining the master's degree in advanced Analytics, with a Specialization in Data Science

Supervisor: Prof. Mauro Castelli

Co Supervisor*:* Victor Costa

11 2023

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledge the Rules of Conduct and Code of Honor from the NOVA Information Management School.

Lisbon, 29th of November 2023

ACKNOWLEDGEMENTS (OPTIONAL)

I thank my parents, first and foremost, for always supporting me, emotionally and financially. Of course, I thank the rest of my family as well, and the friends I've made along the way, with whom I laughed, cried, and learned.

I also thank my advisers, Professor Mauro Castelli and Victor Costa, for helping and supporting me with the research and writing of this thesis.

Last but definitely not least, I'm thanking my dog, for getting me out of the house whenever we both needed to, and because she's very nice.

ABSTRACT

Artificial Intelligence is applicable to many different tasks, one of those tasks being Object Detection. For this task, the most common Machine Learning models are Artificial Neural Networks. While many Artificial Neural Networks are experimented with, Convolutional Neural Networks are currently the most common and most developed models that can be used in Object Detection. However, Artificial Intelligence models that interpret data using different sets of algorithms are still consistently researched, namely Transformer-based models. In this thesis, Artificial Intelligence models that integrate Transformer architectures are compared with a state-of-the-art Convolutional Neural Network, the comparisons being based on three Object Detection tasks: Identifying blueberry batches in images, identifying balloons in images, and identifying objects represented in the COCO2017 dataset. The first task is the most extensively researched one: it's a real-world Precision Agriculture task, and four datasets were annotated from existing footage to train the Object Detection models to perform this task. This task involves a great deal of experimental work, so the other two tasks serve as more stable benchmarks: the second task involves an open-source dataset, and for the third task comparisons were made between already existing results of training these models. Models are compared based on their predictive power and their training time. This thesis aims to determine whether Transformer-based Artificial Intelligence models can feasibly compete with state-of-the-art solution in Object Detection tasks, and, consequently, whether further research on the use of Transformers for Object Detection can be expected to be fruitful in a practical sense.

KEYWORDS

Transformers; Object Detection; Neural Networks; Artificial Intelligence; Precision Agriculture

Sustainable Development Goals (SGD):

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

CNN Convolutional Neural Network: a neural network architecture.

DETR Detection Transformer: a deep learning model used for object detection; the model incorporates a transformer neural network.

YOLOv8 You Only Look Once, version 8: a state-of-the-art deep learning model used for object detection.

IoU Intersection over Unity: number between 0 and 1 assigned to each prediction, that indicates the percentage of overlap between predictions and ground truths

AP Average precision: a metric based on the precision-recall curve, used commonly in object detection.

mAP Mean Average Precision: mean of the average precisions of each class

mAP0.5 mAP, considering as correct predictions any predictions with an IoU over 0,5

mAP0.5:0.95 Average value of mAP scores with the following lower threshold for IoU: 0,5; 0,55; 0,6; 0,65; 0,7; 0,75; 0,8; 0,85; 0,9; 0,95

ANN Artificial Neural Network: reproduction of a biological Neural Network that uses mathematical functions and computational principles to process information

BB-1 Dataset composed of 3840x2160 images of a blueberry plantation, and slightly edited versions of said images, which was used to train Object Detection models to detect blueberry batches

BB-2 Dataset composed of 3840x2160 images of a blueberry plantation that was used to train Object Detection models to detect blueberry batches

BB-a Dataset composed of 960x540 images of a blueberry plantation, and slightly edited versions of said images, which was used to train Object Detection models to detect blueberry batches

BB-b Dataset composed of 960x540 images of a blueberry plantation that was used to train Object Detection models to detect blueberry batches

BLN Dataset composed of pictures with balloons that was used to train Object Detection models to detect balloons

DETR-r50 Detection Transformer with a ResNet50 backbone for feature extraction

DETR-r101 Detection Transformer with a ResNet101 backbone for feature extraction

1. INTRODUCTION

The identification and recognition of objects, visually or by any other means, is a cognitive capability of most animals. The prospect of machines that can learn how to identify objects in any way garnered great interest from researchers and spurred the concept of Computer Vision. Computer Vision has been one of the most important research areas in Artificial Intelligence, due to its versatility and potential when it comes to its application to real-world problems. Currently, the state-of-the-art Artificial Intelligence models applied in Computer Vision integrate and primarily rely on Convolutional Neural Networks, using convolutions to properly interpret images.

Recently, the application of Transformer-based Neural Networks to Computer Vision problems has been tested in multiple scenarios. The latest Transformers have achieved results comparable to those obtained by CNNs on Computer Vision tasks but require large amounts of data to achieve said performances (S. Khan et al., 2022). However, this is a relatively new area of application for Transformer-based models, and as such, Vision-based Transformers are yet to be further tested, and optimized, to the degree that current state-of-the-art models are.

This thesis has as its main research goal comparing the effectiveness of a Computer Vision Transformer — Detection Transformer (DETR) — and a cutting-edge CNN — You Only Look Once (YOLOv8) — on Object Detection tasks. The effectiveness of these models will be evaluated based on their performance, judged by their average precision (Padilla et al., 2020), runtime, and computational cost. Two variants of DETR — DETR-r50 and DETR-r101 — will be evaluated. These variants only differ in their backbones, which are used for feature extraction.

After analysing their performance on the COCO2017 dataset, both variants of DETR and YOLOv8 are trained on five datasets and evaluated on their predictive power and runtime. One of the datasets is the Balloon dataset, and it's both publicly accessible and balanced. The other four datasets come from a real-world problem: the detection of batches of blueberries in footage of a blueberry plantation. This problem is common in Precision Agriculture and not exclusive to blueberries, and most of the research done ultimately serves to produce the best possible solution to this problem. All blueberry batch datasets consist of manually annotated images of a blueberry plantation extracted from drone footage acquired by a private company (that requested to remain anonymous) of a blueberry plantation.

In the second chapter, the theoretical foundations of the realized work will be provided: first, explanations on Neural Networks, with a bigger emphasis on Transformers, followed by a section on Object Detection, and the two models that will be compared, ending with some insights related to the real-world case study: the detection of blueberry batches in a blueberry plantation. The third chapter will be dedicated to the processes and methodology of the research done, including the annotation of the blueberry plantation footage and the setups of the two models. The fourth chapter will present and discuss the results of this outcome. The fifth chapter will present the conclusions taken from the research presented thus far, and the sixth chapter discusses possible paths in the future.

2. LITERATURE REVIEW

In this chapter, several concepts vital to this thesis will be introduced and elaborated upon. After a brief introduction to computer vision, there is a general description of the field of Object Detection tasks, followed by two detailed sub-sections: one on metrics and forms of measurement used, and one on data imbalance issues common to the field. Afterwards, Neural Networks are described, with special attention to networks used in Object Detection or other Computer Vision problems, along with an overview of Transfer Learning. Then, a section is dedicated to Transformer architectures, with some emphasis on DETR, and a final section consisting of a general exposé on Precision Agriculture and related work on the topic.

First, a small introduction on Computer Vision: as a study and research field dedicated to giving machines the ability to "see", Computer Vision aims to allow computers to identify and interpret visual inputs, static or in video form, much like a human would. This field has several increasingly sought-after applications: facial recognition, interpretation of visual feedback from medical procedures, crop monitoring, falsification detection, and so forth (A. I. Khan & Al-Habsi, 2020).

Through Computer Vision, the automation of monitoring, inspection, and surveillance of images (Bond et al., 2019) is made possible. Machine Learning has led to the development of many computer vision systems capable of performing different visual tasks. In these systems, the computer vision component analyses the image and represents it as a multi-dimensional array of pixels, and Machine Learning models process the image's representation, and try to identify patterns in the arrays of pixels corresponding to images. These systems are capable of accurately indicating the contents of images, automatically classifying images (A. I. Khan & Al-Habsi, 2020), and even generating images (Gregor et al., 2015).

2.1. OBJECT DETECTION

A fundamental task in Computer Vision is the use of computer applications to identify objects in images, known as Object Detection. In object detection, the finality is to find all instances of one or more kinds of objects, such as cars, trees, people, and so on. Presently, development on Object Detection is very advanced, with a great number of tasks being performed by Deep Learning models (J. Wang et al., 2021) (C. Wang et al., 2022) (Koyun et al., 2022).

The goal of object detection is to detect all instances of objects from one or several known classes, such as people, cars, or faces in an image. Usually, only a small number of objects are present in the image, but objects from one class can appear in different shapes, sizes, positions, and even colors, within one image. Objects can be represented by a location, a location and scale, a bounding box, or even a segmentation mask.

Systems made for object detection identify objects belonging to a specified class by learning characteristics common to this class's objects. Generally, it is necessary to have hundreds or thousands of examples of an object class to capture some aspects of class variability. If said class variability can be explicitly built into the model, less training data is needed, but it is usually not feasible to try to specify this class variability in the model, due to it being very high in most cases. Therefore, deep learning models can learn class variability when trained with large datasets (Amit et al., 2020).

There are two kinds of object detection based on deep learning: one-stage detection and two-stage detection. Generally, one-stage detection directly divides the image into $n \times n$ grids and computes predictions for each grid. Sometimes, only one grid is used: all objects in an image are inferred at once (Carion et al., 2020). In one-stage detection, predictions are made concerning a grid of possible object centers (Z. Tian et al., 2019), or anchors (Lin et al., 2017). Meanwhile, two-stage detection models usually generate regional proposals and map them on the feature map first, and then perform prediction calculations on the region of interest (RoI) region (Wen et al., 2023).

The performance of these Object Detection systems ultimately depends on the way their initial guesses are set. Therefore, said systems require some management of initial guesses to achieve their best possible results. Recent object detectors have strived to circumvent this limitation, with technologies such as directly predicting the set of detections with absolute box prediction concerning the input image (Carion et al., 2020).

Usually, object detection methods are either generative (Amit et al., 2002) (Felzenszwalb et al., 2005) or discriminative (Rowley et al., 1998) (Felzenszwalb et al., 2010). Generative methods model two distributions: one for an object's shape, and one for the image features in a devised window, with this second distribution being conditional on the shape. Afterwards, the probability of the window containing an object is compared with the probability of the window only containing the object's background. Discriminative models build classifiers that attempt to differentiate between images, or sections of images, containing instances of the target class, and images/sections not containing them (Amit et al., 2020). In this thesis, the focus will be on discriminative models.

2.1.1. Metrics

The effectiveness of an object detection model can be evaluated with multiple different forms of measurement metrics. The performance metric most used to qualify these models is the Average Precision (AP). To explain AP, it is necessary to know the concept of Intersection over Unity (IoU), as well as two metrics commonly used in machine learning — precision and recall.

The IoU is a coefficient of similarity between two sets of data, based on the Jaccard index (Jaccard, 1901). In the context of object detection, the IoU measures the overlapping and the total area of a prediction and a ground truth. Being O_p the predicted object, and O_{gt} the ground truth object, the IoU is calculated with the following formula (Padilla et al., 2020): $IoU = \frac{area(O_p \cap O_{gt})}{area(O_p \cup O_{gt})}$.

To calculate the precision and recall, it is necessary to know the amounts of True Positives (TPs), False Positives (FPs), and False Negatives (FNs). This is how TPs, FPs, and FNs are determined in Object Detection:

- TP: A correct prediction of a ground-truth object.
- FP: An incorrect detection of a non-existent object or a misplaced detection of an existing object.
- FN: An undetected ground-truth object.

A single prediction can only be classified as a true positive if the prediction and a ground truth have an IoU above the threshold, or a false positive if otherwise. In case multiple predictions overlap the same ground truth, the one with the highest confidence is classified as a true positive, and the others as false

positives. The precision and recall of a given data item are determined by the number of available predictions, which changes with the model's confidence threshold.

The AP is based on the precision-recall curve: this curve represents the model's different possible precision values, for different confidence levels, as the response variable of the model's possible recalls. A high area under the precision-recall curve indicates that both precision and recall are, on average, high for the model's predictions. However, in many real-world cases, the precision-recall curve has a zig-zag-like pattern, which makes the measurement of the area under the curve inaccurate usually. Therefore, the precision-recall curve is interpolated, and for every recall value R, the corresponding adjusted precision value is the highest observed precision for a recall greater than or equal to R (Padilla et al., 2020). This interpolation can be made over a fixed number of possible recall values or all recall values.

With $P_{interp}(R_{n+1})$ being the interpolated value of the precision at R_{n+1}, with the interpolation made over all recall values, the formula for average precision is the following (Padilla et al., 2020): $AP_{all} = \sum_n (R_{n+1} - R_n) P_{interp}(R_{n+1})$.

For situations with more than one object class, the Mean Average Precision (mAP), which is the mean of the classes' respective AP values, is used. Additionally, rather than using just one AP/mAP value, associated with one IoU threshold, it is very common to calculate AP/mAP for multiple IoU thresholds — from 50% to 95% with steps of 5%, to be precise — and use the mean of those different AP/mAP values, commonly noted $mAP0.5{:}\,0.95$. Finally, Average Recall (AR) or Mean Average Recall (mAR) may also be used; these metrics are obtained via the same procedures as AP/mAP, but with the precision-recall curve now portraying recall as the response variable of precision.

2.1.2. Imbalance in Object Detection

A known challenge associated to training Machine Learning models is data imbalance, and training for Object Detection tasks is no exception. A dataset is said to be imbalanced when some aspect of it is disproportionately represented. In object detection, this can mean multiple different things, and in a 2019 survey (Oksuz et al., 2019), imbalance problems were grouped into four main variations: class imbalance, scale imbalance, spatial imbalance, and objective imbalance. These designations will be adopted in this paper.

Class imbalance is the most common kind of imbalance: each data object is associated with a certain class, and sometimes, one or more classes may be significantly less (or more) represented than others. In object detection, this can mean one of two things:

- The total area occupied by important objects in an image is relatively low compared to the image's total area: foreground-background imbalance (H. Gu et al., 2023).
- There are significantly fewer/more important objects of one or more classes than important objects of other classes: foreground-foreground imbalance. This is ruled out in object detection problems featuring only one class.

The former is very common in object detection data since it can be difficult to capture images that can present certain objects in a size proportionate to an image. The latter is the more familiar example when discussing data imbalance outside of object detection. Additionally, the former is significantly

harder to solve and may require major data overhauls, but the latter can be mitigated, or completely fixed, in the data's "sampling" stage.

Scale imbalance occurs when object instances have different sizes, and when some sizes have more representation than others. This is natural and common with objects in nature. This may also result in feature imbalance, as contributions from a model's layers in the feature extraction phase can be uneven due to the disparities in sizes. Multiple methods have been proposed to solve this issue, tackling it either from the object-level imbalance viewpoint (Singh et al., 2018) (Lin et al., 2017), or from a feature level imbalance viewpoint (Liu et al., 2018) (Ghiasi et al., 2019).

Spatial imbalance can be divided into three categories: imbalance in regression loss, IoU distribution imbalance, and object location imbalance. These categories are influenced by factors related to the spatial properties of bounding boxes. Imbalance in regression loss is related to the loss function, as it pertains to the impact individual examples have on regression loss; IoU distribution imbalance is usually caused in the annotation of objects in training data and pertains to IoU distribution biases; Object location imbalance is related to the annotation anchors of objects and potentially to the training images themselves, and pertains to how object instances are positioned in an image, and occurs when certain parts of the image contain a lot of objects whilst other parts contain very little (Oksuz et al., 2019).

Objective imbalance is often an issue with how the model is configured, as it occurs when there are multiple objective/loss functions, each one being typically meant for a different task. These functions may be incompatible when it comes to their ranges and optimal solutions. To remedy this, it's vital to either disregard some of the functions — and therefore, the task that uses them as a metric — or to re-balance them so that there can be a solution that accomplishes all objectives (Oksuz et al., 2019).

2.2. NEURAL NETWORKS

A Neural Network is a group of neurons that are connected, or associated when it comes to their function. Deep Learning, a subset of Machine Learning, is defined as the use of Neural Networks in Machine Learning. Artificial Neural Networks (ANNs) attempt to reproduce the function of the human brain by mimicking its structure. In Deep Learning models, data flows through different neurons organized in layers and the training process consists in determining the weight of the connections between neurons. Much like a living being, an ANN has as its objective the memorization and subsequent recognition of certain objects, or patterns (Zhang et al., 2021).

There are many types of neural networks, but the basic principles are very similar:

- Each neuron in the network can receive input signals, process them, and send an output signal.
- Every neuron is connected at least with one other neuron, and each connection is evaluated by a real number: the weight coefficient, which reflects the degree of importance of the given connection in the neural network.
- The weight coefficients change due to the influence of a measure of the network's performance.
- The network, in its entirety, is composed of at least three layers of neurons: one input layer, one output layer, and multiple hidden layers with varying attributes and tasks.

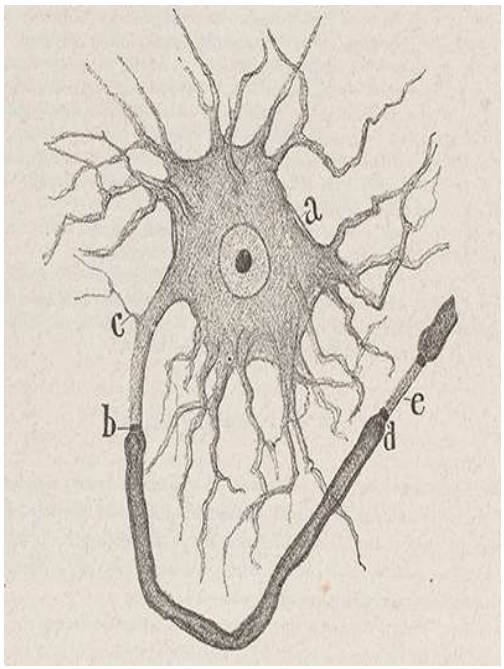

Figure 1 – Drawing of a neuron schematic proposed by Santiago Ramón y Cajal (Cajal, 1899-1904). Caption: a) protoplasm's membrane; b) connecting disk; c) collateral nervous branch; d) connecting disk of a Ranvier node; e) axon section without myelin.

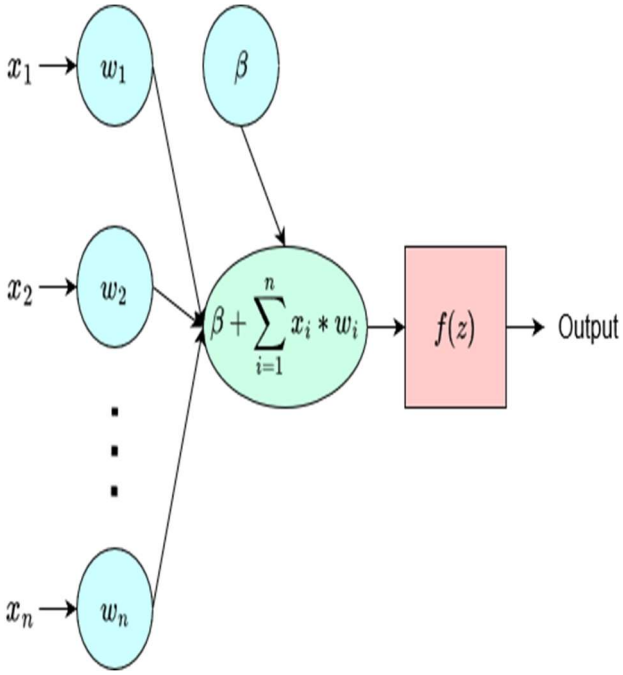

Figure 2 – Representation of a neuron in a Neural Network: $\boldsymbol{x}$ – data object, represented as a vector $\boldsymbol{x} = [\boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_n}]$; $\boldsymbol{n}$ = number of features in an object $\boldsymbol{x}$; $\boldsymbol{w}$ — weights vector, $\boldsymbol{w} = [\boldsymbol{w_1}, \boldsymbol{w_2}, \dots, \boldsymbol{w_n}]$; β — the bias associated to the neuron ; $\boldsymbol{f}$ – activation function.

Though ANNs have been researched for over 40 years, as we see in (Hopfield, 1982) for example, they were impractical for real-world applications because they usually require vast amounts of data to achieve proper results. However, in problems where vast amounts of data can be collected, ANNs have been more and more favoured as applications to solve or find solutions to these problems in recent times.

Though they are more potent, state-of-the-art ANNs are generally more computationally expensive than other Machine Learning models, as they have many layers, and therefore, many steps, to them. Therefore, it's required that these networks are trained in higher-end hardware, which typically implies cloud computing.

Additionally, deep learning models are complemented by optimization algorithms. These algorithms minimize the training error of a model, by minimizing the model's loss function. The performance of the optimization algorithm directly affects the model's training efficiency. To improve the performance of deep learning models, it is possible to tune the hyperparameters of optimization. However, that implies understanding the principles of different optimization algorithms and their hyperparameters. Some optimization algorithms more commonly applied to Deep Learning models are Stochastic Gradient Descent — SGD (Ketkar et al., 2017) — and Adam (Kingma et al., 2017).

2.2.1. Feed-forward Networks

The most popular neural networks are Feed-Forward networks: they take data inputs and pass them through multiple transformative layers, ultimately producing an output signal. Feed-Forward Networks can come in many forms and sizes and are used in several tasks, not just as predictors, but also to transform data and perform other supportive tasks, when necessary.

The most basic form of a feed-forward network is the single-node perceptron. A perceptron is a probabilistic model originally conceived to understand brain activity (Rosenblatt, 1958). The single node perceptron receives as input a set of vectors, each vector representing a data point, and the class to which they belong (Vanneschi & Castelli, 2019). There are specific weights attributed to each feature of the input, which are used to determine the node's output. The set of operations performed within the node can be represented by this expression: $\Phi_n(x) = h(w_n \cdot x + d_n)$, with x being the input (with all features) to the node, w_n being the node's set of weights, d_n being the node's only independent term — the bias — and h being an activation function, introduced to prevent linearity (Collobert & Bengio, 2004).

The weights and the bias on each hidden layer node are initially randomized, and they are changed in each training cycle according to a function involving the classification errors of the last completed cycle, to properly classify all data entries; this is known as backpropagation. Training stops either when one or more performance thresholds are hit — this can be determined by common metrics such as accuracy or precision — or when a set limit of training cycles is reached.

Feed-forward Networks usually consist of multiple perceptron nodes, organized in multiple layers: this kind of network is known as the Multilayer Perceptron, or MLP for short. An MLP is fully connected, i.e., every node in one layer is connected to every node in the next layer, and every node in the previous layer. It has one input layer, one output layer, and at least one hidden layer. Each node in an MLP acts as a single-node perceptron, and the set of outputs from all nodes in a layer constitutes the input of each node in the next layer. Just as with single-node perceptrons, the error is determined and backpropagated after each training cycle, and training only stops after a fixed number of iterations, or when the model's performance crosses a certain threshold.

2.2.2. Convolutional Networks

Nowadays, image data is usually represented as a two-dimensional grid of pixels, whether the images are monochromatic, or in colour. Each pixel corresponds to one or multiple numerical values respectively. When dealing with images, the traditional approach is to flatten images and treat them as vectors of numbers, so that image data is properly formatted for MLPs, or models that are not Neural Networks. This approach disregards the relative positioning of the pixels in an image, so image data treated this way is more inefficient to use in Machine Learning tasks.

Convolutional Neural Networks, or CNNs, were originally designed with the intent to leverage prior knowledge that, in an image, nearby pixels are typically related to each other, to build efficient models for learning from image data.

As seen in many computer vision projects (Gómez & Karatzas, 2017) (Kamilaris & Prenafeta-Boldú, 2018), CNNs are still amply used in Computer Vision problems. However, recent studies have shown

that Transformer-based models are already capable of achieving competitive results — Transformers can beat CNNs in prediction power, speed, and resources.

In Object Detection, large and deep CNNs have been capable of competitive performances on high-difficulty datasets (Krizhevsky et al., 2017) using purely supervised learning. Even with the advent of new models with different architectures, CNNs are usually the top performers when it comes to object detection, as with most other areas within Computer Vision.

A consistent top performer and one of the most famous CNN-based object detection models is the Region-based CNN (R-CNN) (Girshick et al., 2014) model, and subsequent models based on it. From each input image, this model extracts image region proposals using Selective Search (Uijlings et al., 2013), without regard for object categories (Endres & Hoiem, 2010). Then, a CNN computes features for each proposed region, and then each region is classified using class-specific, linear Support Vector Machines.

Currently, one of the most popular CNN-based models for object detection is **YOLO**, which in this case is short for You Only Look Once (Redmon et al., 2016). YOLO treats object detection as a regression problem, determining bounding boxes and class probabilities right from image pixels. Unlike some models, YOLO can make predictions for a full image in one go, rather than having to focus on certain image regions (Gu et al., 2018); thanks to this, YOLO implicitly encodes contextual information.

YOLO is actively being developed and optimized, currently in its 8th version (Jocher et al., 2023) — this is the state-of-the-art model used in this project as a reference, to be compared with the Transformer-based Object Detection model(s), due to its accessibility and quality as a model. YOLOv8 has several advantages, from performing well on many varied tasks to being relatively light as a model, which enables fast, resource-economic training. Therefore, it is a very popular model widely used for Object Detection.

Most versions of YOLO were trained on the ImageNet (Deng et al., 2009) dataset, but YOLOv8 was originally trained on the COCO 2017 object detection and panoptic segmentation dataset (Lin et al., 2015). This dataset contains 118k training images and 5k validation images. Each image is annotated with bounding boxes and panoptic segmentation. This dataset is one of the most popular for object detection due to its variety, both in categories and in scenarios depicted in images. The official YOLOv8 benchmarks, when it comes to metrics such as mAP, all stem from this dataset.

2.2.3. Transfer Learning

In many problems available data may be insufficient to lead to meaningful results when training deep learning models from scratch. The ideal approach would be to gather more data, which can be a very cumbersome, sometimes unsustainable process. For example, in Object Detection it's rare to come across large datasets with which a Neural Network can be properly trained, and adding more data may require manual annotation. Other issues such as datasets having high levels of noise — be it in the relation between a class and a label, or in data objects themselves — are also common in unstructured data. To circumvent these issues, and to generally improve model efficiency, transfer learning (Pan and Fellow, 2009) is often applied in deep learning.

The concept behind transfer learning is applying information gained by training a model to solve one problem to solving different problems, provided the new problems have something in common with

the original problem, such as the field of expertise or the data objects involved. Situations that call for transfer learning are those where a problem that can't be easily solved is similar enough to another problem. This is determined by the problem's domains and tasks: a domain D is constituted by a feature space λ and a marginal probability distribution $P(\lambda)$: $D = \{\lambda, P(\lambda)\}$; and a task T is constituted by a label space μ and an objective predictive function f: $T = \{\mu, f\}$ (Pan and Fellow, 2009). The possible settings for transfer learning are the following:

- Inductive transfer learning: the domain in both the original and the new problem is the same/has the same feature format and probability distribution, but the tasks are different, albeit similar.
- Transductive transfer learning: the task in the original and the new problem is the same/has the same kind of predictive function and has at least part of the label space in common, but the domains are different, albeit similar.
- Unsupervised transfer learning: both the domain and task differ for the two problems, though they are similar; this kind of transfer learning applies to tasks such as clustering and dimensionality reduction (Bengio et al., 2011).

Inductive transfer learning is the most common and applicable and is performed in deep learning by replacing the last layers of a neural network with untrained layers, that will be trained with the new task in mind: since the layers that come before these are trained to identify features of the first task, if said task and the new task are similar then the feature information learned in these layers can be used for the new task.

A common example of this transfer learning in neural networks is, rather than using a single neural network to solve a problem, using one network to extract features from data, so that said data can be used to train a second network to solve the problem. This falls under inductive transfer, as the integrity of whatever data undergoes this process is preserved after going through the first network. Another popular example of transfer learning in deep learning is one that greatly helps in situations where training a model from scratch is unfeasible, usually due to available data or time constrictions: fine-tuning.

2.2.4. Fine-tuning

Fine-tuning is a kind of transfer learning often used in most current deep learning problems, especially those where the available data is scarce. It consists of replacing the classification layer(s) of a model that's been trained on a significantly broad domain, using that model's state as a checkpoint, and training said model on the new domain (Reyes et al., 2015). The features and most weights of the model are transferred to the new domain, as well as other parameters apart from the final classification layers. Compared to training a model from scratch, fine-tuning a pre-trained model is significantly faster and leads to a more accurate final model (Too et al., 2019).

2.3. TRANSFORMERS

Transformers are deep learning models that are characterized by their reliance on a self-attention mechanism: these models' inputs are turned into sequences, and self-attention is used to interpret the relationships between the sequence's elements. Attention mechanisms are not exclusive to transformers: they have been used in feed-forward networks (Chaudhari et al.,2019) and RNNs

(Santana et al., 2021). But unlike other networks, transformer networks have the capability of relying only on self-attention to compute representations of their inputs and outputs, without using convolution or sequence aligned RNNs.

Transformers can attend to complete sequences and can directly learn long-range relationships within the data features as a result, unlike RNNs that can only attend to short-term context. Transformers have been shown to scale to high-end, complex models, and to work well with large-scale datasets.

By design, transformers assume minimal prior knowledge about the problem's structure, compared to CNNs and RNNs (LeCun et al., 2015) (Hochreiter et al., 1999). As such, these models are usually pre-trained on unlabelled, large-scale datasets. An example of this is a Natural Language Processing (NLP) model, BERT (Devlin et al., 2019): this model is trained over different pre-training tasks on unlabelled data, then in a posterior, fine-tuning phase, the model is initialized with the pre-trained model parameters, which undergo fine-tuning in different downstream tasks. Therefore, there is less of a need for manual annotations, with cases usually not requiring them; the representations created with this pre-training model have rich relationships between the entities in a given dataset and are highly expressive and generalizable. However, their effectiveness relies heavily on factors such as the abundance and nature of data (S. Khan et al., 2022).

Originally, Transformer models were designed for NLP tasks, but nowadays, their application to other areas, such as Computer Vision, is being widely researched (S. Khan et al., 2022). Currently, there are many registered successful uses of transformer-based models for object detection (Carion et al., 2020) (Wang et al., 2021), as well as for other computer vision tasks such as image recognition/classification (Zhai et al., 2021), segmentation (L. Ye et al., 2019), video understanding (Sun et al., 2019), image generation (Parmar et al., 2018), and many other tasks (Guo et al., 2021) (Yuan et al., 2021) (X. Wang et al., 2020).

2.3.1. Attention Mechanisms

An attention function maps a query and a set of key-value pairs to an output. The query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key (Vaswani et al., 2017). Attention mechanisms have been used in models such as RNNs, ones used for Natural Language Processing tasks as an example, but Transformers are based solely on these mechanisms and have a unique implementation optimized for parallelization — multi-head attention.

Self-attention: Self-attention is an attention mechanism that computes a representation of a sequence by establishing and evaluating relations between different positions of the same sequence (Vaswani et al., 2017).

Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment, and learning task-independent sentence representations.

Scaled Dot-Product Attention: The kind of attention used in Transformers is called Scaled dot-product attention, a variation of dot-product attention. Its application to transformers can be described by the following formula: $Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$, where Q is a matrix where each line

corresponds to a query, K is a matrix where each line corresponds to a key, V is a matrix where each line corresponds to a value. Each query has a dimension of d_k, the same goes for each key, and the dimension of each value is d_v. It's identical to dot-product attention apart from the scaling factor: $\frac{1}{\sqrt{d_k}}$. For masked self-attention, the formula is slightly different: $MaskedAttention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}} \circ M\right)V$, where M is an upper-triangular matrix, and $\circ$ denotes Hadamard product. While predicting a position in one of the sequences, the attention scores of future positions in the sequence are set to zero in masked self-attention (S. Khan et al., 2022).

Functionally, what is being done in the self-attention process is the following:

1. Dot-product between each query and each key.
2. Scaling by multiplying the dot-product with $\frac{1}{\sqrt{d_k}}$.
3. Masking (Hadamard product with M, this step is optional in scaled dot-product attention but used in transformers).
4. The SoftMax function is applied.
5. Dot-product between the result of the previous steps and the values matrix, V.

This form of attention is used over regular dot-product attention due to the suspicion that, for large values of d_k, the dot products grow large in magnitude: the larger d_k is, the bigger the variance of the dot product of each query and each key will be. For these values, the gradient of the SoftMax function will be very small. To counteract this, scaling was introduced, via the dimension of each query/key. With this method, the queries and the keys' compatibility can be directly assessed; relationships between variables can be ascertained through this method.

In self-attention, both the queries and the key-value pairs described come from the same place.

Multi-head attention: The attention function presented above is not performed only once in one layer; the queries, keys, and values are all linearly projected h times. Each of these linear projections is different, with learned parameters. This is accomplished with parameter matrices: W_i^Q for the projections of the queries, W_i^K for those of keys, and W_i^V for those of the values. In these matrices, each column is a projection, with the columns in W_i^Q and W_i^K having a dimension of d_k, and the columns in W_i^V having a dimension of d_v.

The attention function is applied in parallel to every projection, and the outputs of each application all have a dimension of d_v The results are concatenated, and linearly projected once more, to get to the final values.

In transformers' Multi-head Attention, there are $\mathrm{h} = 8$ parallel attention layers, known as heads. In each head, $d_k = d_v = d_{model} = 64$. Due to the reduced dimension of each head, the total computational cost is like that of single-head attention with full dimensionality.

2.3.2. Transformer Architecture

Both the inputs and the respective outputs of whatever data is used to train the model are embedded (separately), and positional encoding is applied to them. Since Transformers were originally designed for Natural Language Processing tasks, their architecture, like that of many NLP models, has encoder and decoder stacks as its crux: the inputs go through self-attention and a feed-forwarding network in

the encoder stack, and the outputs go straight to the decoder stack, where they undergo through both a self-attention mechanism and an attention mechanism that uses the encoder stack's output.

Encoder stack: The encoder is composed of a stack of N identical layers, each layer typically having two sub-layers. The first sub-layer is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. Both layers are followed by layer normalization (Ba et al., 2015) between the original sequence and the sequence after going through either sub-layer. This is accomplished via residual connections around each sub-layer. The output of each sub-layer is $\mathrm{LayerNorm}(\mathrm{x} + \mathrm{Sublayer}(\mathrm{x}))$, where $\mathrm{Sublayer}(\mathrm{x})$ represents the procedures implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension d_{model} (Vaswani et al., 2017).

Decoder stack: The decoder is also composed of a stack of N identical layers, each layer typically having three sub-layers. Like the encoder, each layer contains a multi-head self-attention mechanism and a position-wise fully connected feed-forward network, but each decoder layer has a third sub-layer to perform multi-head attention: the previous decoder layer's output results in this layer's queries, and the keys and values are taken from the encoder's output. In addition to this, the self-attention sub-layers in the decoder are different from the ones in the encoder: masking is applied in the decoder's self-attention so that for any position p, future positions — p+1; … - are not attended to, just as shown in the masked attention formula. Thanks to this, each position's prediction only depends on known positions. Like the encoder, each decoder sub-layer has a residual connection around it, and layer normalization is performed (Vaswani et al., 2017).

2.3.3. Transformers in Computer Vision

While CNN-based models are typically the go-to for computer vision tasks, Transformers have been steadily gaining more and more popularity in all forms of computer vision tasks. This rise in popularity is closely tied to the concept of attention, which has been steadily increasing in relevance in Computer Vision, even before the creation of the original Transformer neural network. Starting with the Recurrent Attention Model (RAM) (Mnih et al., 2014), a Recursive Neural Network that uses this mechanism. Eventually, self-attention was introduced in Computer Vision, with great success (X. Wang et al., 2017). Later, pure deep self-attention-based models started appearing in Computer Vision, achieving good results as well, and showing that attention-based models have the potential to become a more general and powerful architecture for Computer Vision, compared to CNNs (Guo et al., 2022).

Vision Transformers have been proven to outperform ResNet models by a significant margin when training larger models on larger datasets (e.g., 300 million images), in image classification (Zhai et al., 2021). However, the quadratic complexity of self-attention makes models using the Transformer architecture less suitable for high-resolution images. Though, newer models circumvent that with strategies such as shifted windows (Liu et al., 2021), or with the use of convolutions to process images.

2.3.4. Transformers in Object Detection

Just as with other popular applications of computer vision such as image classification and segmentation, transformers have steadily gained more and more popularity in object detection. Presently, the most popular examples in supervised detection are:

- YOLOS (Fang et al., 2021) – a transformer-based adaptation of YOLO.

- DEtection TRansformer (Carion et al., 2020) – a network that uses a CNN to process images, followed by a transformer that generates predictions from this processed data and annotations.
- Swin Transformer (Liu et al., 2021) – a transformer that uses the shifted windows approach to compute self-attention.

In unsupervised detection, the most notorious network is a variation of the detection transformer: UP-DETR (Dai et al., 2020).

Among these models, the model that was more extensively studied and researched is the **Detection Transformer (DETR)**. DETR is a transformer-based model created for object detection. To use the Transformer architecture in object detection tasks, the inputs require some big changes in representation. Therefore, DETR includes a conventional CNN backbone, which acts as a feature extractor. Said CNN backbone learns a 2D representation of an input image by flattening it and adding positional encoding. Then, this representation is passed to a transformer encoder. At each multi-head self-attention layer of the encoding, spatial position encoding is added to the queries and keys.

A transformer decoder takes a small number of learned positional embeddings as inputs – the object queries. Then, following the typical encoder-decoder structure of transformer models, the outputs of the decoder involve these object queries, and the encoder output as the corresponding keys and values. After going through multiple layers of multi-head self-attention and encoder-decoder attention, the result is passed to a FFN that predicts a detection — an object class and a bounding box or a "no object" class. In a single pass through the decoder, DETR infers a set of N predictions — N is fixed and set to be significantly larger than the typical number of objects in an image. Objects are decoded in parallel at each decoder layer, contrary to the original transformer where objects are decoded one at a time (Carion et al., 2020).

DETR's loss function's outcome is an optimal and bipartite matching between ground truth objects and the model's predictions – regarding class, position, and size. The goal is to find one-to-one matching between the ground truth and predicted objects without duplicates. The finality of finding this matching is the same as that of assignment heuristics used to match ground truth objects and region proposal anchors (Ren et al., 2015) in other detectors (Carion et al., 2020). The fitness of the bipartite matching between a set of N predictions and a set of ground truths is defined by the following cost function: $\hat{\sigma} = \underset{\sigma \in \mathfrak{C}_N}{\operatorname{argmin}} \sum_i^N \mathcal{L}_{match}(y_i, \widehat{y_{\sigma(i)}})$, where $\sigma \in \mathfrak{C}_N$ is a permutation of N elements with the best – in this case, the lowest – cost function value, and $\mathcal{L}_{match}(y_i, \widehat{y_{\sigma(i)}})$ is the cost of matching the pair of the ground truth y_i and a prediction with index $\sigma(i)$. Afterwards, object-specific bounding box losses are optimized. Like YOLOv8, the main build of DETR has been trained on the COCO 2017 (Lin et al., 2015) dataset.

DETR takes a notably longer time to converge than other models such as R-CNN (Ren et al., 2015), and has complexity issues when applied to high-resolution data. These limitations are, however, mitigated in a model adapted from DETR: Deformable DETR (Zhu et al., 2020). This model is powerful, yet still in development.

2.4. Computer Vision in Precision Agriculture

An area that may greatly benefit from Computer Vision applications is precision agriculture. Precision agriculture is the application of technologies, such as object detection, to manage spatial and temporal variability associated with all aspects of agricultural production (Pierce & Nowak, 1999). Its purposes are to increase crop quality and to promote better management of resources and improve environmental quality.

2.4.1. Applications

In agriculture, computer vision has proven its effectiveness on many ends (H. Tian et al., 2020). Due to its versatility, it has been applied in many different forms, such as:

- Crop health and growth monitoring (Pérez-Zavala et al., 2018) (G. Wang et al., 2017).
- Crop harvesting (Y. Ting et al., 2009).
- Evaluation of fruit quality and ripeness (Hossain et al., 2019).

With properly identified image data, these typically time-consuming and intensive tasks have been successfully automated, and made applicable at a large scale, greatly increasing efficiency, and requiring much, much less time and manpower. Through the applications listed above (and many more), computer vision is nowadays heavily used to gauge the economic benefits of agricultural products and to manage crops and resources accordingly.

As computer vision technology has been successfully applied to soil management, crop maturity detection, and crop yield estimates, completely automated intelligent farms have been created. Yet, there are still many unexplored possible applications, and technologies that could be applied, in the ever-expanding field of Precision Agriculture.

It is not uncommon to see systems with multiple image processing and feature extraction techniques, with machine learning models only being applied at the final step (Ponce et al., 2019).

2.4.2. Challenges Faced

While computer vision methods have been very useful and successful in solving problems in agriculture, there are still a few hindrances to current technology. One of them is the absence of a large-scale public database, with research results often relying on smaller datasets, with data usually collected by the researchers themselves, leading to results that are not universal and comparable (H. Tian et al., 2020) due to multiple reasons: biological variations, environmental variations, differences in how data is handled, and so forth.

Another challenge to the widespread usage of computer vision in agriculture, which adds to the first one mentioned, is the great amount of variability that can be found: many possible kinds of crops used, in many different environments — and therefore under different situations when it comes to soil contents, amount of sunlight, water available — and the considered time frames. Due to this, one kind of problem requires notably different workflows for different cases (Tian et al., 2020). These issues are taking time to be addressed, as Artificial Intelligence rapidly progresses.

2.4.3. Similar Case Studies

Computer Vision has been applied to blueberries before, with studies such as that of the application of Deep Learning architectures for the detection of internal blueberry damage (Z. Wang et al., 2018). This was accomplished by using hyperspectral transmittance data to train CNNs — ResNet and ResNeXt — to detect internal damage through the sound's images.

Additionally, a study very similar to the one made on blueberries for this thesis was conducted on grapes, to detect grape bunches (J. Wang et al., 2021). In this project various models were tested — including earlier versions of YOLO and DETR — but the best performer was SwinGD, a model based on the Swin Transformer (Liu et al., 2021).

The main difference between the Swin transformer and the original transformer architecture is that data objects (the images) are cut into smaller windows, reducing the computational complexity when analysing the patches. This would be missing the global effect most Object Detection models have, but the windows are changed from one self-attention layer to another: the gaps in one layer's windows are bridged in the next one. The windows are small at first but get bigger in deeper layers.

3. METHODOLOGY

Extensive tests were performed in order to ascertain whether there's objective value in the use of Transformer-based models for Object Detection tasks.

Two base models were compared in this project: DETR and YOLOv8. In DETR, though the images' features are determined by a CNN backbone, the processing of said features and subsequent classification are conducted by a transformer-based architecture; therefore, DETR was deemed a suitable representative of Transformers used for Object Detection. YOLOv8 is a fully CNN-based model released in 2023 and represents the state-of-the-art due to its known good performances, popularity, and easy practical use. Though DETR, as used in this thesis's research, is still in its earlier versions, YOLOv8 is a model with a great deal of development and optimization — it is the eighth official version of a series of well-known and powerful object detection models.

To perform a thorough examination of DETR, two versions of the model — one with a ResNet50 backbone and one with a ResNet101 backbone — were trained on five different datasets, and test scores from training either version using the COCO2017 dataset were also taken into account.

Both DETR versions were trained and evaluated on data from a real-world problem: identifying fruits — namely, blueberry batches — in a plantation. This data is exclusive and was manually annotated for this project. Several versions of this dataset were used for different training processes, varying in size and in whether data augmentation processes were used. Such processes were applied to the images in their entirety and to the bounding boxes used to mark the objects to be detected.

Additionally, both versions of DETR were also trained on a sample dataset: the balloon dataset, an open-source dataset for object detection tasks, chosen for being easy to understand and to adapt to the code bases used.

To make a proper comparison, a state-of-the-art CNN used for object detection — YOLOv8 — was trained and evaluated on the same datasets. Additionally, the performance of the two models on the COCO 2017 dataset was compared, as both models were originally trained on this dataset.

Rather than training the models from scratch, pre-trained versions of both DETR and YOLOv8 were fine-tuned to either set of training data: as all four blueberry batch datasets and the balloon dataset are too small to realistically train a model — none of these datasets had more than 156 training images — training models from scratch was very unlikely to lead to satisfactory results. Both models were previously trained on the COCO2017 dataset.

When it comes to the DETR variations, whilst a GPU with 12 gigabytes (GB) of RAM can be used to train either model with the default hyperparameters, some hyperparameter settings proved to make these models too heavy to be trained. Therefore, all models with their respective hyperparameter combinations were trained using a 32GB RAM GPU, on all available datasets. YOLOv8 was also trained on the same 32GB RAM GPU, as the most powerful version required between 19 and 20 GB to be trained even on default settings.

3.1. EVALUATING ON COCO2017

Firstly, the performance of both DETR and YOLOv8 on the COCO2017 dataset — the dataset both models were originally trained on — was assessed. When it comes to the many versions of DETR, the metrics considered were the training times, $mAP0.5:0.95$, $mAP0.5$, and the result of applying cross-entropy to DETR's loss. When comparing with YOLOv8, only training time and $mAP0.5:0.95$ were considered. This comparison was made on publicly available data from the training of both models on COCO2017. Unfortunately, it wasn't possible to manually train either DETR variation or YOLOv8 on this dataset, due to a lack of time and resources for training a dataset of this size, so data from public training runs was used.

DETR-r50, DETR-r101, and YOLOv8 were all trained for 500 epochs on COCO2017 data, with each model's default hyperparameter settings.

3.2. REAL CASE DATA

The real case data consists of annotated drone footage of a blueberry plantation. The objects annotated are blueberry batches, as the number of batches in a crop can serve as a proxy of that crop's yield. The footage itself was supplied, and annotation was conducted using **Roboflow**, a web-based machine learning platform. For this project, horizontal bounding-box annotations were used. The second reason stems from the following: in the available footage, the objects — blueberry batches — can be hard to tell apart without some context, even when manually annotating the images. An example of this is leaves being confused with blueberry batches in certain situations, when they get directly obfuscated by sunlight.

Each model was trained with four separate datasets stemming from the real case: two of these datasets used the original footage, and the remaining two datasets used a zoomed-in version of the footage. For the remainder of this paper, these datasets will be referred to as:

- Original footage (3840x2160), 156 training images, 104 of those resulting from data augmentation: **BB-1.**
- Original footage (3840x2160), 52 training images, no data augmentation performed: **BB-2.**
- Zoomed-in footage (960x540), 156 training images, 104 of those from data augmentation: **BB-a.**
- Zoomed-in footage (960x540), 52 training images, no data augmentation performed: **BB-b.**

BB-1 and BB-2 were created first, with no intent to generate any more datasets originally. But these datasets proved to be significantly imbalanced; since only one class is represented, the objects have relatively similar sizes, and the loss functions used in either DETR or YOLOv8 aren't conflicting with each other, the imbalance is theorized to stem from one of two likely issues:

- foreground-background imbalance – class imbalance.
- placement of objects in an image – spatial imbalance (Oksuz et al., 2019).

Figure 3 – Frame taken from the original blueberry footage; this image is annotated and present in the BB-1 and BB-2 datasets.

To mitigate these issues, the BB-a and BB-b datasets were created. The key difference between these two datasets and the other blueberry datasets is that they had zoomed-in images: every image in BB-a or BB-b originally had $\frac{1}{16}th$ of the area of any image in the BB-1 and BB-2 datasets. The differences between the two groups of datasets is exemplified by the differences between Figure 3 and Figure 4. Since these datasets used zoomed-in footage, the total area occupied by each object in an image is significantly higher than in the original footage, thus diminishing the possibility of foreground-background imbalance. Additionally, images don't have very different concentrations of objects based on certain areas: no image has a high number of objects in multiple areas and no objects in multiple different areas. In these new datasets, although much fewer objects of interest — blueberry batches — were present in each image, the space occupied by them was usually larger, and the objects themselves were relatively much bigger and distinguishable in these images.

BB-1 and BB-2 differ in that one of them has images resulting from data augmentation, and the other does not. BB-a and BB-b also differ from each other in this regard. Both datasets containing augmented data — that is, BB-1 and BB-a — have a total of 156 training images and 12 validation images. Out of the training images, 52 were manually annotated, and the remaining 104 are the result of applying to the first 52 images the following data augmentation processes:

- **Variation of the image's saturation** – a resulting image's saturation was between 75% and 125% of the original image.
- **Bounding box rotation** – the contents in an image's bounding boxes were rotated, to increase the model's resistance to objects that have different directions.

Figure 4 - Frame taken from the zoomed-in blueberry footage; this image is annotated and present in the BB-a and BB-b datasets.

The datasets with no augmentation processes applied have a total of 52 training images and 12 validation images. The annotated datasets were converted into two different annotation formats: the COCO format, and the YAML format. DETR was trained with data in the COCO format, and YOLOv8 was trained with data in the YAML format. The following diagram shows the full dataset production process and preparation:

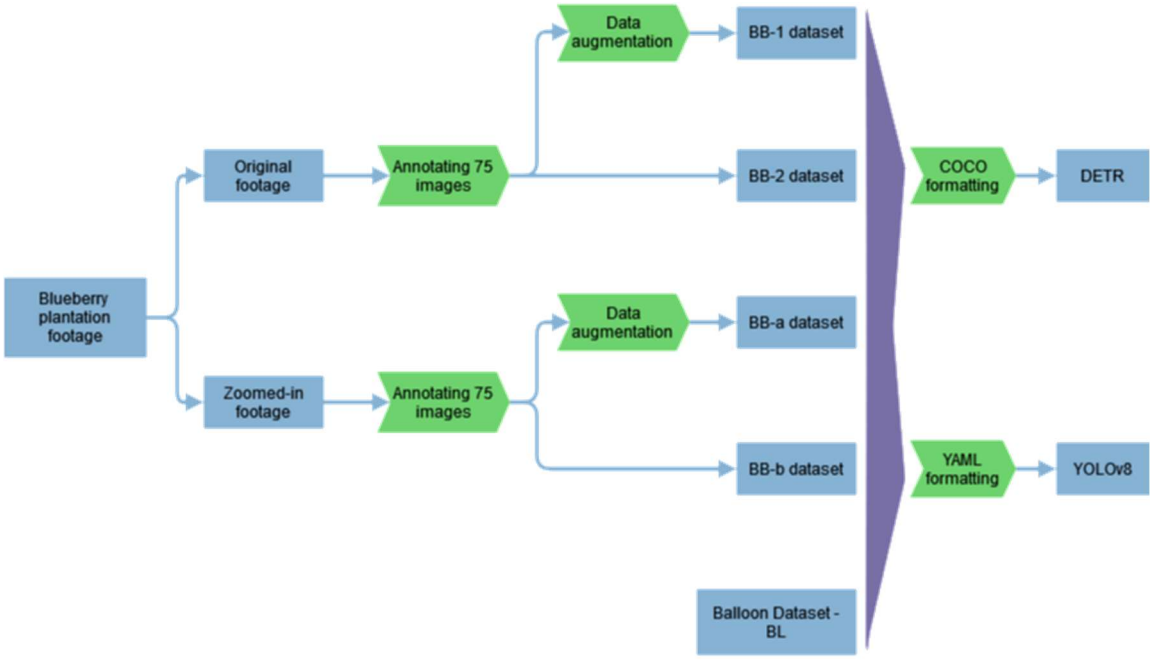

Figure 5 – Full dataset preparation pipeline

3.3. SAMPLE DATA

The sample data is the balloon dataset, consisting of 61 training images, 15 validation images, and 9 test images. This dataset consists of pictures with varying amounts of balloons and correspondent annotations and is publicly available. No data augmentation processes were applied. Due to some labeling inconsistencies — balloons were wrongly labeled as something else — both the COCO formatted and YOLOv8 formatted versions of this dataset were slightly changed from their original state to correct this issue using simple parsing methods.

For the remainder of this paper, this dataset will be referred to as **BLN**.

Figure 6 – Image from the balloon dataset, without annotations.

3.4. TRAINING DETR

Due to the scarcity of available data for every dataset, training from scratch wasn't feasible, therefore fine-tuning was used to properly train DETR on every dataset used. To do this, a slightly modified branch of the DETR repository was used — *https://github.com/HitGobba/detr* — as well as a modified version of the Jupyter Notebook "finetune_detr.ipynb" contained in this repository: *https://github.com/woctezuma/finetune-detr*.

Two different variants of DETR were trained on this data:

- DETR-r50
- DETR-r101

DETR-r50 comes with a ResNet50 backbone, and DETR-r101 in their name comes with a ResNet101 backbone.

For both versions of DETR, at first, there were 5 different training runs per dataset, where the only variation within them was in the number of epochs: 100, 200, 300, 400, 600. These numbers were picked based on the default number of epochs for this implementation of DETR: 300. The supposedly ideal number of epochs was determined by the results of these training runs; more specifically, by the model's cross-entropy loss and mean Average Precision variations during these training runs, as well as by the runtime. The former was more important in this first phase, as it is an indicator of overfitting in the model: it's ideal to stop training if the model starts to overfit, but the latter is also quite significant, as mAP is the main indicator used in this project to gauge a model's power and capacity to perform object detection with the best possible quality. Cross-entropy loss was monitored for training data and validation data as well, but mAP scores were only monitored for validation data.

For DETR, $mAP0.5{:}\,0.95$ was the main indicator, with $mAP0.5$ serving as a secondary indicator, of mean average precision. Therefore, in some situations, a higher number of epochs was chosen despite any evidence of overfitting, as the following tests and hyperparameter changes could potentially mitigate this issue to a satisfactory level. The formula used to determine the best mAP was the following:

$$mAP = 0{,}9 * p + 0{,}1 * q;\ p = mAP0.5{:}\,0.95, q = mAP0.5.$$

Table 1 – The "ideal" number of epochs for each dataset, for each model backbone.

	DETR–r50	**DETR–r101**
BB-1	400	300
BB-2	350	400
BB-a	350	105
BB-b	270	220
BLN	300	350

The following tests consisted of slightly changing some hyperparameters' values whilst fixing the others, to gauge the influence of said hyperparameters. For these tests, the number of epochs was fixed at the value corresponding to the dataset/backbone combination in Table 1. The hyperparameters in question are the following:

- Number of attention heads within the multi-head attention mechanisms.
- The transformer's initial learning rate.
- Size of the batches of data (images) used in training.

First, each of these hyperparameters' influence was tested in isolation: in each training run, only one hyperparameter was changed. Then, upon evaluating the outcome of training, further testing was conducted, changing multiple hyperparameters simultaneously. However, some hyperparameter changes were not combined in some cases, on account of results from the individual hyperparameter testing. An example of this is the combination of an increased batch size and an increased number of attention heads: the former contributed to overfitting issues and, after testing it in one model for each dataset, it was revealed that the latter didn't mitigate overfitting, therefore their combination would be expected to lead to overfitting.

Table 2 – All tested hyperparameter value combinations, including the model's default.

Test	Attention heads	Initial learning rate	Batch size
0	8	0,0001	2
1	16	0,0001	2
2	8	0,001	2
3	8	0,00001	2
4	8	0,0001	4
5	8	0,00001	4
6	16	0,00001	2
7	16	0,0001	4
8	16	0,00001	4
9	8	0,000001	2

Table 3 – tests realized in each dataset/DETR backbone combination.

	DETR – ResNet50	DETR – ResNet101
BB-1	0,1,2,3,4,5,6,7,8	0,1,2,3,4,5,6,8
BB-2	0,1,2,3,4	0,1,3,4
BB-a	0,1,2,3,4,9	0,1,3,4,5
BB-b	0,1,3,4,9	0,1,3,4,9
BLN	0,1,2,3,4,5,6,7,8	0,1,3,4,5,6,8

After these tests, for each dataset/backbone combination, the hyperparameter combination of the trained model with the best balance between mAP scores on the validation dataset (as high as possible) and overfitting (as little as possible) was chosen, and another instance of DETR-r50/DETR-r101 with the same hyperparameters was trained on both the training and the validation data, and its

performance on test data was ultimately evaluated. Whether or not a model was overfitting was informed by DETR's personalized loss function (Carion et al., 2020).

A model was deemed to overfit if a steady increase in the loss function's value (both loss functions used have minimization as their objective) on validation data during training were to be observed. This phenomenon leads to a growing disparity between the training loss and the validation loss, which is correlated with overfitting.

The best epoch, this time based on test data, was chosen. This was possible because, when training DETR, validation data had no influence in the training process itself, and only served to garner further insights into the model's performance. Naturally, there was no risk of data overflow either.

3.5. TRAINING YOLOV8

YOLOv8 comes in multiple variants for object detection: nano(n), small(s), medium(m), large(l), and extra-large(x). These versions vary in complexity and, therefore, in file size and in how computationally expensive they are, with the extra-large model — YOLOv8x — naturally being the most complex, and the most powerful, albeit the most computationally expensive and ultimately cumbersome to run. Out of these variations, YOLOv8x was used in this project due to it being the most powerful state-of-the-art CNN-based model available to compare DETR to and the available resources being enough to train this model.

For YOLOv8, the testing procedures were very similar to those used for DETR, starting once again with 5 different training runs, where the only variation within them was in the number of epochs: 50, 100, 200, or 300. As with DETR, these numbers were chosen based on the default number of epochs used in training for YOLOv8x — 100. Likewise, the supposedly ideal number of epochs was determined by the results of these training runs. Also, mAP was used as the metric to evaluate the model's predictive power, with $mAP0.5{:}\,0.95$ as its main indicator, and $mAP0.5$ as its secondary indicator, using the same formula as DETR.

Since YOLOv8 doesn't have a personalized loss function like DETR, bounding box loss was the way to assess overfitting. However, this function isn't normally restricted to binary values, unlike DETR's cross-entropy loss, meaning that any discrepancies between training loss values and validation loss values are more significant in this case.

Table 4 – The "ideal" number of epochs for each dataset, for YOLOv8x.

	YOLOv8x
BB-1	190
BB-2	100
BB-a	170
BB-b	115
BLN	200

As with DETR, the following tests consisted of changing some hyperparameters' values:

- Initial learning rate.
- Final learning rate.
- Size of the batches of data (images) used in training.

Once again, each of these hyperparameters' influence was tested in isolation. In each training run, only one hyperparameter was changed and, upon evaluating the outcome of training, further testing was conducted: changing multiple hyperparameters simultaneously, but leaving certain combinations out based on previous results. Once again, for each dataset, the hyperparameters of the model with the best performances were adopted by another instance of YOLOv8: In this case, the data used to validate the model's performance when testing epoch numbers and hyperparameters was added to the training data split, and a test split was used to evaluate the model's performance.

The final learning rate in YOLO can be customized but is also affected by the initial learning rate: the true final learning rate is the result of $lr0 * lrf$, $lr0$ being the initial learning rate and lrf being the final learning rate parameter. Another noteworthy aspect of YOLOv8 is its built-in patience threshold: after a certain number of epochs without noticeable improvements, the model stops training. The patience threshold used for every training run, including that of the final model, was 120 epochs.

Table 5 – All tested hyperparameter value combinations for YOLOv8, as they are inputted.

Test	Dropout rate	Initial learning rate	Final learning rate	Batch size
0'	0	0,01	0,01	16
1'	0	0,001	0,01	16
2'	0	0,01	0,001	16
3'	0	0,01	0,01	4
4'	0	0,001	0,001	16
5'	0	0,01	0,001	4

Table 6 – Tests realized for each dataset.

	YOLOv8
BB-1	0',1',2',3',4',5'
BB-2	0',2',3',5'
BB-a	0',1',2',3',5'
BB-b	0',2',3'
BLN	0',1',2',3',4',5'

4. RESULTS AND DISCUSSION

The results showed that when trained with imbalanced data, as the real case data appears to be, YOLOv8 performed marginally better than DETR, but on more consistent data, such as the balloon or COCO2017 datasets, DETR achieved results close to, and in the balloon dataset case, slightly better than YOLOv8.

Naturally, real case data can come in several shapes and sizes, and can lead to very inconsistent results, as seen in this thesis. An example of this is blueberry datasets having almost exclusively small objects — not only is this a data imbalance issue, but smaller objects are also harder to detect — whilst objects in balloon data were large and medium-sized. Nonetheless, the results obtained showed DETR has potential as an Object Detection model.

When it comes to overfitting, slightly different standards were applied to DETR and YOLOv8 models:

- On DETR, anything would be fine if cross-entropy loss values on the validation set didn't steadily increase at any point during training.
- On YOLOv8, anything would be fine if box loss on the validation set didn't steadily increase at any point during training, though a big difference between box loss values in training data and in validation data was something to be avoided.

Results are thoroughly reported and discussed in this section. Additionally, one can find training logs and statistics, as well as graphics depicting the evolution of the values of important metrics — cross-entropy loss, $mAP0.5$ and $mAP0.5:0:95$ — throughout training cycles, in this GitHub repository: ***https://github.com/HitGobba/TIOD-results***. This repository will be referred to as TIOD-results throughout the rest of the thesis. Main results can be found in this document, though.

4.1. RESULTS ON COCO2017

Both variations of DETR, as well as YOLOv8x (the only YOLOv8 variant used), are associated with public results of training runs where all these models were trained with the COCO2017 dataset. Furthermore, the "official" checkpoints of these models, available for fine-tuning purposes, are trained with COCO2017. Therefore, since COCO2017 is a very big and well-rounded dataset with 80 object categories, the performances of DETR-r50, DETR-r101, and YOLOv8x, on this dataset, were compared, to gain further insights.

4.1.1. DETR – ResNet50 backbone

The training results corresponding to this model when trained with COCO2017 data featured a small but growing difference between the cross-entropy loss in validation data and the cross-entropy loss in training data after the 400th epoch, but this difference isn't very alarming. When it comes to mAP scores, they rose sharply for about 30 epochs and kept increasing less and less until the 400th epoch, where both $mAP0.5:0:95$ and $mAP0.5$ saw a slight upward spike but stabilized afterwards until the end of the training cycle.

4.1.2. DETR – ResNet101 backbone

When training this model with COCO2017 data, the results were very similar to those from training DETR-r50, but with slightly better $mAP0.5:0:95$ values throughout the entire training cycle. Aside from this variable, every other variable's values were extremely similar between DETR-r50 and DETR-r101, with $mAP0.5:0:95$, $mAP0.5:0:95$ and cross-entropy loss evolving over training time in DETR-r101 almost exactly like when training DETR-r50 with the same dataset.

Graphics depicting cross-entropy loss, $mAP0.5$ and $mAP0.5:0:95$, for both DETR-r50 and DTR-r101 trained from scratch with COCO2017 data, are in the TIOD-results repository, in the following path: "detr coco".

4.1.3. YOLOv8x

YOLOv8x got higher mAP scores than either DETR variant when trained with COCO2017 data. The difference between YOLOv8x and DETR models isn't tremendously great, however, when it comes to mAP scores. No evidence of YOLOv8x overfitting to COCO2017 training data was found, additionally. YOLOv8's loss values dropped very quickly in training but stabilized shortly after the 10th epoch at values close to 1. A similar phenomenon occurred regarding the mAP scores: there was a huge increase in mAP scores with every passing epoch for the first few epochs, but mAP started increasing less and less with time, though never completely satisfied.

4.1.4. Model Results

Logs of the final models' results, as well as checkpoints of those same models at their best epoch (when it comes to mAP values) can be found in the official DETR repository — *https://github.com/facebookresearch/detr* — for DETR models, and in the Ultralytics YOLOv8 documentation — *https://docs.ultralytics.com/models/yolov8* — for YOLOv8x.

Table 7 – Results of the training scores from the final models trained with COCO2017 data.

	DETR – ResNet50	DETR – ResNet101	YOLOv8x
Number of features	41302368	60242272	68229648
Number of epochs	500	500	500
Highest mAP50:95	0,4234	0,4370	**0,5383**
Highest mAP50	0,6262	0,6409	**0,7046**
Best score[(1)]	0,4436	0,4573	**0,5549**
Best epoch	474	488	462

[(1)]score formula: $0{,}9 * (mAP0.5:0.95) + 0{,}1 * (mAP0.5)$

4.2. RESULTS ON BB-1

The training results on this dataset were extremely poor, which indicated that the dataset may be imbalanced (a supposition that would be confirmed by training with YOLOv8x). As an example, any

training runs made only to test hyperparameters were not getting $mAP0.5$ scores (on the validation dataset) higher than $0{,}08$. This is due to the nature of the data in question: the blueberry plantation images in this dataset, and every other blueberry dataset in this work, are mostly filled with noise, which can seriously hamper the predictive power of the trained model. When it comes to the definitive trained models (trained on training AND validation data) the mAP scores showed significant improvements over these values, as seen in Table 8.

4.2.1. DETR – ResNet50 backbone

Firstly, the ideal number of epochs was determined: only the usual epoch numbers of 100, 200, 300, 400, and 600 were tested, and the results of these tests would show that 400 is the most viable number of epochs. Whilst the mAP scores were extremely low in every test, they'd stop meaningfully increasing and would stabilize when training was conducted over more than 400 epochs. Also, no significant overfitting was observed in these tests, no matter the number of epochs, so a number as high as this was deemed safe for hyperparameter testing.

This dataset/backbone combination was the first to be tested during research, and therefore, every hyperparameter combination was tested. The hyperparameter tests indicated trends that would prove to be consistent for the remaining dataset/backbone combinations, such as:

- An increased number of attention heads being correlated with overfitting and lower mAP scores — seen on tests 1 and 8.
- A decreased learning rate possibly mitigating or outright preventing overfitting — seen on tests 5 and 8, from the combination of a decreased learning rate with an increased batch size, under the hypothesis that the latter causes overfitting.

Additionally, training the model with an initial learning rate of $0{,}001$, higher than the standard learning rate, led to a curious effect: the model's training flatlined, in that mAP scores were fixed at zero, and that cross-entropy loss was consistent throughout the entirety of the training process, with the validation loss being slightly lower than the training loss. Due to this, this hyperparameter change wasn't combined with any others in testing.

As shown by the results of the conducted tests, the most successful was test 3, where an initial learning rate of $0{,}00001$ was used, with no other hyperparameter changes. This is due to different hyperparameter configurations leading to either lower mAP scores on validation data, or overfitting, as shown by the cross-entropy loss changes during training to both training and validation sets.

Logs of the tests conducted, as well as graphics depicting cross-entropy loss, $mAP0.5$ and $mAP0.5{:}\,0{:}\,95$, are in the TIOD-results repository, in the following path: "detr BB-1/r50", and Table 13, located in the Appendix, presents training times and best scores corresponding to the conducted hyperparameter tests.

4.2.2. DETR – ResNet101 backbone

When training DETR-r101 with the default settings of attention heads, batch size, and initial learning rate, this model showed clear signs of overfitting from the increase of the cross-entropy loss on validation data — as the cross-entropy loss on training data was decreasing — when trained on more than 200 epochs, which indicates overfitting. However, the mAP scores also showed a significant

increase at around that mark, therefore the following ablation studies were conducted with training cycles of **350** epochs, with the prospect of mitigating overfitting by changing other hyperparameters. 350 epochs was the limit, as any increase in mAP stemming from longer training runs would already be overshadowed by the increasing level of overfitting and high training time.

The basic hyperparameter tests — tests 1, 2, 3, 4 — were conducted. A higher number of attention heads once again led to slightly lower mAP scores and so did an increased batch size. When testing only a lower learning rate, the mAP scores were slightly higher than normal, and the model's cross-entropy loss on validation data also decreased with time, which indicates that a lower learning rate could lead to less/no overfitting for the model.

All other tests were performed except for test 7. Test 7 consists of increasing the batch size and the number of attention heads simultaneously. In theory, this combination wouldn't lead to higher mAP scores or positive progress regarding the overfitting issue present in most tests involving this model and dataset. These suppositions are also backed by the results of the same test when conducted on DETR-ResNet50.

Every test but 3 and 5 showed signs of overfitting. The settings corresponding to these tests are exactly alike, aside from the batch size. These tests also use an initial learning rate of $0{,}00001$. Once again, the most successful model was that where an initial learning rate of $0{,}00001$ was used, with no other hyperparameter changes (Test 3).

Logs of the tests conducted, as well as graphics depicting cross-entropy loss, $mAP0.5$ and $mAP0.5{:}\,0{:}\,95$, are in the TIOD-results repository, in the following path: "detr BB-1/r101", and Table 14, located in the Appendix, presents training times and best scores corresponding to the conducted hyperparameter tests.

4.2.3. YOLOv8x

When trained with this data, YOLOv8x performed very poorly, albeit marginally better than either variation of DETR. Most hyperparameter settings didn't lead to significant signs of overfitting. YOLOv8x also achieved good values (the best possible values it could in theory) in significantly fewer epochs than either variation of DETR. Due to its standard batch size being significantly higher than DETR's as well, training one epoch in YOLOv8x is much faster than training one epoch in DETR. Due to this, YOLOv8x training was much, much faster, than DETR training.

When it comes to the epoch tests — training runs with 50, 100, 200, and 300 epochs — no traces of overfitting were exposed by the bounding box loss in validation data, and mAP scores rose sharply until about the 30th epoch, then had a gradual increase in some of these test runs but stabilized in others. When performing both the 200 and the 300 epoch runs, early stopping was triggered, though in the 300-epoch run, it was only triggered well after the 200th epoch. Knowing this, the epoch number for any future training runs with this data, to avoid losing any opportunity to have a potentially better model, was set at **190**.

When realizing the hyperparameter tests, test 0' — the model's default settings — had the best results regarding the mAP scores. Two of the tests — 1' and 4' — had the same exact training results as other tests: 1' (changing the initial learning rate) had the same results as 0' (default settings), and 4' (changing the initial and final learning rates) had the same result as 2' (changing the final learning rate).

Therefore, a hypothesis was raised: The "lr0" hyperparameter, standing for the model's initial learning rate, has no effect on model performance.

Apart from these tests, tests 2', 3', and 5' culminated in worse results than test 0' regarding mAP scores. None of the tests resulted in models overfitting to training data, as shown by box loss values on validation data.

In every training run, some of the earlier epochs have a non-existent box loss value for validation data — this would be found to also occur in all other YOLOv8x training runs when training with other datasets. However, after those earlier epochs, every epoch has a real box loss value on validation data. Therefore, a box loss curve for validation data can still be traced.

Logs of the tests conducted, as well as graphics depicting box loss, $mAP0.5$ and $mAP0.5:0:95$, are in the TIOD-results repository, in the following path: "yolov8/bb-1", and Table 15, located in the Appendix, presents training times and best scores corresponding to the conducted hyperparameter tests.

4.2.4. Model Results

Logs of the final models' results, as well as checkpoints of those same models at their best epoch (when it comes to mAP values) can be found at "detr BB-1/r50 final", "detr BB-1/r101 final" and "yolov8/bb-1 final" for DETR-r50, DETR-r101 and YOLOv8x respectively.

Table 8 – Results of the training scores from the final models trained with BB-1 data.

Final models	DETR – ResNet50	DETR – ResNet101	YOLOv8x
Number of features	41279238	60219142	68153571
Number of epochs	400	300	190
Highest mAP50:95	0,1563	0,1681	**0,2393**
Highest mAP50	0,4235	0,4492	**0,5383**
Best score[(2)]	0,1830	0,1953	**0,2674**
Best epoch	396	294	91
Training time (h:m:s)	03:16:50	03:09:27	00:24:50

[(2)]score formula: $0{,}9 * (mAP0.5:0.95) + 0{,}1 * (mAP0.5)$

4.3. RESULTS ON BB-2

When training the model with this dataset, only the simplest ablation studies were conducted: only the isolated effects of changing the studied hyperparameters (as specified in Chapter 3.3) were studied. This is due to this model's scores already being very low when trained on this data, even compared to the other datasets, making further testing fruitless when it comes to finding the best conditions for DETR to perform well.

The difference in mAP scores and in the cross-entropy loss values, between models trained with BB-1 and those trained with BB-2, isn't very significant. However, in models trained with BB-2 data, the

disparity between cross-entropy loss values in training data and loss values in validation data is lower than in models trained with BB-1 data, when the hyperparameter settings are the same in both models. From this, a hypothesis comes to light: datasets containing images resulting from data augmentation are more prone to overfitting in this problem. Once again, the mAP scores in the final models showed significant improvements over the hyperparameter test values, as seen in Table 8.

Additionally, test 2 (DETR only) was only realized on the ResNet50 backbone. Since this test had already been realized on the BB-1 dataset for both backbones and training proved to be completely ineffective, this test was only realized on this dataset to determine whether the size of the dataset may have been too low for fine-tuning to originate a proper trained model. Either backbone would be suited to realize this test one more time, and it was concluded that test 2 would inevitably lead to no useful results when training this model with a ResNet101 backbone on this same data, as the same result — $max.mAP = 0; max.mAP0.5 = 0$ — was obtained. Therefore, this test wasn't realized for any other blueberry batch dataset afterward.

4.3.1. DETR – ResNet50 backbone

When training DETR-r50 with this dataset, mAP scores were lower than when the same model was trained with the BB-1 dataset, for all basic tests. However, the difference between cross-entropy loss in training data and cross-entropy loss in validation data was consistently much lower, which shows that the model overfits to training data less than when trained with BB-1. So far, the supposition is that data augmentation leads to more overfitting, but also to increased correctness of the model's predictions.

Since there were seemingly no overfitting issues, training time was the only constriction when establishing an ideal epoch number. When it comes to mAP scores, they increased steadily until a growth spike was hit shortly after the 200th epoch. Afterward, mAP kept increasing steadily until shortly before the 350th epoch, where it was mostly consistent, and showed no signs of increasing, for the remainder of training when using higher epoch numbers. Therefore, **350** epochs was deemed a fit number, as any more would theoretically not improve model performance in any meaningful way.

As mentioned before, only the general tests — 0, 1, 2 (to confirm the suspicion of the effect of a lower learning rate), 3, and 4 — were conducted, since mAP was so low no hyperparameter combination would bolster it in a meaningful way. Once again, test 3 produced the best results. The decision was fully based on mAP in this scenario, as overfitting wasn't an issue in any test.

Logs of the tests conducted, as well as graphics depicting cross-entropy loss, $mAP0.5$ and $mAP0.5{:}\,0{:}\,95$, are in the TIOD-results repository, in the following path: "detr BB-2/r50", and Table 16, located in the Appendix, presents training times and best scores corresponding to the conducted hyperparameter tests.

4.3.2. DETR – ResNet101 backbone

Unlike what occurred when training this model with BB-1, overfitting wasn't observed in almost every training run, though it wasn't fully absent. First, the epoch tests showed that **400** would be an appropriate number of epochs, judging from the tests, even though the trained model would start suffering from overfitting shortly after epoch 200. This is due to the disparity between cross-entropy

loss in training and in validation data being lower when this model is trained with BB-2 than when this model is trained with BB-1, for epoch tests.

The basic hyperparameter tests — tests 1, 3, 4 — were conducted. Whilst a higher number of attention heads and an increased batch size once again led to slightly lower mAP scores, no concrete evidence of overfitting was found in tests 3 and 4. When testing only a lower initial learning rate, the mAP scores were slightly higher than normal, and the model's cross-entropy loss on validation data also decreased with time, which indicates that a lower learning rate could hypothetically lead to less/no overfitting.

No other test was conducted, due to the mAP scores on tests 1, 3, and 4 being very low, making it so that combining parameters to try to get better scores wouldn't be too fruitful. Therefore, the best model ultimately corresponds to the one with the hyperparameter settings of Test 3: $0{,}00001$ initial learning rate.

Logs of the tests conducted, as well as graphics depicting cross-entropy loss, $mAP0.5$ and $mAP0.5{:}\,0{:}\,95$, are in the TIOD-results repository, in the following path: "detr BB-2/r101", and Table 17, located in the Appendix, presents training times and best scores corresponding to the conducted hyperparameter tests.

4.3.3. YOLOv8x

When performing the first training runs to determine the ideal epoch number, like what succeeded when training this model with BB-1, training this model with BB-2 led to a more powerful model than both DETR-r50 and DETR-r101 trained with BB-2. When it comes to the epoch number itself for this model, due to the mAP scores quickly rising and stabilizing the epoch number settled on in this case was **100**.

Due to results obtained from training this model with BB-1, only tests 0', 2', 3', and 5' were realized, since any test involving the initial learning rate was redundant due to this parameter not having any influence on training results. When it comes to the models trained under the settings of tests 3' and 5', the box loss values of training data are very similar to those of test data, meaning that the models are decidedly not overfitting. However, the mAP scores were so low compared to the model trained using the settings of test 2' that the final model was trained using the latter's test settings, as the model trained with BB-2 in these conditions showed some evidence of overfitting but compensated with the model's performance.

Just like what occurred when training YOLOv8x with BB-1 data, in all training runs, some of the earlier epochs have a nonexistent box loss value for validation data. However, after those earlier epochs, every epoch has a real box loss value on validation data. Therefore, a box loss curve for validation data can still be traced.

Logs of the tests conducted, as well as graphics depicting box loss, $mAP0.5$ and $mAP0.5{:}\,0{:}\,95$, are in the TIOD-results repository, in the following path: "yolov8/bb-2", and Table 18, located in the Appendix, presents training times and best scores corresponding to the conducted hyperparameter tests.

4.3.4. Model Results

Logs of the final models' results, as well as checkpoints of those same models at their best epoch (when it comes to mAP values) can be found at "detr BB-2/r50 final", "detr BB-2/r101 final" and "yolov8/bb-2 final" for DETR-r50, DETR-r101 and YOLOv8x respectively.

Table 9 – Results of the training scores from the final models trained with BB-2 data.

Final models	DETR – ResNet50	DETR – ResNet101	YOLOv8x
Number of features	41279238	60219142	68153571
Number of epochs	350	400	100
Highest mAP50:95	0,1121	0,1293	**0,2574**
Highest mAP50	0,3358	0,3699	**0,5576**
Best score[(3)]	0,1345	0,1534	**0,2875**
Best epoch	345	363	99
Training time (h:m:s)	01:26:28	01:55:53	00:09:04

[(3)]score formula: $0{,}9 * (mAP0.5{:}\,0.95) + 0{,}1 * (mAP0.5)$

4.4. Results on BB-a

Just as with the BB-1 and BB-2 datasets, despite this dataset being composed of smaller frames than the two mentioned, there were still strong signals of data imbalance. However, taking more drastic measures to have a dataset without this issue more assertively was unfeasible. Though, there were significant improvements to the model's performance when trained on this dataset compared to models trained on BB-1 or BB-2, which can attest to that the issues with those two datasets were mitigated.

Unfortunately, training with this dataset showed very high levels of overfitting for both DETR with the ResNet50 backbone and DETR with the ResNet101 backbone, with cross-entropy loss on validation data rising during training done to ascertain a proper epoch number, and during hyperparameter tests.

When training DETR with BB-1 or BB-2, any tests involving lowering the initial learning rate to $0{,}00001$ showed no overfitting. Additionally, in some cases, repeating one of these tests' circumstances aside from the decreased learning rate resulted in an overfitting model. With this correlation in mind, a possible solution to the overfitting problems encountered emerged: lowering the learning rate even further. Thus, Test 9 — initial learning rate of $0{,}00001$ with no other hyperparameter changes — was created.

After running into these issues in DETR, YOLOv8 was trained on the same dataset, and trained YOLOv8 models also registered overfitting. The mAP scores of YOLOv8 models trained with BB-a were slightly higher than the scores of models trained with either BB-1 or BB-2 data, but the difference in performance between YOLOv8 and either DETR variant is proportionally smaller than the difference in performance between the same models when trained with BB-1 or BB-2 data.

4.4.1. DETR – ResNet50 backbone

When conducting the usual epoch tests, the cross-entropy loss on validation data was slightly increasing with time until about the 200th epoch, then it started growing, as did the disparity between it and cross-entropy loss in training data. However, when observing the evolution of mAP scores with time, the conclusion was that it was still worthwhile to perform the remaining tests with **350** epoch runs. After deciding that, it was time to test models once again.

Test 2 was realized for the final time on blueberry plantation data. With similarly useless results as when DETR was trained with BB-1 and BB-2 under the same conditions, the conclusion was that increasing the learning rate in DETR would not lead to meaningful results on any blueberry data. Therefore, this test was not repeated anymore.

As overfitting issues were present in every epoch test and tests 0, 1, 3, and 4, when training this model with BB-a, Test 9 was attempted. However, the cross-entropy loss in validation data still increased steadily during training after a certain epoch, though less so than during other training runs with this model on the same dataset.

Afterward, not to alter the learning rate any further, the epoch number was limited in the final model: rather than 350 epochs, the final model was trained over only **100** epochs, following the hyperparameter settings of test 9, as the cross-entropy loss on validation data didn't see a systematic increase until a few epochs after that, and mAP scores didn't significantly improve under these settings after 100 epochs when performing Test 9.

Logs of the tests conducted, as well as graphics depicting cross-entropy loss, $mAP0.5$ and $mAP0.5{:}\,0{:}\,95$, are in the TIOD-results repository, in the following path: "detr BB-a/r50", and Table 19, located in the Appendix, presents training times and best scores corresponding to the conducted hyperparameter tests.

4.4.2. DETR – ResNet101 backbone

Once again, the usual epoch tests were conducted, with the supposed ideal number of epochs being much lower than in other model/dataset combinations. This is not exclusively due to the overfitting issues with DETR when trained on this dataset: the highest mAP score was encountered at epoch 54, and while mAP scores varied less after a few epochs, they stabilized a bit around the 95th epoch. Therefore, further testing was conducted with **105** as the definitive number of epochs (10 epochs were added to allow for some leeway when it comes to model performance variations caused by hyperparameter changes).

In this case, since the ideal number of epochs was very low, the observed overfitting didn't require drastic measures to mitigate: the realization of test 3 led to a model with no concrete evidence of overfitting, further cementing the hypothesis that lowering DETR-r101's learning rate reduces the risk of overfitting. This also applied when this model was trained with the hyperparameter settings corresponding to test 5, which produced the best mAP scores. Therefore, the final model followed these hyperparameter settings.

Logs of the tests conducted, as well as graphics depicting cross-entropy loss, $mAP0.5$ and $mAP0.5{:}\,0{:}\,95$, are in the TIOD-results repository, in the following path: "detr BB-a/r101", and Table

20, located in the Appendix, presents training times and best scores corresponding to the conducted hyperparameter tests.

4.4.3. YOLOv8x

As usual, the tests to determine the "ideal" number of epochs were conducted. But in this case, whilst the overfitting was present as it has been when training with either BB-1 or BB-2, the mAP scores of YOLOv8x models trained with BB-a were comparable to those of DETR models trained with the same dataset, rather than being unequivocally better. From the four training runs performed with different numbers of epochs, it was determined that **170** was a fitting epoch number to train YOLOv8x with BB-a.

As BB-a is very different from BB-1 and BB-2, when it comes to image definition and focus, all tests but test 4' were realized. Test 4' was avoided because the results from test 1' confirmed that the initial learning rate was redundant, as once again the results from test 1' were the same as those from test 0'. As for the results of other tests, all tests exposed a certain degree of overfitting, as seen in the box loss scores of several different models. However, mAP scores weren't very different from each other. Additionally, the mAP scores observed were comparable to, and in some cases worse than scores obtained when training either DETR-r50 or DETR-r101 with this data. Ultimately, the best-performing model, once again, corresponded to the one trained in test 2'. Therefore, the same hyperparameter settings were used to train the final model.

As previously referred, in every YOLOv8x training run when training on BB-a, some of the earlier epochs have a non-existent box loss value for validation data. However, after those earlier epochs, every epoch has a real box loss value on validation data, and a box loss curve for validation data can still be traced.

Logs of the tests conducted, as well as graphics depicting cross-entropy loss, $mAP0.5$ and $mAP0.5{:}\,0{:}\,95$, are in the TIOD-results repository, in the following path: "yolov8/bb-a", and Table 21, located in the Appendix, presents training times and best scores corresponding to the conducted hyperparameter tests.

4.4.4. Model Results

Logs of the final models' results, as well as checkpoints of those same models at their best epoch (when it comes to mAP values) can be found at "detr BB-a/r50 final", "detr BB-a/r101 final" and "yolov8/bb-a final" for DETR-r50, DETR-r101 and YOLOv8x respectively.

Table 10 – Results of the training scores from the final models trained with BB-a data.

Final models	DETR – ResNet50	DETR – ResNet101	YOLOv8x
Number of features	41279238	60219142	68153571
Number of epochs	100	105	170
Highest mAP50:95	0,2349	0,2852	**0,4133**
Highest mAP50	0,5483	0,6389	**0,7345**
Best score[(4)]	0,2663	0,3206	**0,4454**
Best epoch	65	83	59
Training time (h:m:s)	0:50:19	00:44:28	00:21:22

[(4)]score formula: $0{,}9 * (mAP0.5{:}\,0.95) + 0{,}1 * (mAP0.5)$

4.5. RESULTS ON BB-B

Just like when training models with BB-a, both data imbalance and overfitting are present when training with standard hyperparameter settings, aside from the varying number of epochs. However, in models trained with BB-b data, the disparity between cross-entropy loss in training data and in validation data is lower than in models trained with BB-a data, when the hyperparameter settings are the same in both models. This adds to the hypothesis that augmented datasets are more prone to overfitting than non-augmented datasets.

The mAP scores were a little bit lower than those obtained when training models with BB-a data, but still significantly higher than scores obtained when training models with BB-1 or BB-2 data. The cross-entropy loss values are also distinctly lower than those of models trained with BB-a data, though they are still high.

When it comes to the tests performed, the basic tests in DETR — 0, 1, 3, and 4 — were all culminating in models with some level of overfitting. To try to avoid that, test 9 was realized, with good results regarding overfitting, on both forms of DETR. When training YOLOv8, only tests 0', 2' and 3' were performed, with possible signs of overfitting being ambiguous in the first two tests but non-existent in the third one.

4.5.1. DETR – ResNet50 backbone

Starting with the usual epoch tests, much like what occurred when training DETR with BB-a, though to a lesser degree, the cross-entropy loss values in validation data started increasing steadily during longer training procedures. This revealed that the model was overfitting, though, from past experiments, this overfitting seemed like something that could potentially be corrected with hyperparameter tuning. Therefore, the chosen epoch number for the hyperparameter tests was **270**, based on the evolution of the mAP scores and the hypothesis that overfitting could still be corrected.

When performing tests 1, 3, and 4, overfitting of the model to training data was observed when training this model, though to a lesser degree when performing test 3. Therefore, test 9 was attempted, leading to satisfying results: though the mAP scores were slightly lower than those of the

model trained with the settings of test 3, the degree of overfitting wasn't critical, and there was no need to reduce the number of epochs.

Logs of the tests conducted, as well as graphics depicting cross-entropy loss, $mAP0.5$ and $mAP0.5{:}\,0{:}\,95$, are in the TIOD-results repository, in the following path: "detr BB-b/r50", and Table 22, located in the Appendix, presents training times and best scores corresponding to the conducted hyperparameter tests.

4.5.2. DETR – ResNet101 backbone

After the usual epoch tests, the ideal number when training this model with BB-b data is **220**. Past 220 any performance increase is negligible or offset by increased overfitting in the model and training time. Overfitting was also evident in most runs, and even when testing with 100 epochs there were small signs of overfitting.

Once again, doubling the amount of attention heads just led to more overfitting and lower mAP scores. However, lowering the initial learning rate to $0{,}000001$ (test 9) presented the best results when it comes to mAP. In this case, not to alter the learning rate any further, much like what occurred when training DETR-r50 with BB-a, the epoch number was limited in the final model: rather than 220 epochs, the final model was trained over **150** epochs, as the cross-entropy loss on validation data didn't see a systematic increase until a few epochs after that, and mAP scores were somewhat consistent under these settings after 150 epochs. Additionally, the best mAP score — $mAP0.5{:}\,0{:}\,95 * 0{,}9 + mAP0.5 * 0{,}1$ — had already been registered at epoch 120.

Logs of the tests conducted, as well as graphics depicting cross-entropy loss, $mAP0.5$ and $mAP0.5{:}\,0{:}\,95$, are in the TIOD-results repository, in the following path: "detr BB-b/r101", and Table 23, located in the Appendix, presents training times and best scores corresponding to the conducted hyperparameter tests.

4.5.3. YOLOv8x

When performing training runs to determine the ideal number of epochs, this model's mAP scores were comparable to both DETR-r50 and DETR-r101 trained with BB-b; these results were very promising, as they corroborated the hypothesis formed when training all models with BB-a data, that DETR can be comparable to YOLOv8. When it comes to the epoch number itself for this model, due to how mAP scores evolved along the multiple training runs the epoch number settled on in this case was **115**.

Knowing the results from tests realized when training YOLOv8x with other blueberry datasets, only tests 0', 2', and 3' were performed, as other tests would, in theory, culminate in poor or useless results. When performing test 3', the same phenomenon that occurred when performing the same test in training with BB-2 data: the reduced batch size led to less overfitting than usual. However, once again, the mAP scores of the corresponding model were significantly lower, which indicated that the settings corresponding to test 3' weren't optimal. However, due to the model trained under the settings of test 2' showing signs of overfitting, the hyperparameters of test 3' were chosen to train the final model.

Once again, in every training run when training on BB-b, some of the earlier epochs have a nonexistent box loss value for validation data. However, after those earlier epochs, every epoch has a real box loss value on validation data, and a box loss curve for validation data can still be traced.

Logs of the tests conducted, as well as graphics depicting cross-entropy loss, $mAP0.5$ and $mAP0.5{:}\,0{:}\,95$, are in the TIOD-results repository, in the following path: "yolov8/bb-b", and Table 24, located in the Appendix, presents training times and best scores corresponding to the conducted hyperparameter tests.

4.5.4. Model Results

Logs of the final models' results, as well as checkpoints of those same models at their best epoch (when it comes to mAP values) can be found at "detr BB-b/r50 final", "detr BB-b/r101 final" and "yolov8/bb-b final" for DETR-r50, DETR-r101 and YOLOv8x respectively.

Table 11 – Results of the training scores from the final models trained with BB-a data.

Final models	DETR – ResNet50	DETR – ResNet101	YOLOv8x
Number of features	41279238	60219142	68153571
Number of epochs	270	150	115
Highest mAP50:95	0,2931	0,2718	**0,3364**
Highest mAP50	**0,6404**	0,6254	0,6042
Best score[(5)]	0,3279	0,3072	**0,3630**
Best epoch	269	101	113
Training time (h:m:s)	00:58:35	00:38:26	00:09:50

[(5)]score formula: $0{,}9 * (mAP0.5{:}\,0.95) + 0{,}1 * (mAP0.5)$

4.6. RESULTS ON BLN

The results obtained when training models with BLN were a tremendous improvement over the results obtained with any of the blueberry datasets. The mAP scores obtained on all models showed that the issue with the very low scores obtained when training with blueberry data resulted from issues with the datasets, and that DETR or YOLOv8 models are viable as predictors once trained with data appropriate for whatever problem requires object detection performed by artificial intelligence.

For this dataset, tests 1-8 were performed, in spite of results obtained when training with other data, as this dataset is drastically different from the others in nature, object size, lighting in the images, and variability. Therefore, the effects of hyperparameters and their combinations could hypothetically be very different. Though, test 2 wasn't repeated for both versions of DETR, as its application on DETR-r50 produced similar results to those obtained when performing the same test on any other dataset.

This time, the results obtained when training DETR were very similar to those obtained when training YOLOv8x, which showed that transformer-based deep learning models can compete with CNN-based models in object detection.

4.6.1. DETR – ResNet50 backbone

Adding to the high mAP values observed, cross-entropy loss was also low on every test on both training and validation data, for both the epoch tests and the remaining ablations. Though this doesn't necessarily mean there was no overfitting, the evolution of these values when conducting epoch tests didn't signal any overfitting. Regarding the proper number of epochs, once again, runs with 100, 200, 300, 400 and 600 epochs were performed, and the results of these tests would show that **300** is the most viable number of epochs, as mAP values ceased to have any meaningful increase beyond this number of epochs.

Since this dataset is so different from all others, all tests apart from Test 9 were performed. The results showed once again that a higher number of attention heads in this model leads to overfitting, and in this case, so does an increased batch size. The results observed show a slight correlation between an increased batch size and better mAP scores, and corroborate some of the previously established hypotheses:

- Increasing the learning rate leads to extremely low (even reaching 0) mAP scores.
- Decreasing the learning rate may decrease the risk of overfitting.

The best hyperparameter settings were those of Test 5, as that test led to the highest mAP scores among all hyperparameter tests, and a model with these settings showed no blatant signs of overfitting.

Logs of the tests conducted, as well as graphics depicting cross-entropy loss, $mAP0.5$ and $mAP0.5:0:95$, are in the TIOD-results repository, in the following path: "detr BLN/r50", and Table 25, located in the Appendix, presents training times and best scores corresponding to the conducted hyperparameter tests.

4.6.2. DETR – ResNet101 backbone

After the usual epoch-related tests — training with 100, 200, 300, 400 or 600 epochs — it was determined that the ideal number of epochs for DETR-r101 to be trained with BLN data is **350**. One important issue to address was that when conducting the ablation tests regarding epochs, the evolution of cross-entropy loss on training and validation data showed that overfitting was occurring when training DETR-r101 on this dataset, though, once again, the cross-entropy loss values themselves were quite low.

The mAP scores were slightly better than those achieved when training DETR-r50 with the same training data, though overfitting was also more present. However, there was no strong evidence of overfitting in any model trained with a learning rate of $0{,}00001$, much like what was observed when training DETR with BB-1 data.

The best hyperparameter settings were those of test 5, as that test led to the highest mAP scores among all hyperparameter tests, and a model with these settings showed no blatant signs of overfitting. Though, when training the model on both the training and validation splits of BLN, the model underperformed when compared to DETR-r50.

Logs of the tests conducted, as well as graphics depicting cross-entropy loss, $mAP0.5$ and $mAP0.5{:}\,0{:}\,95$, are in the TIOD-results repository, in the following path: “detr BLN/r101”, and Table 26, located in the Appendix, presents training times and best scores corresponding to the conducted hyperparameter tests.

4.6.3. YOLOv8x

When training this model with the BLN data, mAP scores were, once again, comparable to those obtained when training DETR-r50 or DETR-r101. This shows that, at least when datasets have only one class and aren’t solely comprised of small/very small objects, the performances of DETR models and YOLOv8x are very alike, at least when it comes to predictive power. The epoch tests showed that the model’s performance regarding predictive power would suffer minimal to no changes when training after about **200** epochs, so this was the chosen epoch number.

Due to the task being different, and the dataset being very different from any of the blueberry datasets, all hyperparameter tests — 0’, 1’, 2’, 3’, 4’, 5’ — were realized when training YOLOv8x with BLN. Tests 1’ and 4’ were once again redundant, showing that, without a doubt, the “lr0” parameter, corresponding to the model’s initial learning rate, is ineffective at least for single class object detection datasets.

Regarding the other tests, the box loss values of training and validation data were very close in all of those tests, with the validation data’s box loss even being lower than the training data’s throughout the model’s training in tests 3’ and 5’. However, the model corresponding to test 2’ still had better mAP scores without any significant traces of overfitting. Therefore, the final model used the same hyperparameter settings as test 2’: every hyperparameter besides the number of epochs and the final learning rate stayed the same, with the new learning rate being $0{,}001$.

Once again, in every training run when training on BLN, some of the earlier epochs have a nonexistent box loss value for validation data. However, after those earlier epochs, every epoch has a real box loss value on validation data, and a box loss curve for validation data can still be traced.

Logs of the tests conducted, as well as graphics depicting cross-entropy loss, $mAP0.5$ and $mAP0.5{:}\,0{:}\,95$, are in the TIOD-results repository, in the following path: “yolov8/balloon”, and Table 27, located in the Appendix, presents training times and best scores corresponding to the conducted hyperparameter tests.

4.6.4. Model Results

Logs of the final models’ results, as well as checkpoints of those same models at their best epoch (when it comes to mAP values) can be found at “detr BLN/r50 final”, “detr BLN /r101 final” and “yolov8/balloon final” for DETR-r50, DETR-r101 and YOLOv8x respectively.

Table 12 – Results of the training scores from the final models trained with BLN data.

	DETR – ResNet50	DETR – ResNet101	YOLOv8x
Number of features	41279238	60219142	68153571
Number of epochs	300	350	200
Highest mAP50:95	0,7060	0,6458	**0,8213**
Highest mAP50	**0,9546**	0,8279	0,9374
Best score[(6)]	0,7308	0,6640	**0,8306**
Best epoch	96	92	150
Training time (h:m:s)	01:23:25	01:34:48	00:15:29

[(6)]score formula: $0{,}9 * (mAP0.5{:}\,0.95) + 0{,}1 * (mAP0.5)$

4.7. About The Datasets

As seen from how all models performed, some issues were present when training with the blueberry datasets. After subpar results were first observed when training all models with BB-1 and BB-2, and relatively very good results were obtained when training all models with BLN, it was clear that these issues stemmed from the BB-1 and BB-2 datasets. Although BB-a and BB-b mitigated these issues, results from training models with those datasets were still underwhelming.

4.7.1. BB-1 and BB-2

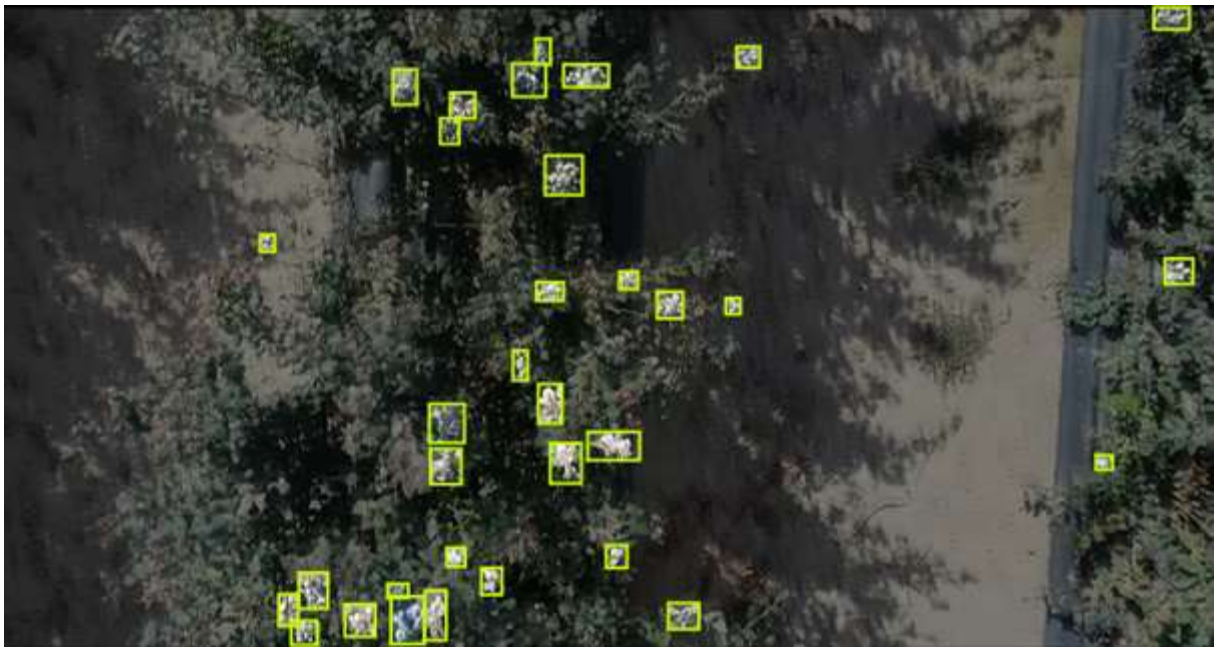

Figure 7 – Annotated training image present in the BB-1 and BB-2 datasets

The images in datasets BB-1 and BB-2 were very challenging to annotate, due to issues such as lighting and definition. Lighting was an issue in two ways: sunlight obfuscating certain areas of images and making objects hard to tell apart, and other areas being too obscured to feasibly identify blueberry

batches. Both these issues could occur in the same image. This was but one of the problems with these datasets.

Data imbalance was also a big issue: As mentioned previously, data objects are very small, and the total area in an image occupied by objects is very small proportionally to the image, as seen in Figure 7.

4.7.2. BB-a and BB-b

These datasets were created to mitigate, or even solve, the problems with the first two datasets. Even as the images are zoomed-in versions of the original images, the resolution in these images wasn't significantly affected. Nonetheless, the transformation did falter in some areas:

- Whilst objects usually aren't densely concentrated in certain areas of an image, it's not uncommon to see that phenomenon in images from this dataset.
- Nothing was done to try to improve the original images' luminosity.
- Finally, both the objects themselves and the total area occupied by them in the image are still quite small, though the disparity in sizes isn't as big as observed in BB-1 and BB-2.

Due to these factors, the models' performances when trained on BB-a and BB-b were still very low. However, mAP values achieved on the validation and test splits were noticeably higher than the corresponding values obtained via training on the BB-1 and BB-2 datasets, which shows that the measures taken were effective.

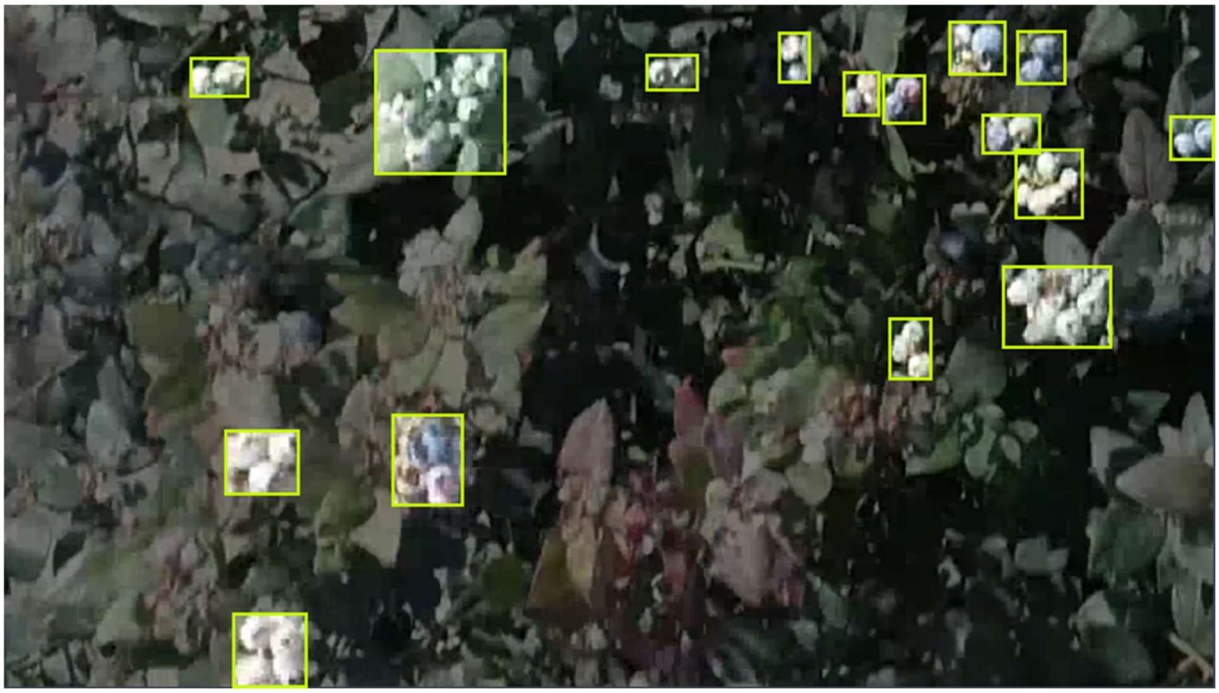

Figure 8 – Annotated training image present in the BB-a and BB-b datasets

As seen in Figure 8, images in the new dataset may still have mostly small objects and the total area within the image occupied by these objects is still small compared to the area of the whole image. The definition isn't as good as in the original dataset, but objects and non-object elements are still mostly perceptible.

4.7.3. BLN

The balloon dataset is open source and already annotated, and the link to it is in the appendix. The scenarios depicted in the images of this dataset are much more diverse, but image quality is better and there are no apparent issues related to lighting when it comes to the images.

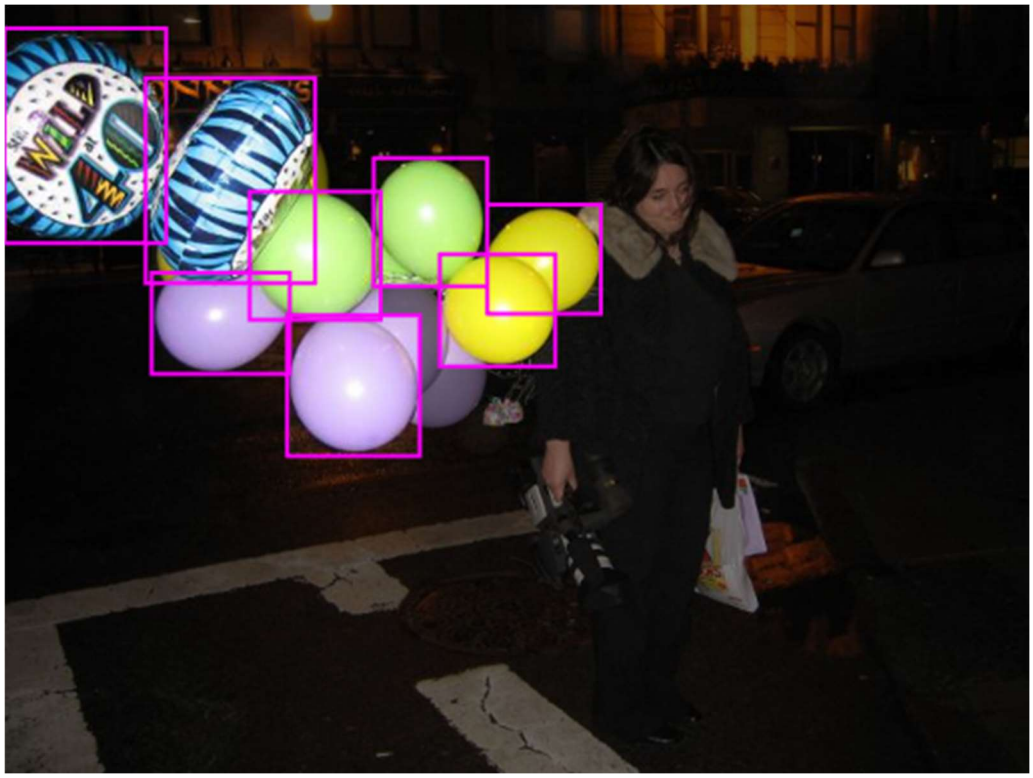

Figure 9 – Annotated training image present in the BLN dataset.

BLN is opposite to the blueberry datasets in two aspects and said aspects are showcased in Figure 7:

- Most objects are large and there are very little to no small objects, which is directly opposite to what occurs in blueberry datasets.
- In most images, objects occupy a lot more space than they do in blueberry plantation images.

The impact of these differences is clear when evaluating the training results, as models trained on BLN have much, much higher scores, and show less overfitting. Though there are overlaps in some cases, as shown in Figure 9, objects are distinct enough to be easily told apart.

5. CONCLUSION

To ascertain whether there may be practical value in researching the use of Transformer-based Deep Learning models in Object Detection, a state-of-the-art CNN-based model — YOLOv8 — was compared with a Transformer-based model — DETR. Within DETR, the comparison between DETR-r50 and DETR-r101, two variants of DETR that only differed from each other in their respective feature extraction backbones, was also made.

To compare DETR-r50, DETR-r101, and YOLOv8x — the strongest variant of YOLOv8 when it comes to predictive power — all these models were trained with four different datasets that represent a real-world problem — the detection of batches of blueberries in a blueberry plantation — and on a fifth dataset — the balloon dataset — that is open-source and relatively simple to analyse. During training, any scenario where any model was overfitting to any kind of training data was avoided. These models were compared on their training times, and the mAP values obtained when training with these 5 datasets, as well as on the mAP values stemming from each model's training with COCO2017 data.

When it comes to the several datasets used in this thesis, all blueberry datasets showed varying levels of data imbalance resulting from several factors, such as the lighting in the dataset's images and the size of data objects. The results obtained when training models with these datasets were very poor for the standards of most object detection problems but were still accounted for when comparing models. Models trained with BLN — that were meant to find balloons in images, of course — performed significantly better within their task than the blueberry batch predictors, showing just how relevant training data is when training any kind of machine learning model.

After performing the several comparisons listed above, the conclusion was that YOLOv8 vastly outperforms either DETR variant when it comes to each model's training time and outperforms DETR when it comes to predictive power in several scenarios, though not by a very wide margin. Additionally, in some scenarios, DETR can outperform YOLOv8 when it comes to predictive power, depending on training data's characteristics: none of DETR's final models outperformed YOLOv8 when it comes to overall mAP scores, though DETR sometimes achieved better mAP values when regarding an IoU threshold of $0{,}5$.

YOLOv8 has been shown to handle small data objects better than DETR, judging from the results obtained when training models with either BB-1 or BB-2 data. However, their predictive power is similar when dealing with medium-sized or large objects. Furthermore, since the loss function used to scour trained YOLOv8 models for any signs of overfitting wasn't built to be an ideally binary one, discrepancies between box loss values corresponding to training data and values corresponding to validation data are more significant. From the box loss values registered in most training cycles, it's concluded that YOLOv8 is more prone to overfitting than DETR. Additionally, DETR showed better $mAP0.5$ vaues than YOLOv8 in several scenarios, showing that DETR may be more powerful under looser IoU thresholds.

Between the two DETR variants, DETR-r50 is less prone to overfitting but DETR-r101 trained models usually have more predictive power. However, the final trained version of either model for each dataset needs to have the proper hyperparameter settings so that the model doesn't overfit, which may negatively impact the model's performance. This may also involve limiting the number of epochs after hyperparameter tuning, which can potentially keep a model from being trained to reach the

highest possible predictive power. Therefore, when it comes to some datasets, training DETR-r50 with a certain batch of training data proved to originate a more powerful model than training DETR-r101 with the same data.

When training the models used in this thesis, it was noticed that lowering the models' respective standard learning rates usually affects the models positively, as models trained using a lower learning rate (initial in DETR-r50 and DETR-r101's cases, final in YOLOv8's case) presented higher mAP scores than their counterparts and/or less evidence of overfitting.

Ultimately, while YOLOv8 is the most powerful model tested in this thesis, as well as being the fastest by a wide margin, it's a model with a tendency to overfit to training data and only in certain cases can hyperparameter settings remedy this. Additionally, DETR is already capable of beating YOLOv8 in some performance tests, despite DETR being a relatively unoptimized model (with either backbone). YOLOv8 is a state-of-the-art model, so DETR being able to rival YOLOv8 in performance shows that the development of Transformer-based models for object detection has great promise, both scientifically and when it comes to practical use of these models.

6. LIMITATIONS AND FUTURE WORKS

6.1. About Real Data

The most significant component in modern Artificial Intelligence is data and this project was yet another example of it: the real case data used proved to be deeply flawed, and while it still produced meaningful insights, it's not an accurate representation of most datasets, at least when it comes to Precision Agriculture. However, it's not a unique dataset: data can come in all shapes and sizes, and real-case datasets are often flawed and incomplete to varying degrees.

The real case data was annotated for this thesis, but there are still other procedures meant to improve object detection data, such as methods different from bounding boxes or techniques to enhance resolution/color contrast, that have not been explored in this project. Additionally, images could hypothetically be even smaller than those in the BB-a and BB-b datasets, though that could also be ultimately impractical, considering that even when blueberry plants bear fruit, most of a blueberry plant's volume is comprised of anything other than its fruits.

6.2. About Transformers in Object Detection

When it comes to DETR, many more iterations couldn't be included in the project due to a lack of resources. Examples of this are the DC5 variants of DETR: in these models, feature resolution is increased by adding a dilation to the last stage of the backbone — which can either be r50 or r101 — and removing a stride from the first convolution of this stage. However, these models are very computationally complex, and the GPUs that could be accessed in this project weren't powerful enough, even considering the possibility of distributing training across multiple GPUs.

Another example is Deformable DETR: a model that adds deformable convolution to enhance feature resolution with less computational cost. However, due to distribution issues with some packages vital to Deformable DETR, as well as certain inconsistencies when it comes to fine-tuning, it was impossible to have a working version of this model for this project.

There are also other Deep Learning models for object detection that incorporate transformer neural networks that aren't versions of DETR or based on DETR, such as the Swin Transformer, a model that uses the shifted windows approach to incorporate the benefits of transformers whilst accelerating the training process. Integrating these models in future research would be a very interesting endeavour and a logical next step for further investigation in Object Detection.

7. REFERENCES

[1] (Vaswani et al., 2017) Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones,L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In Advances in neural information processing systems, 5998–6008.

[2] (Zhang et al.,) Zhang, Aston and Lipton, Zachary C. and Li, Mu and Smola, Alexander J. 2021. Dive Into Deep Learning.

[3] (Collobert & Bengio, 2004) R. Collobert and S. Bengio 2004. Links between Perceptrons, MLPs and SVMs. Proc. Int'l Conf. on Machine Learning (ICML).

[4] (S. Khan et al., 2022) S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," ACM Computing Surveys (CSUR), 2021

[5] (Ba et al., 2015) Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. arXiv preprint. arXiv:1607.06450, 2015

[6] (Bond et al., 2019) Bond, Raymond, Ansgar Koene, Alan Dix, Jennifer Boger, Maurice D Mulvenna, Mykola Galushka, Bethany Waterhouse Bradley, Fiona Browne, Hui Wang, and Alexander Wong, Democratisation of Usable Machine Learning in Computer Vision. arXiv preprint arXiv:1902.06804, 2019.

[7] (A. I. Khan & Al-Habsi, 2020) Khan, A.I.; Al-Habsi, S. Machine learning in computer vision. Procedia Comput. Sci. 2020,167, 1444–1451.

[8] (Zhai et al., 2021) Dosovitskiy, Alexey; Beyer, Lucas; Kolesnikov, Alexander; Weissenborn, Dirk; Zhai, Xiaohua; Unterthiner, Thomas; Dehghani, Mostafa; Minderer, Matthias; Heigold, Georg; Gelly, Sylvain; Uszkoreit, Jakob (2021-06-03). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". arXiv:2010.11929

[9] (Parmar et al., 2018) N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, D. Tran, Image transformer, International Conference on Machine Learning, pp. 4055–4064, 2018

[10] (Guo et al., 2021) Guo, M. H.; Cai, J. X.; Liu, Z. N.; Mu, T. J.; Martin, R. R.; Hu, S. M. PCT: Point cloud transformer. Computational Visual Media Vol. 7, No. 2, 187–199, 2021.

[11] (Yuan et al., 2021) Yuan, L.; Chen, Y.; Wang, T.; Yu, W.; Shi, Y.; Jiang, Z.-H.; Tay, F. E.; Feng, J.; Yan, S. Tokens-to-Token ViT: Training vision transformers from scratch on ImageNet. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, 558–567, 2021.

[12] (Liu et al., 2021) Liu, Z.; Lin, Y. T.; Cao, Y.; Hu, H.; Guo, B. N. Swin transformer: Hierarchical vision transformer using shifted windows. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, 10012–10022, arXiv:2103.14030, 2021.

[13] (Mnih et al., 2014) Mnih, V.; Heess, N.; Graves, A.; Kavukcuoglu, K. Recurrent models of visual attention. In: Proceedings of the 27th International Conference on Neural Information Processing Systems, Vol. 2, 2204–2212, 2014.

[14] (X. Wang et al., 2017) Wang, X. L.; Girshick, R.; Gupta, A.; He, K. M. Non-local neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 7794–7803, 2018.

[15] (Guo et al., 2022) Guo, M.H., Xu, T.X., Liu, J.J., Liu, Z.N., Jiang, P.T., Mu, T.J., Zhang, S.H., Martin, R.R., Cheng, M.M., Hu, S.M.: Attention mechanisms in computer vision: A survey. Computational Visual Media pp. 1–38. 2022.

[16] (Carion et al., 2020) Carion N., Massa F., Synnaeve G., Usunier N., Kirillov A., Zagoruyko S. End-to-end object detection with transformers. In: ECCV. 2020.

[17] (Zhu et al., 2020) X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable DETR: Deformable transformers for end-to-end object detection," arXiv preprint arXiv:2010.04159, 2020.

[18] (Wen et al., 2023) Long Wen, Yu Cheng, Yi Fang, and Xinyu Li. A comprehensive survey of oriented object detection in remote sensing images. Expert Systems with Applications, page 119960, 2023.

[19] (Krizhevsky et al., 2017) A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," Commun. ACM, vol. 60, no. 6, pp. 84–90, 2017.

[20] (Girshick et al., 2014) Girshick, R., Donahue, J., Darrell, T. and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. 2014.

[21] (Uijlings et al., 2013) Uijlings J. R. R., Van De Sande K. E. A., Gevers T., et al. Selective search for object recognition[J]. International journal of computer vision, 104: 154-171. 2013.

[22] (Endres & Hoiem, 2010) I. Endres and D. Hoiem. Category independent object proposals. In ECCV, 2010.

[23] (Redmon et al., 2016) Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. 2016.

[24] (Gu et al., 2018) Gu J., Wang Z., Kuen J., Ma L., Shahroudy A., Shuai B., Liu T., Wang X., Wang G., Cai J., Chen T. Recent advances in convolutional neural networks. Pattern Recognition 77:354–377, 2018.

[25] (Ren et al., 2015) S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," Advances in neural information processing systems, vol. 28, 2015.

[26] (Lin et al., 2015) Lin T. Y., Maire M., Belongie S., Hays J., Perona P., Ramanan D., Dollar P., Zitnick C.L.: Microsoft COCO: Common objects in context. In: ECCV, 2014.

[27] (Pierce & Nowak, 1999) Pierce, F.J. & Nowak, P. Aspects of precision agriculture. Advances in Agronomy 67: 1-85, 1999.

[28] (Hossain et al., 2019) Hossain M. S., Al-Hammadi M., Muhammad G. Automatic fruit classification using deep learning for industrial applications. IEEE Trans Ind Inf 15:1027–34, 2019.

[29] (H. Tian et al., 2020) Tian H., Wang T., Liu Y., Qiao X., Li Y. Computer vision technology in agricultural automation - a review. Inform Process Agric 7(1):1–19, 2020.

[30] (Z. Wang et al., 2018) Wang Z., Hu M., Zhai G. Application of Deep Learning Architectures for Accurate and Rapid Detection of Internal Mechanical Damage of Blueberry Using Hyperspectral Transmittance Data. Sensors. 18(4):1126. 2018.

[31] (J. Wang et al., 2021) Wang, Jinhai & Zhang, Zongyin & Luo, Lufeng & Zhu, Wenbo & Chen, Jianwen & Wang, Wei. SwinGD: A Robust Grape Bunch Detection Model Based on Swin Transformer in Complex Vineyard Environment. Horticulturae. 7. 492. 10.3390/horticulturae7110492. 2021.

[32] Khorana, Rohit. (2023). Low-earth Satellite Orbit Determination Using Deep Convolutional Networks with Satellite Imagery. 10.31224/3005.

[33] (Z. Tian et al., 2019) Tian, Z., Shen, C., Chen, H., He, T.: FCOS: Fully convolutional one-stage object detection. In: ICCV (2019)

[34] (Lin et al., 2017) Lin, T.Y., Goyal, P., Girshick, R.B., He, K., Dollar, P.: Focal loss for dense object detection. In: ICCV (2017)

[35] (Gómez & Karatzas, 2017) L. G´omez, D. Karatzas, Textproposals: a text-specific selective search algorithm for word spotting in the wild, Pattern Recognition 70 60–74 (2017).

[36] (Amit et al., 2020) Amit, Y., Felzenszwalb, P., & Girshick, R. (2020). Object detection. Computer Vision: A Reference Guide, 1-9.

[37] (Padilla et al., 2020) Padilla, R., Netto, S. L., & Da Silva, E. A. (2020, July). A survey on performance metrics for object-detection algorithms. In 2020 international conference on systems, signals and image processing (IWSSIP) (pp. 237-242). IEEE.

[38] (Jaccard, 1901) P. Jaccard, "Etude comparative de la distribution florale dans une portion des alpes et des jura," Bulletin de la Societe Vaudoise des Sciences Naturelles, vol. 37, pp. 547–579, 1901.

[39] (Amit et al., 2002) Amit, Y. (2002). 2d Object Detection and Recognition: Models, Algorithms and Networks. MIT Press

[40] (Felzenszwalb et al., 2005) Felzenszwalb, P., Huttenlocher, D. (2005). Pictorial structures for object recognition. International Journal of Computer Vision 61(1) 55-79

[41] (Rowley et al., 1998) Rowley, H.A., Baluja, S., Kanade, T. (1998). Neural network-based face detection. IEEE Transactions on Pattern Analysis and Machine Intelligence 20(1) 23-38

[42] (Felzenszwalb et al., 2010) Felzenszwalb, P., Girshick, R., McAllester, D., Ramanan, D. (2010). Object detection with discriminatively trained part based models. IEEE Transactions on Pattern Analysis and Machine Intelligence 32(9) 1627-1645

[43] (X. Wang et al., 2020) X. Wang, C. Yeshwanth, and M. Nießner, "SceneFormer: Indoor scene generation with transformers," arXiv preprint arXiv:2012.09793, 2020.

[44] (Santana et al., 2021) A. de Santana Correia and E. L. Colombini, "Attention, please! A survey of neural attention models in deep learning," arXiv preprint arXiv:2103.16775, 2021.

[45] (LeCun et al., 2015) Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, 2015.

[46] (Deng et al., 2009) J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, ImageNet: A Large-Scale Hierarchical Image Database. IEEE Computer Vision and Pattern Recognition (CVPR), 2009

[47] (Fang et al., 2021) Y. Fang, B. Liao, X. Wang, J. Fang, J. Qi, R. Wu, J. Niu, and W. Liu, "You only look at one sequence: Rethinking transformer in vision through object detection," CoRR, vol. abs/2106.00666, 2021

[48] (Dai et al., 2020) Z. Dai, B. Cai, Y. Lin, and J. Chen, "UP-DETR: unsupervised pre-training for object detection with transformers," CoRR, vol. 21 abs/2011.09094, 2020.

[49] (Chaudhari et al.,2019) S. Chaudhari, G. Polatkan, R. Ramanath, and V. Mithal, "An attentive survey of attention models," arXiv preprint arXiv:1904.02874, 2019.

[50] (Hochreiter et al., 1999) S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural computation, 1997.

[51] (Kamilaris & Prenafeta-Boldú, 2018) A. Kamilaris and F. X. Prenafeta-Boldú, "A review of the use of convolutional neural networks in agriculture," J. Agric Sci., vol. 156, no. 3, pp. 312–322, Apr. 2018, doi: 10.1017/S0021859618000436.

[52] (Pérez-Zavala et al., 2018) Pérez-Zavala R, Torres-Torriti M, Cheein FA, et al. A pattern recognition strategy for visual grape bunch detection in vineyards. Comput Electron Agric 2018;151:136–49.

[53] (G. Wang et al., 2017) Wang G, Sun Y, Wang J. Automatic image-based plant disease severity estimation using deep learning. Comput Intell Neurosci 2017;2017:2917536

[54] (Y. Ting et al., 2009) Yuan Ting XC. Information acquisition of cucumber fruit in greenhouse environment based on nearing infrared image. Spectrosc Spect Anal 2009;29(8):2054–8.

[55] (Devlin et al., 2019) Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. ArXiv, abs/1810.04805.

[56] (Ye L. et al., 2019) L. Ye, M. Rochan, Z. Liu, and Y. Wang, "Cross-modal self-attention network for referring image segmentation," in CVPR 2019.

[57] (Sun et al., 2019) C. Sun, A. Myers, C. Vondrick, K. Murphy, and C. Schmid, "VideoBERT: A joint model for video and language representation learning," in ICCV, 2019.

[58] (Ketkar et al., 2017) Ketkar, N. (2017). Stochastic Gradient Descent. In: Deep Learning with Python. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-2766-4_8

[59] (Kingma et al., 2017) Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.~

[60] (Gregor et al., 2015) Gregor, K., Danihelka, I., Graves, A., Rezende, D. & Wierstra, D.. (2015). DRAW: A Recurrent Neural Network For Image Generation. Proceedings of the 32nd International Conference on Machine Learning, in Proceedings of Machine Learning Research 37:1462-1471 Available from https://proceedings.mlr.press/v37/gregor15.html.

[61] (Vanneschi & Castelli, 2019) Vanneschi, L., & Castelli, M. (2019). Multilayer Perceptrons. Encyclopedia of Bioinformatics and Computational Biology.

[62] (Rosenblatt, 1958) F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain, Psychological Review 65 (1958) 386-408.

[63] (Ponce et al., 2019) Ponce CR, Xiao W, Schade PF, Hartmann TS, Kreiman G, Livingstone MS. Evolving images for visual neurons using a deep generative network reveals coding principles and neuronal preferences. Cell. 2019;177(4):999–1009.

[64] (Bengio et al., 2011) Bengio, Y.: Deep learning of representations for unsupervised and transfer learning. Unsupervised and Transfer Learning Challenges in Machine Learning, Volume 7 p. 19 (2012).

[65] (Reyes et al., 2015) Reyes, A.K., Caicedo, J.C., & Camargo, J.E.. Fine-tuning Deep Convolutional Networks for Plant Recognition. Conference and Labs of the Evaluation Forum. (2015).

[66] (Too et al., 2019) E. C. Too, L. Yujian, S. Njuki, and L. Yingchun, "A comparative study of fine-tuning deep learning models for plant disease identification," Comput Electron Agric, vol. 161, pp. 272–279, 2019, doi: https://doi.org/10.1016/j.compag.2018.03.032.

[67] (Pan & Fellow, 2009) Pan, S.J., Fellow, Q.Y., 2009. A Survey on Transfer Learning, pp. 1–15.

[68] (Oksuz et al., 2019) Oksuz, K., Cam, B.C., Kalkan, S., & Akbas, E. (2019). Imbalance Problems in Object Detection: A Review. IEEE Transactions on Pattern Analysis and Machine Intelligence, 43, 3388-3415.

[69] (Singh et al., 2019) B. Singh, M. Najibi, and L. S. Davis, "Sniper: Efficient multi-scale training," in Advances in Neural Information Processing Systems (NIPS), 2018.

[70] (Liu et al., 2018) S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[71] (Lin et al., 2017) T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[72] (Ghiasi et al., 2019) G. Ghiasi, T. Lin, R. Pang, and Q. V. Le, "NAS-FPN: learning scalable feature pyramid architecture for object detection," in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

[73] (H. Gu et al., 2023) Gu, H., Dong, H., Konz, N., & Mazurowski, M.A. (2023). A systematic study of the foreground-background imbalance problem in deep learning for object detection. ArXiv, abs/2306.16539.

[74] (Hopfield, 1982) Hopfield J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences of the United States of America, 79(8), 2554–2558. https://doi.org/10.1073/pnas.79.8.2554

[75] (S. Cajal, 1899-1904) Textura del sistema nervioso del hombre y de los vertebrados : estudios sobre el plan estructural y composición histológica de los centros nerviosos adicionados de consideraciones fisiológicas fundadas en los nuevos descubrimentos / [Santiago Ramón y Cajal]. Wellcome Collection. Source: Wellcome Collection.

[76] (C. Wang et al., 2022) Wang, C., Bochkovskiy, A., & Liao, H.M. (2022). YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors. 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 7464-7475.

[77] (Koyun et al., 2022) O. C. Koyun, R. K. Keser, ̇I. B. Akkaya, and B. U. Töreyin, "Focus-and-detect: A small object detection framework for aerial images," Signal Process. Image Commun., vol. 104, 2022, Art. no. 116675.

[78] Jocher, G., Chaurasia, A., & Qiu, J. (2023). Ultralytics YOLOv8 (Version 8.0.0). Retrieved from https://github.com/ultralytics/ultralytics

APPENDIX – RESULTS TABLES

Table 13 – DETR-r50 trained on BB-1 training data (400 epochs runs).

Test	Training time	Best epoch	Best score[7]	Overfitting?
0	03:12:14	365	0,0685	No
1	03:13:43	324	0,0612	Yes
2	03:29:14	0	0,0000	No
3	03:13:06	358	0,0840	No
4	02:23:46	357	0,0710	Yes
5	02:23:26	335	0,0654	No
6	03:27:19	153	0,0330	Yes
7	02:30:30	278	0,0584	Yes
8	02:28:54	337	0,0230	No

[7]score formula: $0{,}9 * (mAP0.5{:}\ 0.95) + 0{,}1 * (mAP0.5)$ – this formula applies to all other tables in this section

Table 14 – DETR-r101 trained on BB-1 training data (300 epochs runs).

Test	Training time	Best epoch	Best score	Overfitting?
0	02:55:46	289	0,0627	Yes
1	02:55:37	171	0,0514	Yes
2	02:53:15	0	0,0000	No
3	02:54:08	158	0,0761	No
4	02:21:17	291	0,0573	Yes
5	02:22:04	296	0,0650	No
6	02:45:33	176	0,0403	Yes
8	02:27:38	162	0,0203	Yes

Table 15 – YOLOv8 trained on BB-1 training data (190 epochs runs).

Test	Training time	Best epoch	Best score	Overfitting?
0'	00:23:28	141	0,1148	Unclear
1'	00:23:38	156	0,1148	Unclear
2'	00:23:20	114	0,1044	Yes
3'	00:27:29	141	0,1123	Unclear
4'	00:24:14	156	0,1044	Yes
5'	00:22:23[(8)]	80	0,1106	Unclear

[(8)]training stopped early at 160 epochs

Table 16 – DETR-r50 trained on BB-2 training data (350 epochs runs).

Test	Training time	Best epoch	Best score	Overfitting?
0	01:14:51	326	0,0517	No
1	01:15:12	278	0,0418	Yes
2	02:08:36	46	0,0004	No
3	01:15:50	336	0,0571	No
4	00:59:53	307	0,0351	No

Table 17 – DETR-r101 trained on BB-2 training data (400 epochs runs).

Test	Training time	Best epoch	Best score	Overfitting?
0	01:41:45	372	0,0534	Yes
1	01:43:49	395	0,0508	Yes
3	01:43:01	355	0,0585	No
4	01:25:34	346	0,0441	No

Table 18 – YOLOv8 trained on BB-2 training data (100 epochs runs).

Test	Training time	Best epoch	Best score	Overfitting?
0'	00:08:38	90	0,1435	Unclear
2'	00:08:24	87	0,1345	No
3'	00:08:17	97	0,0892	No
5'	00:08:20	91	0,0807	No

Table 19 – DETR-r50 trained on BB-a training data (350 epochs runs).

Test	Training time	Best epoch	Best score	Overfitting?
0	04:09:47	347	0,1897	Yes
1	03:03:55	208	0,2014	Yes
2	04:52:43	57	0,0038	Yes
3	05:42:46	324	0,2282	Yes
4	03:20:01	178	0,1948	Yes
9	04:38:21	209	0,2256	Yes

Table 20 – DETR-r101 trained on BB-a training data (105 epochs runs).

Test	Training time	Best epoch	Best score	Overfitting?
0	01:15:36	80	0,2231	Yes
1	01:38:36	90	0,2100	Yes
3	01:29:19	89	0,2554	No
4	01:13:26	55	0,2308	Yes
5	00:42:21	67	0,2645	No

Table 21 – YOLOv8 trained on BB-a training data (170 epochs runs).

Test	Training time	Best epoch	Best score	Overfitting?
0'	00:20:46	151	0,2098	Unclear
1'	00:20:35	151	0,2098	Unclear
2'	00:20:35	148	0,2172	Unclear
3'	00:22:37[(9)]	46	0,2356	Yes
5'	00:22:16[(10)]	139	0,2141	Yes

[(9)]training stopped at 167 epochs [(10)]training stopped at 160 epochs

Table 22 – DETR-r50 trained on BB-b training data (270 epochs runs).

Test	Training time	Best epoch	Best score	Overfitting?
0	01:23:40	254	0,2106	Yes
1	00:51:06	248	0,2006	Yes
3	01:23:22	168	0,2520	Yes
4	01:02:10	129	0,1919	Yes
9	00:50:29	151	0,2164	No

Table 23 – DETR-r101 trained on BB-b training data (220 epochs runs).

Test	Training time	Best epoch	Best score	Overfitting?
0	00:48:55	215	0,2251	Yes
1	00:48:51	132	0,2067	Yes
3	00:49:43	102	0,2740	Yes
4	00:39:33	64	0,2303	Yes
9	00:48:56	120	0,2310	Yes

Table 24 – YOLOv8 trained on BB-b training data (115 epochs runs).

Test	Training time	Best epoch	Best score	Overfitting?
0'	00:09:07	109	0,2297	Unclear
2'	00:08:49	41	0,2705	Unclear
3'	00:08:31	113	0,1555	No

Table 25 – DETR-r50 trained on BLN training data (300 epochs runs).

Test	Training time	Best epoch	Best score*	Overfitting?
0	01:12:18	268	0,7715	Unclear
1	01:13:12	132	0,7488	No
2	01:12:26	0	0,0035	No
3	01:12:53	258	0,7639	No
4	00:57:35	289	0,7944	Yes
5	00:58:00	121	0,8195	Yes
6	01:02:41	180	0,7656	Unclear
7	01:00:13	288	0,7516	Yes
8	00:59:47	222	0,7241	Yes

Table 26 – DETR-r101 trained on BLN training data (350 epochs runs).

Test	Training time	Best epoch	Best score*	Overfitting?
0	01:39:14	312	0,7786	Unclear
1	01:41:17	312	0,7537	Unclear
3	01:40:48	142	0,8091	Unclear
4	01:24:40	302	0,7931	Yes
5	01:41:38	193	0,8199	No
6	01:23:54	329	0,7334	Unclear
8	01:27:43	341	0,7639	No

Table 27 – YOLOv8 trained on BLN training data (200 epochs runs).

Test	Training time	Best epoch	Best score*	Overfitting?
0'	00:14:56	96	0,7963	No
1'	00:14:46	96	0,7963	No
2'	00:15:04	150	0,8306	No
3'	00:16:16	155	0,7261	No
4'	00:15:32	150	0,8306	No
5'	00:17:06	155	0,7207	No

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa