DETECTING STOMATA WITH COMPUTER VISION FOR PLANT BREEDING

W.K. VISSERS

FEBRUARY 2023

SUPERVISOR
DR.IR. C. SALM

CRITICAL OBSERVER
T.E. VAN DEN BERG MSC

DEPARTMENT OF EEMCS
FACTULTY OF ELECTRICAL ENGINEERING, MATHEMATICS AND COMPUTER SCIENCE

UNIVERSITY OF TWENTE.

Abstract

Climate change is a growing concern and is affecting global food security. To address these issues, precision agriculture is emerging as a crucial research field. This field involves listening to the plants and meeting their direct needs. Stomata, the microscopic pores in plant leaves, play a vital role in regulating gas exchange, enabling the uptake of carbon dioxide and the release of oxygen during photosynthesis. The response of stomata to certain stimuli can be analysed using a microscope on live plants, with this knowledge better plants can be bread based on specific stomatal features. However, counting stomata is currently a time-consuming task that can be automated with the use of object detection models. The study focuses on automating this process using models from the yolov7 family, including small, medium, and large models. The study aimed to determine the best model based on various training methodologies and tested all models on unseen images from three different species. The results indicate that both the yolov7x (large) and yolov7-tiny (small) models performed well in detecting stomata. The yolov7x (large) model achieved a precision of 88.9% and recall of 75.1% on all test images, and a precision of 98.1% and recall of 95.5% on the most trained species, chrysant. On the other hand, the yolov7-tiny (small) model had a precision of 88.4% and recall of 80.3% on all test images, and a precision of 96.9% and recall of 95% on the chrysant test set. However, it is worth noting that the performance of the yolov7x model dropped when analysing tomato images, which contained the smallest stomata among all test images. Yolov7x proves to be a well capable model for the task of stomata detection, especially with chrysant. Future development should focus on increasing performance on tomato and cucumber.

Contents

List of figures

List of Tables

Chapter 1 – Introduction

The current climate is changing, the seasons are becoming more extreme. Droughts are more occurring on a yearly basis and pose a serious threat for the agricultural sector [1]. For farmers crop loss is a serious issue, and food scarcity is becoming a more serious problem. Within 27 year the global food production can be downed as much as 18% [2].

This project is executed for the 4TU Federation, an initiative from four universities in the Netherlands (Delft, Eindhoven, Twente, Wageningen). The main goal of this team is to develop 'vegetation-integrated, energy harvesting, autonomous sensors' [3]. This sensor is a set goal of plantenna to address multiple challenges the agricultural sector is facing or will be facing in the future.

A critical part of this team is to fully analyze and understand plant behavior into microscopic detail. One observable aspect of a plants leaf with a microscope are stomata on the leaf's epidermis (leaf surface) [4]. Stomata play a vital role in the gas exchange of a plant and are critical for photosynthesis. Understanding and analyzing these processes trough a quantitative manner will contribute towards the real goal of the plantenna team.

To be more precise the project is executed to aid van den Berg in his research. Currently there is a lot of research on the stomatal response from different influences, such as drought an Co^2 avalability. [41]. Van den Berg is also conducting such research and seeks for aid in stomata detection to speed up the process.

The team has already developed a smart microscope system that is able to take images from a life leave sample. By simulating a day night cycle, the kinematic behavior of stomata can be analyzed. The microscope automatically takes images in a set interval from the leaf's surface. As one can imagine this creates a gigantic number of images, which all must be annotated by hand by a researcher for stomata. This can become a real time-consuming chore [5]. An automated system can increase the research output, where only manual checking is needed. In this way stomatal research can speed up, and work quicker towards a solution for the emerging environmental changes.

The goal of this project is to create an automated system which can detect stomata. By having such system, the researcher can focus again on interpreting and analyzing the results and processes more images more easily. The research output can drastically increase, which benefits society, as stomatal research gives meaningful insights in the plant's internal behavior.

Research questions

Before the project starts a set of research questions are made. These questions will aid the 'design' process and set real quantitative goals. The questions are following a certain flow, the first question focus on determining the method of use for stomata detection. The second question center more on the actual execution of the solution found for the first question. And the final question really focusses on possible improvements, where the effect of combining multiple solutions is questioned.

The research questions are defined as the following:

- How can existing knowledge on computer vision techniques be used to create a real time automatic stomata detection algorithm?
- Can a deep learning algorithm be made that is able to detect stomata on different plant breeds with a precision of 95%
- Can earlier knowledge of the leaf's surface reduce the number of false positive detected stomata?

Chapter 2 - Background research

2.1 - Stomata

Understanding what a stoma looks like, what its function is and how it opens is essential, without this knowledge manual annotating images is not possible.
Stomata are one of the most researched parts of a plant. It is universally known that the main function of a stoma is to exchange gas from the inside of the leaf to the outside, and vice versa [6]. H_2O CO_2 and O_2 are exchanged and play a vital role in transpirations, respiration, and photosynthesis. The gas intake and outtake are regulated by the stoma, which is achieved by opening and closing of the two guard cells, which form the stoma [7] . stomata are accountable for 95% of the water loss of a plant. If a stoma is opened, water can evaporate on the leaf surface which causes a negative pressure inside the plant, to return to the steady state, water from the leaf stem and eventually water from the roots will replace the evaporated water. This evaporation of water makes it possible for the plant to move water and nutrients from its roots via the stem towards its leaves. In other words, stomata control the flow of water out of the leaf, which can be beneficial during drought.
This exchange of gasses is only possible due to the structure of stomata, which are only observable with a microscope [4]. Every stoma is built according to the exact same blueprint, two guard cells surround the stomatal pore whose shape looks like a tiny sausage. If the ion concentration inside the guard cell rises, water starts to flow into these cells. This causes the cells to swell and bend, which results in an O shape. Now the stomatal pore is completely exposed, and gases can flow in and out. The stomata closes again if the cells start to pump out the ions, which causes the water to leave the cells. The guard cells shrink and the overall shape changes more to a letter I. Figure 2–2–1 shows the the shape of stomata and the basic functioning.

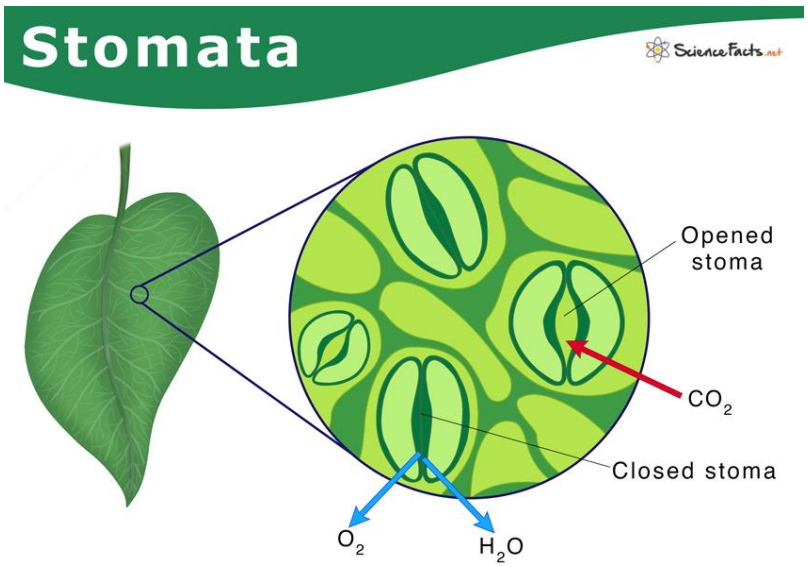


Figure 2–2–1: Basic functioning of stomata [8]

The shape of a stomata is clearly distinguishable from other neighboring cells. However, the differences can still be minimal, especially if a microscope has a different level of magnification. Stomata also differ in sizes and shapes between plant species [9]

In the biological research field there is one plant specie set as the research benchmark specie which is the Arabidopsis [10]. This specie is also known for its recognizable shape and stomatal features. The stoma is nicely rounded and opened the shape really comes close to an O. Another specie with a close stomatal resemblance to the Arabidopsis is the chrysant. The stoma from this specie shows a close resemblance, like shape and size. This specie is mainly used in this project, alongside tomato. The stomata from tomato are a few times smaller.

2.2 - From machine learning to object detection

This chapter will describe the most important concepts concerning this project. The chapter starts with machine learning and how data is processed will be covered. Classification and regression will be explained. Eventually the subdomain of Machine Learning namely deep learning will be covered. The chapter strives to give enough background information, to ensure readers understand all concepts used in this project.

2.2.1 - Machine learning

There are a lot of real-world problems that can be solved with a certain step wise algorithm. Nonetheless there are also a dozen of problems which cannot be solves with these algorithms. The input and the desired output are known, however there are no real algorithms which can reach that desired output. For these problems, no code can be written with classical coding methods. Machine learning is the field that tackled this problem. By learning and recognizing certain patterns, the desired output can be reached. However, for this a lot of data is needed and training is needed to come to the desired output.

To tackle the problems of machine learning a lot of data is needed. This data is sorted in three sets: training-, validation- and testing-sets. If you test and validate your algorithm with your training the result will of course be always perfect, since it trained wit this data. Therefore, different test and validation data is needed to get a fair result of performance.

2.2.2 - Classification and regression

For both classification and regression problems the goal is to find the best possible rule or function that maps the input into a useful output. For classification this is a discrete value which correspond to a certain class. These classes are usually represented with a label. One of the most known classifications in the machine learning field is the written MNIST dataset [11] here the images are bound to certain labels, which in this case is the actual number on the image.

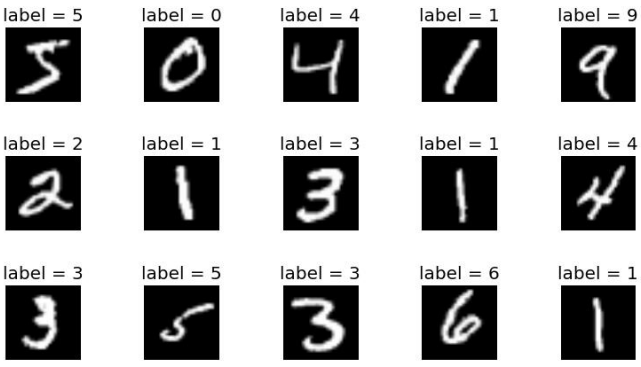


Figure 2–2: Classification from MNISt dataset

A machine learning model which solved a classification problem can be evaluated with different values. The most used method is the accuracy, which represents the percentage of right classified results from the model.

Another evaluations method is to look at precision and recall. To clarify these concepts a few terms are explained a bit more. A model can have multiple results based on what the actual class was. There are true positives (TP), which means the input is correctly classified. Then there are false positives (FP) here the input is classified as a class that in reality is not that class. The other way around False Negative (FN) is when is part of that class but classified as non-class. True negative (TN) is when a non-class input is classified as non-class. Figure 2–2–1 gives a visual representation.

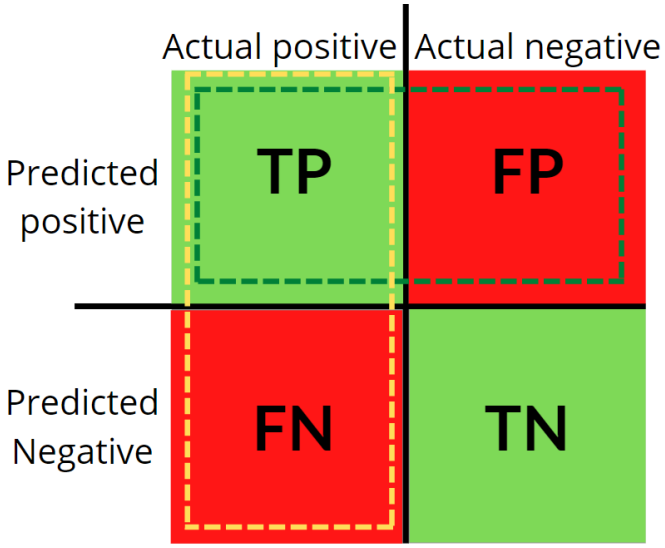


Figure 2–3: Confusion matrix- precision green – recall yellow

With this knowledge precision and recall can be calculated with two simple formulas

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Precision can be written as: The percentage of found instances that are part of this class. And recall as: the percentage of all existing instances of a class that is correctly classified.

However, this brings us the downside of these two metrics, a high precision does not mean a high recall. You can prioritize detecting high true positives over false positives, but this means that false negatives are not considered. Resulting in a high precision but an overall low recall.
Therefore, the F1 score exists to find a balance in precision and recall. It is a value that can be calculated with precision and recall, which is F1 score. The score is calculated with the following formula:

$$F1\ score = \frac{2 * precision * recall}{precision + recall}$$

2.2.3 - Computer vision

Computer vision is a technology that dates back towards the early years of modern computing. Here David Marr [Marr2010] laid the foundations of ‘low-level’ computer vision in 1978. With the focus on edge detection and segmentation. Before the exact process of the traditional image processing workflow is shown, some characteristics of images are handled.

Digital Images are in essence a big matrix of values that is passed to the computer which displays it in the screen. This matrix can be seen as a function of I(x,y) where I is the pixel intensity and x,y are the coordinates of the pixel.

There are different ways to visualize or save an image, this variation is caused by the actual information in the pixel. One can safe only a binary map, which is just a simple one or zero. Grayscale is a representation on a single scale from 0-255. And images saved with rgb values have 3 digits ranging from 0-255. Figure 2–4: RGB representation of an imageFigure 2–4 shows this representation

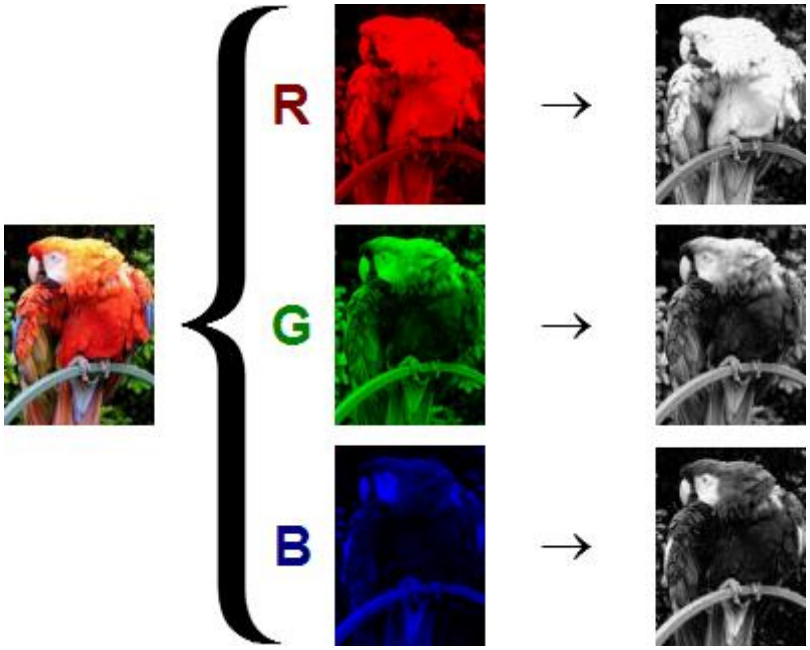


Figure 2–4: RGB representation of an image [12]

2.2.3 - Deep learning

Deep learning is a subdomain of machine learning, which is becoming has become a popular approach for machine learning. Deep learning has many closely related definitions, however only one will be stated. From [13] deep learning is defined as: “Deep learning is a set of algorithms in machine learning that attempt to learn in multiple levels, corresponding to different levels of abstraction. It typically uses artificial neural networks. The levels in these learned statistical models correspond to distinct levels of concepts, where higher-level concepts are defined from lower-level ones, and the same lower-level concepts can help to define many higher-level concepts.”.
Although there are multiple descriptions there are two clear key aspects: (1) Models consist of

multiple layers, which process information nonlinear and (2) Supervised and unsupervised learning of features happens at higher, more abstract layers.
Deep learning is a relatively quite new technology in the field of machine learning. One of the earliest breakthroughs in research of deep learning dates to 2006 [6]. From this research the conclusion was drawn that there should be more research into deep learning, however the researchers already mentioned the success of a multi-layer approach.

The current success of deep learning can be explained by multiple aspects. Throughout the years computational power from computer have drastically improved, speeding up training times. Secondly there is a lot of data available, with this digital age, we humans create a lot of data which can be used as training input. And finally, advancements made in machine learning overall, better models enable more efficient training. Al these aspects contributed to enormous advancements in the technology.

Deep learning is in essence the implementation of neural networks. A neural network is a simple mathematical data transformation in a layered fashion. Every layer in a neural network has a few parameters, which are called the weights. These weights will transform the data as it goes through the layers, and by changing these weights the network learns. But change in a single weight can have influence in all the following layers. Therefore, it is quite hard to calculate the influence of a single weight change.

A so-called loss function if introduced to indicate how well a network is performing. The function will consider the network prediction and the target. How bigger the loss function, how further away the predictions are from the target. To let the network, learn the weights will be changed to minimalize the loss function. The changing of weights is a task for the optimizer, which will implement a backpropagation algorithm. See Figure 2–5Learning process from neural network below to see the general process.

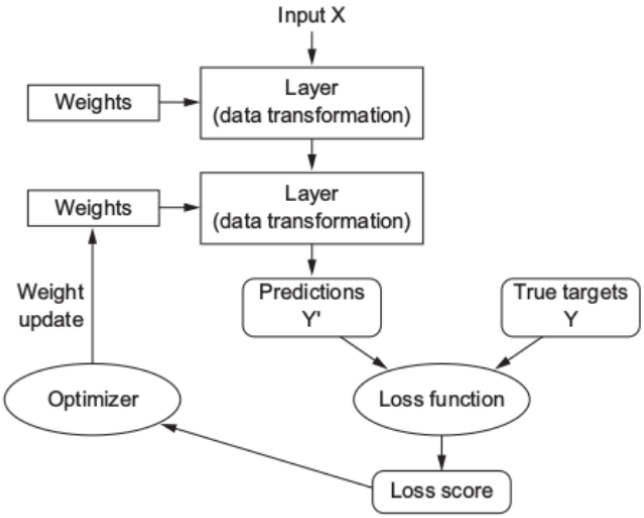


Figure 2–5Learning process from neural network [14]

2.2.4 - Convolutional neural networks

Convolutional neural networks are the most used deep learning technique in the domain of computer vision. Of course, these networks are also applied in other fields, since this paper concerns computer vision, the concept will be explained within this domain.

2D images need a way to be represented as input for the whole network. This is done by converting the image into a 3D tensor. In essence a 2d image has 3 channels namely the R, G and B channels. These channels are overlapping for every pixel location, since only together they can form the desired color in the pixel. Figure 2–6 shows a visual representation of a three channel tensor.

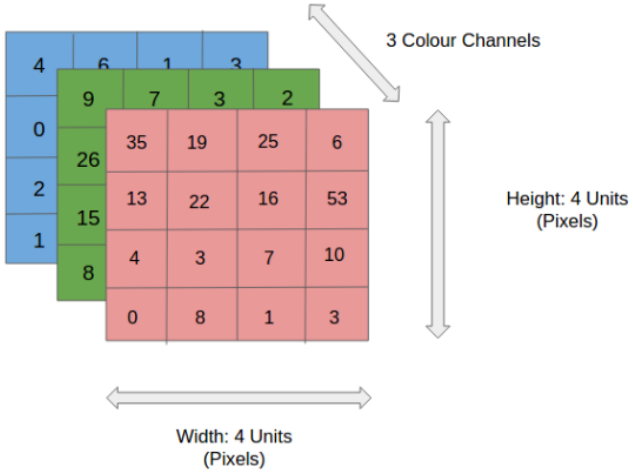


Figure 2–6: 4X4X3 RGB Image [15]

The tensor can be transformed into an array which thereafter can be used as an input for the network. In this way global patterns can be recognized. However, this has one downside, when the image becomes larger the array size increases, which is not scalable. Additionally, an image can have many of the same patterns, a line or curve can be existent on many parts of the picture. Therefore, to solve this problem a smaller segment will slide over the tensor.

The basis of convolutional neural networks is the convolutional operation. As already described an image is stored as a 3D tensor, also known as a feature map. A convolutional operation will get this feature map as input, and returns an output, which is also a feature map. A second 3D tensor which is generally way smaller than the original image tensor, while go trough the feature map. This tensor is called kernel or filter. This filter will slide over the full width and length of the feature map, this is called stride and is usually equal to a value of 1. On every location the product of the filter and tensor and will be saved on the corresponding location of the feature map. This will create a 2-dimensional tensor which will indicate where the input will correspond the most to the pattern of the filter.

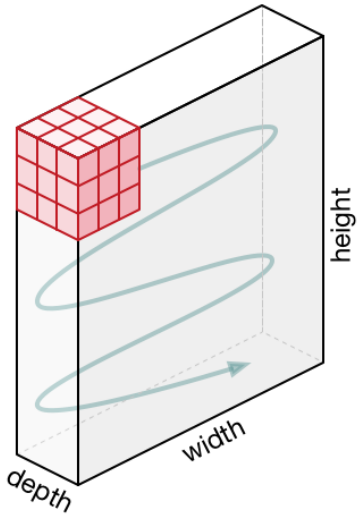


Figure 2–7: Movement of kernel trough feature map [15]

By using multiple filters different patterns can be detected. The values of the filter, and thus specific patterns will be changed during training, therefore is feature recognition a real training process. There is an option to keep the length and width of the feature map the same, which is padding. Padding creates extra rows and columns to the filter.

By having more convolutional operations in sequence, more abstract features can be recognized. After every convolutional layer a feature map is generated, which will have information about lines or corners. More abstract patterns will be created if these prior convolutional layers are used as input. Multiple lines and specific corner can be part of an eye for instance.

The feature maps are still quite big, and this becomes computational demanding if you are some layers deep. Therefore, pooling is introduced, pooling will drastically reduce the computational need. Additionally pooling will also ensure that later layers will have information over a bigger portion of the input. Pooling will generally be a layer within the network, where the filter behaves like prior layers, only the filter will let trough a specific value for a stride (2x2 for example). With this pooling the length and width will be halved for this layer. There are multiple types of pooling, like max pooling, which only lets trough the maximal value. Another type of is average pooling which takes the average value for the stride.Figure 2–8 shows these types of pooling and how it reduces a tensor.

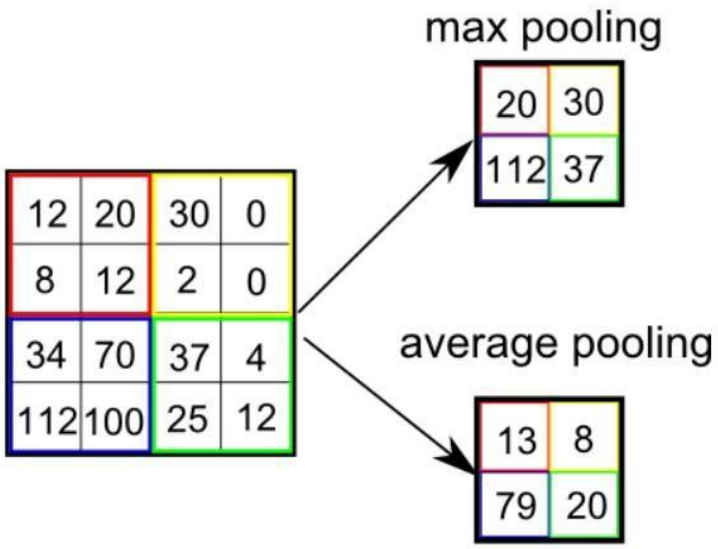


Figure 2–8: Types of pooling

Generally, a convolutional neural network will exist from multiple convolutional layers and pooling layers which will get features out of the image. These features will eventually end up in an 1D tensor, which will be the input for the fully connected layers. These layers will use the found features for the classification and regression tasks. Figure 2–9 shows a rough summary of convolutional neural networks in one picture.

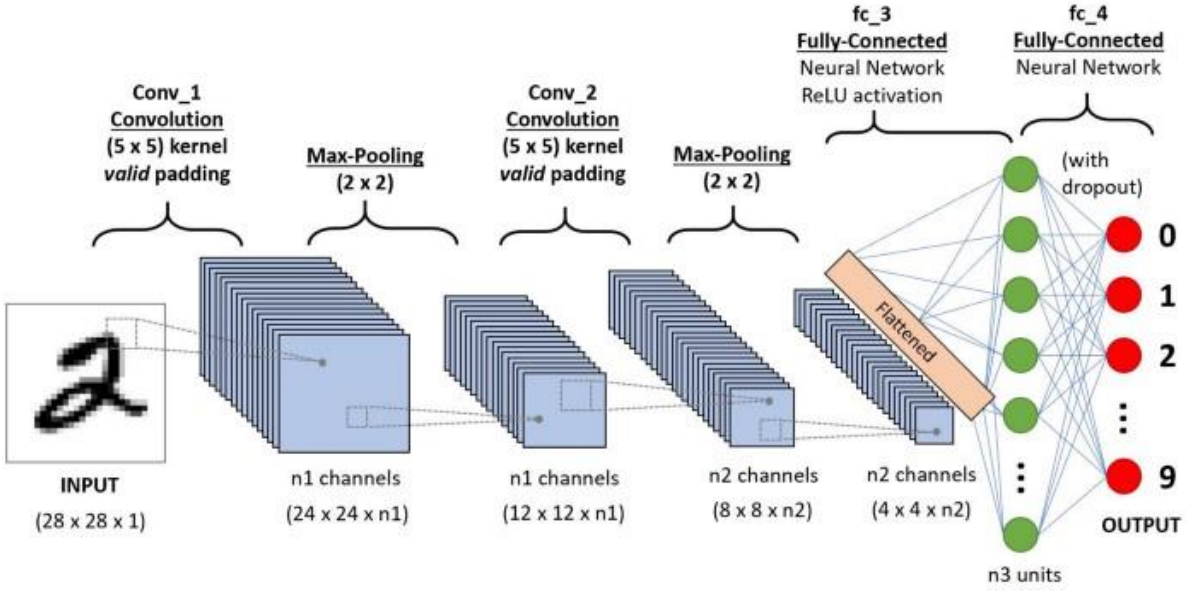


Figure 2–9: A CNN sequence to classify hand written digits [15]

2.2.5 - Applications and architectures

In the field of Convolutional neural networks there a dozen of applications. A few examples are object detection, translation, speech recognition, automated chatbots, self-driving cars, text and image

generation. In short when there is a lot of data available, a network can be trained and can recreate this with its own interpretation of the data.
Every application has its own network approach which works the best for that specific application. Therefore, a lot of architectures have been created, which al have their own strengths and weaknesses. [26] this website gives a lot of insight in CNN methods a how many papers are written with the specific method. A few of these architectures are picked and explained below.

LeNet [16]**:** LeNet is one of the first convolution neural network architectures, it was developed by YannLeCun. The architecture was created to classify handwritten number and postal codes. Many explain convolutional neural networks with the 'simpler' LeNet architecture.

AlexNet [17] **:** This architecture is bases on LeNet only this is a way bigger and deeper version. The architecture competed in the *ImageNetILSVRC challenge* [18], where is placed itself first place. AlexNet differentiate itself by allowing layers be be connected without any pooling between the layers.

GoogleLeNet [19] **:** GoogleLeNet is developed by a google employee Christian Szegedy, and won the first price in the *ImageNetILSVRC challenge* [18] from 2014.The architecture was more recourse efficient, which meant a wider and deeper network was created with the same computational complexity as LeNet. The used method was inception layers also called *Inceptionv1*, the main idea having layers in with which all are connected to a single pooling layer. Later version improved the inception method.

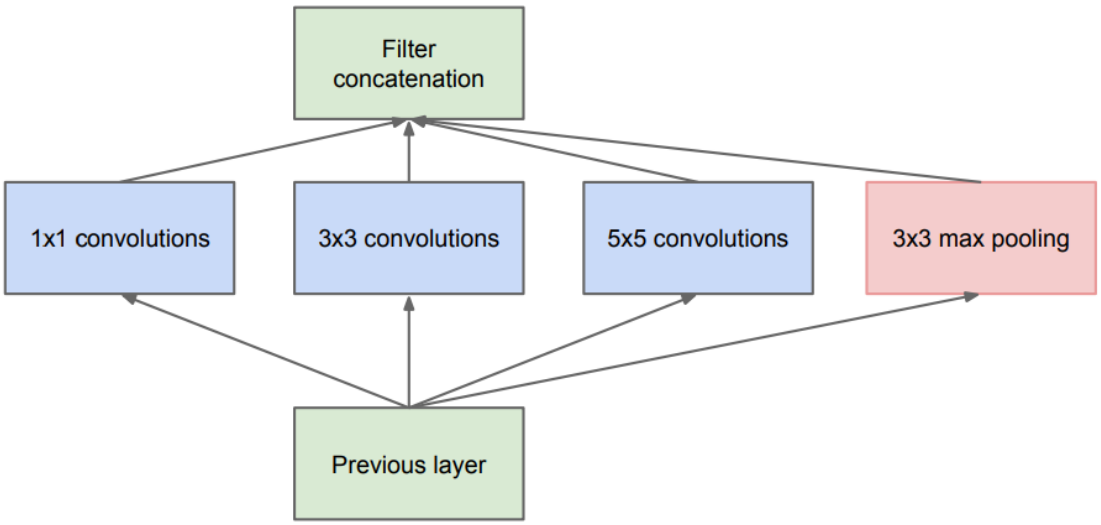


Figure 2–10: Google LENET interception [12]

VGGNet [20]**:** VGGNet achieved second place in the *ImageNetILSVRC challenge* [18] from 2014. It was developed by Karen Simonyan and Andrew Zisserman. The architecture is built wilt depth as its backbone. The original VGGNet has 16 layers (VGG16) and also has a later developed VGG(19) which has 19 layers. The idea was that more layers give a better result, which was also the case.

ResNet [21] **:** ResNet (Residual Neural Network) won the *ImageNetILSVRC challenge [8]* [18] from 2015, and is developed by Kaiming He. This network uses skip connections and batch normalizations, it also does not have any fully connected layers at the end.

DenseNet [22]**:** The developers from densenet notices that every CNN became more and more deeper, which started to cost more computations. Therefore, they developed a new architecture, which connect every layer with any other layer that follows. This feed-forward layering causes the number of connections to become $\frac{L(L+1)}{2}$. This means that for every layer the feature maps from all preceding layers are used as an input.

2.2.6 - Learning and optimalisation

The above-mentioned networks still need a way to learn. This is generally done by changing weights. These weights are changed to minimalize a certain *loss function.* The most used methods to do this is with gradient descent.
A crucial part of gradient descent is the loss function; therefore this will be briefly explained. In essence the loss function is a number which shows how far away the network output lies from its target. This is done with the weights assigned to the connections, which can be denoted as $f(x, W) = y'$

Where x is the current input and W are the weights, y' is the output of the network.

Consequently, the loss function is the output of a function with the target y and output y'

$$f(y', y) = loss\_function$$

Since only the only variables in the network are the weights the loss function can be defined as a function with its only input as weights

$$f(W) = loss\_function$$

Every operation done in a neural network can be derived, since it is all calculations of a function. This means that the derivative of the loss function can be taken. Now the loss function can be simplified and can show on a certain point whether it is rising or descending. With this the needed input to let the loss function decrease can be derived, in other words *Gradient descent.* An important aspect of gradient descent is the size of steps weights are changed; this is also called the *Learning rate*. If the learning rete is too small, it will take quite a lot of steps to reach the minimal value. Additionally, only a local minimum can be found. On the other hand, with too big step the size the global minimum can be skipped. Figure 2–11 shows visually what the effects of a bad learning rates are.

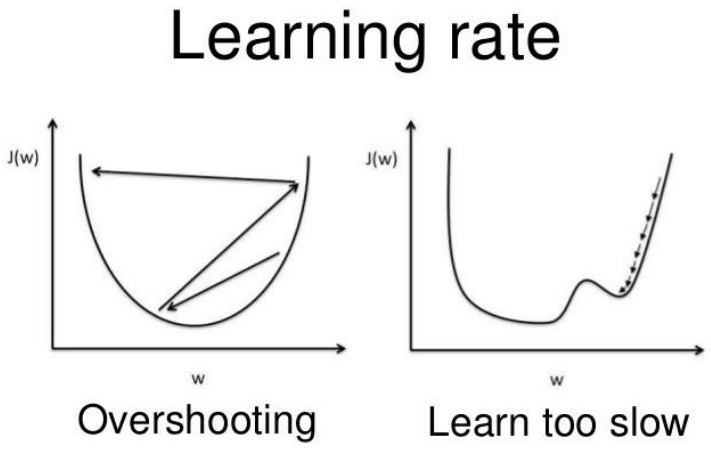


Figure 2–11: Learning rate pitfalls [23]

There are multiple optimisation techniques for Gradient descent. One is with momentum, this technique does not only take its current weights into account, but it also concerns prior values. In essence this can be seen as a ball rolling down a hill. It picks up momentum, and after reaching the bottom, this momentum will shoot the ball a bit up. When this is zero the ball will roll back into the lowest point. Another technique is Adagrad, in this technique the learning rate is changed. The learning rate will be lower for more occurring characteristics. And for less occurring characteristics the learning rate will be increased. One major problem is that if the learning rete get decreased too much there will be no learning. This problem is solved by 'forgetting' older values. The most used optimizer in current techniques is Adam, is uses both Gradient descend momentum and RMSprop. This means currently most models have an adaptive learning rate [24].

The goal of training is to increase the output performance, however too much training can have opposite effects. When a model is trained too much it will start to overfit, which means it cannot process new data. The goal of the model is to generalise and create concepts of the training data and use this knowledge to classify new data. Therefore, prevention of overfitting is key, otherwise the network will not produce a satisfactory result. There are multiple methods to achieve this, a few will be discussed below.

The first method is mostly applicable when the network has a quite high number for trainable parameters. When this is the case, the model will easily memorize the target class. Which is not ideal for generalizing new data. By reducing the network capacity, the network will be forced to learn the patterns that minimalize loss. Doing this too much and the network will start to underfit (the opposite of overfitting).

Another technique is to apply weight regularization to the model. With this method the network will not be allowed to get high weight values. This is achieved by making the loss function higher if the networks get higher weights. There are two types of weight regularization: L1 and L2. L1 the added loss value is proportional to the absolute values of the weights. With L2 regularisation the added loss proportional to the square of the weights. The two types can be used simultaneously.

The last method called dropout, is the most used method and proofed itself as quite efficient. By adding dropout, a random part of a layer's output will be set to zero. By changing this randomly the network will not learn any non-important features. The Dropout rate is the percentage of the output that will be set to zero. Only during the training phase of the network dropout will set values to zero, during the testing phase the output will be proportionally reduced with the dropout rate.

2.2.7 - Metrics for object detection

Precision and recall are not the only metrics used in object detection and semantic segmentation. Object detection models all produce bounding boxes, these are the boxes where the detected object lies within. Semantic segmentation produces masks, which is an overlay of the exact shape of the detected object. This means that semantic segmentation is harder as object detection, as the mask generally have more complex shapes.

Intersection over Union and average precision

Besides only precision and recall there are more important metrics for object detection. This metric is created with bounding boxes. The bounding boxes can create a nice metric which shows how well boxes are places on the object. This metric is called 'intersection over union' or IoU. The IoU is calculated according to the following principle:

$$Intersection\ over\ Union, IoU = \frac{Area\ of\ intersection\ of\ two\ boxes}{Area\ of\ union\ of\ two\ boxes}$$

During training the boxes that are compared are the prediction box (produces by the CNN) and the ground truth are compared on their respective IoU level. Figure 2–12 shows a visualisation of the area of intersection and the area of union, alongside with a real world example of IoU.

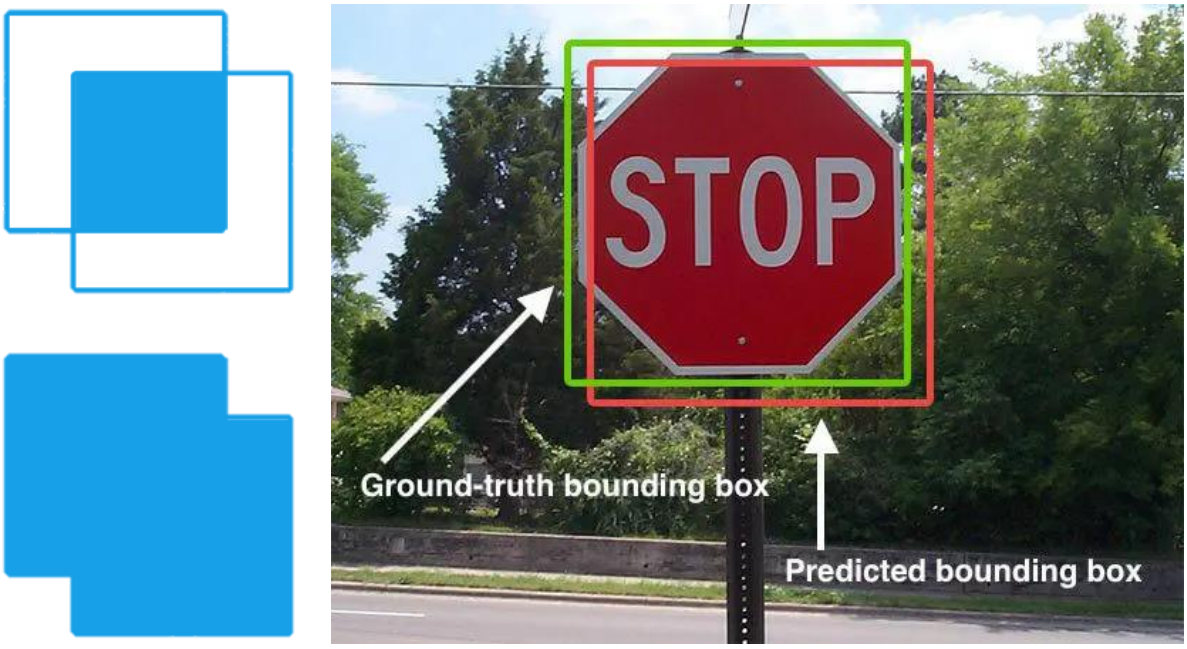


Figure 2–12: Left: bounding box intersection, union and a real world example

The metric holds such importance in the research field of object detection that this metric is used as to 'benchmark' object detection models, this is always done in combination with Average precision.

Average precision (AP) combines precision and recall into one metric. The F1 score already does this quite nice, but AP is generally seen by the object detection field as more powerful. AP is calculated with the precision recall curve. The precision recall curve combines both precision and recall into one chart. This is done by plotting precision as a function of recall. Figure 2–13 shows that a ideal pr curve drops to 0 when recall is 0. This means that only when recall is 1 precision is 0. A less perfect classifier has a more general slope down towards a recall of 1

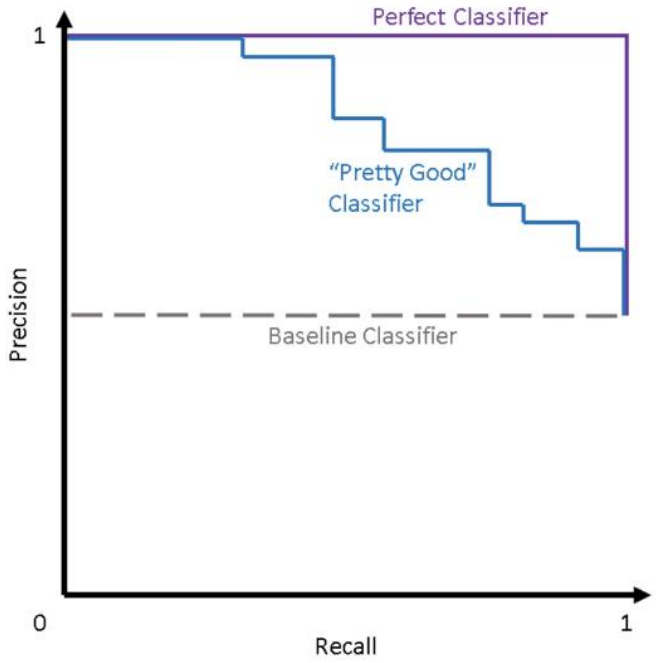


Figure 2–13: Precision and recall curve, levels of classifier [25]

With this curve the average precision can be calculated. As the average precision is the area underneath the pr curve. AP is calculated with the following formula, where AP is the integral of the pr function.

$$Avarage\ precision\ (AP) = \int_0^1 p(r)dr$$

Coco dataset

The coco dataset [26] is a dataset full of images and their annotations. The dataset has around 80 classes, ranging from people to utilities, to animals. The coco dataset is the metric used to compare every object detection model. When models are presented in their respective papers, the is always an evaluation based on the coco dataset. The models are trained and tested, which produces the metrics: Precision, Recall, F1 score, AP (average precision) at IoU of 0,5 and AP at IoU of 0,5 up until 0,95.

By combining the average precision and IoU a strong metric is formed, that evaluates the model on precision recall and how well the boxes are placed on the target object. Which in essence is all what and object detection or segmentation task needs to do.

Chapter 3 - State of the art

This project is not the first attempt to create a stomata detection algorithm, there have been multiple serious attempts before. Each of these attempts use different models, methodologies and datasets. To get a better understanding on how to tackle the problem a state-of-the-art analysis is conducted. Where multiple things like methods, models, architectures, and performance are summarized.

3.1 – Conducted research

M.W da Silva Oliveira et al [27] created an autonomous method to count the number of stomata on a microscopic picture. They achieved this by applying more traditional image processing methods. Firstly, there was a gaussian filter applied, to reduce the noise. In image processing a gaussian filter basically blurs the image. After the filter stomata are detected by finding regional minima, this is achieved by grouping pixels with the same intensity where surrounding pixels have a high intensity. With 24 images from 5 different species an average precision of 94% was achieved.

Laga et al. [28] developed a fully autonomous technique to detect stomata. Before the detection takes place the images are passed trough a gaussian filter, and transformed into a binary map. This is done by either assigning a grayscale value to 0 or 1, values higher than 0.5 were 1 and lower are 0. Then the stomata are detected with template matching. By rotating and changing size of the templates most stomata could be detected, although the researchers stated that for future work more templates are needed.

Kaue et al. [29]made a stomata detection algorithm with high resolution microscope images. Their approach was to first convert the image to RGB to CEILab colour space. After the image is multiple CEILab channels, stomata are detected using Wavelet spot detection, with the focus on spotting black spots. After the spot detection a watershed transform is used, this finds a gradient around the black spot. With this result a segmentation mask is made. This method achieved precision of 98,34% and a recall of 98,24%.

Casado-García et al [30] noticed that most prior techniques for automatic stomata detecting lack generalisation. They score quite well on recall and precision, however according to the authors are not ideal for applying these ‘hand tuned’ methods on different species. Therefore, they used the YOLOV3 algorithm to train a model that can detect stomata from multiple species **(yolo is explained later in chapter INSERT).** The algorithm achieves a f1score of 0.91 with the dataset from trained species.

In a later work by Meeus et al. [31]models based on convolutional neural networks were used as well. Here multiple models were employed, two basic models with three convolutional layers and two dense layers, and finally a output layer. The last model was based on the VGG19 architecture where some weights were already trained. All parameters were optimised with the Adam learning rule. From all trained algorithms the VGG19 scored the best, with an F1 of 0.87,0.89,0.67 on training, validation, unseen test sets respectively, with and average accuracy of 94%.

Song et al [32] used mask-RCNN to detect and obtain coordinates of stomata. The images used for training are mainly from black poplar leaves. These images are also obtained from living plants. The program obtained an segmented mask, then extracted the boundary box, fitted an ellipse over the box and used this ellipse to gather the parameters. The model manged a precision of 84,7% and a recall of 69%.

From the work of Liang et al [33] a better insight for living samples is given. The authors conducted their experiments with microscope images gathered from living plants. The algorithm is divided into two parts, one part is for detection and the other part is segmentation. In their research they used Faster R-CNN which is a convolutional neural network for detection of closing and opening stomata. The segmentation is done with cv techniques which are based on work from Li et al. [34] . Although no quantitative results are given, the qualitative results are promising, especially knowing that this setup is comparable to this project.

Dai et al [35] also developed a detection algorithm on living leaves. In this work the researchers used an adapted version of YOLO-X and implemented transfer learning. Al this is done with microscope images form living plants. With their implementation of YOLO-X the researchers achieved an accuracy of 98.7% and a precision of 95,9%. The only drawback is the fact that the model was trained with only one specie.

3.2 Conclusion

There has been a lot of earlier research done in the field of stomata detection. Some methods rely on traditional image processing, more newer methods use deep learning as their backbone. Deep learning is a still continuously developing technique, and this is reflected in the research done as well. A lot of different papers all use existing models with custom variations to improve their predecessors.

From the literature review and state of the art analysis a lot of insights and information is given. Stomata detection is something that is feasible with convolutional neural networks, since stomata clearly show some features which are trainable and extractable. There is evidence of many successful attempts, however many methods do not use the new state-of-the art models. Therefore, in this project an attempt is made to use a state-of-the-art model namely yoloV7, and an older already used model on the microscopic images that are collected from the setup from the plantenna team. With this a comparison can, be made and checked whether newer models indeed improve recognition.
Additionally, a basic ‘guide’ will be built to improve the training process for the yolo model.
An after-detection addition to the model will be made with the pre-knowledge of image context. As many images are captured throughout a cycle of one specific parts of the leaf’s surface, one could state that the number of stomata does not change throughout these images. Therefore, checks can be made on stomata existence throughout images of the same surface area.

If a stoma is found in most images, this is probably a true positive. If stomata are found in only a few images, this ‘hit’ can be an indication of a false positive. It is possible to code a filter that will ignore these false positives and therefore increase the precision of the overall algorithm. During the project the impact of this will be tested alongside overall model performance.

Chapter 4 - Mask-Rcnn and YoloV7

The two models were not selected by coincidence. Mask-Rcnn has been proven to be an effective model for stomata detection [36]. On the other hand, YoloV7 [37] was released last summer, and research on this model is simply scarce especially for stomata detection. Therefore, understanding how this model differentiates itself from other models, and how it generally works is critical.

4.1 Mask R-CNN

Mask R-CNN is a product of many iterations of the R-CNN model. R-CNN is a model for object detection that was introduced by Girshick et al. [38]. It is the predecessor of Faster R-CNN and was one of the first successful approaches to object detection using deep learning.

R-CNN works by first generating a set of candidate object bounding boxes, or region proposals, in the input image. These region proposals are then passed through a convolutional neural network (CNN) to extract features from the image. The extracted features are then inserted into a support vector machine (SVM) classifier, which predicts the class label and bounding box location for each region proposal. Figure 4–1 below shows this process

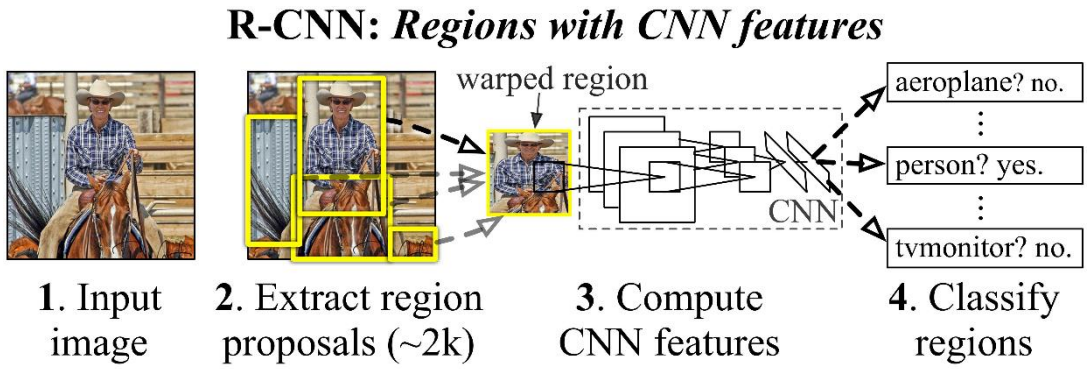


Figure 4–1: Visualisation of the RCNN pipeline [29]

One of the main advantages of R-CNN is its ability to perform object detection and semantic segmentation in a single model. Semantic segmentation is the task of labeling each pixel in the image with a class label, such as "sky," "tree," or "building." R-CNN is able to perform semantic segmentation by predicting a class label for each pixel within the region proposal, rather than just for the bounding box as a whole.
On the other hand, R-CNN has several limitations. It is relatively slow, since it requires processing each region proposal separately through the CNN, which is computationally expensive. R-CNN also requires a large amount of annotated training data.

A newer version of R-CNN was introduced to resolve its shortcomings, which was Faster R-CNN [39]. Faster R-CNN uses a region proposal network (RPN) to generate the region proposals, rather than using a separate model such as selective search. This makes the model faster than its predecessor, because the region proposals are generated on the fly, rather than being pre-computed. Faster R-CNN also uses a shared set of CNN features for all the region proposals, rather than processing each proposal separately, which further increases efficiency.

Faster R-CNN consists of two main components: a region proposal network (RPN) and a Faster R-CNN detector. The region proposal network is responsible for generating a set of candidate object bounding boxes, or region proposals, in the image. These region proposals are then passed to the Faster R-CNN detector, which classifies each proposal as an object or background and refines the bounding box location.

To generate the region proposals, the region proposal network first applies a sliding window over the input image, at multiple scales and aspect ratios. At each location, the network applies a set of convolutional and max pooling layers to extract features from the image. These features are then fed into two fully connected (fc) layers: one that predicts the objectness score for each bounding box, and another that predicts the bounding box's location and size. The objectness score is a measure of how likely it is that the bounding box contains an object. The location and size predictions are used to refine the bounding box to better enclose the object. The region proposal network is trained to optimize both the objectless score and the bounding box location and size predictions.

The Faster R-CNN detector is a multi-task network that simultaneously predicts the class labels and bounding box locations of the objects in the image. It takes as input the region proposals generated by the region proposal network, along with the feature maps extracted from the input image by a convolutional neural network (CNN). The Faster R-CNN detector processes the region proposals in parallel, using a shared set of CNN features, to predict the class labels and bounding box locations of the objects.

Mask R-CNN is an extension of Faster R-CNN which was introduced in 2017 [36]. It adds an additional branch to the Faster R-CNN network that predicts a binary mask for each object. This allows Mask R-CNN to perform instance segmentation, which is the task of segmenting individual objects within an image and overlay a mask. Instance segmentation is a more challenging task than object detection because it requires not only identifying the objects in the image, but also specifying its shape.

The Mask R-CNN network consists of a feature extraction backbone (such as a ResNet), a region proposal network (RPN), and a detection head. The feature extraction backbone is responsible for extracting features from the input image, which are then passed to the RPN and detection head. The RPN generates region proposals similarly to Faster R-CNN. The detection head processes the region proposals to predict the class labels, bounding box locations, and masks for the objects in the image.

The detection head of the Mask R-CNN network consists of several convolutional and fully connected layers that are trained to classify the objects and refine the bounding box locations. It also includes an additional branch for predicting the object masks. This branch consists of a series of convolutional layers that process the feature maps from the feature extraction backbone and produce a mask for each object. The mask is a binary image that indicates the pixels that belong to the object.

4.2 - Yolo, ‘You only look once’

Yolo is a model for object detection that was introduced by Joseph Redmon and Ali Farhadi in their 2016 paper [40]It is a popular choice for object detection tasks as it is fast and accurate. The main idea behind yolo is to use a single CNN to directly predict the class labels and bounding box locations of the objects in an image. This is done without the need for a region proposal network or separate

classification step. This allows yolo to perform object detection in a single pass, rather than in multiple stages (Mask-RCNN), which makes the model faster than other models. Hence the abbreviation ‘you only look once’

Yolo achieves this divides the input image into a grid of cells and predicts the class labels and bounding box locations for the objects within each cell. It uses a CNN to extract features from the image and predict the class labels and bounding boxes. The CNN is trained to optimize the objectiveness scores, which are a measure of how likely it is that a bounding box contains an object, and the bounding box locations.

One of the key features of YOLO is its ability to handle multiple sized objects within the same image. This is done by using anchor boxes, which are predefined bounding box shapes that are used to represent the different scales of objects in the image. The CNN is trained to predict the class labels and bounding box locations for each anchor box, and the final bounding boxes for the objects are obtained by adjusting the anchor boxes to fit the objects. This is done by non max suppression, which reduces all the prediction bounding boxes into one. This is done based on the IoU with the anchor box, and the confidence of a bounding box. Figure 4–2 shows the pipeline of yolo for a detection.

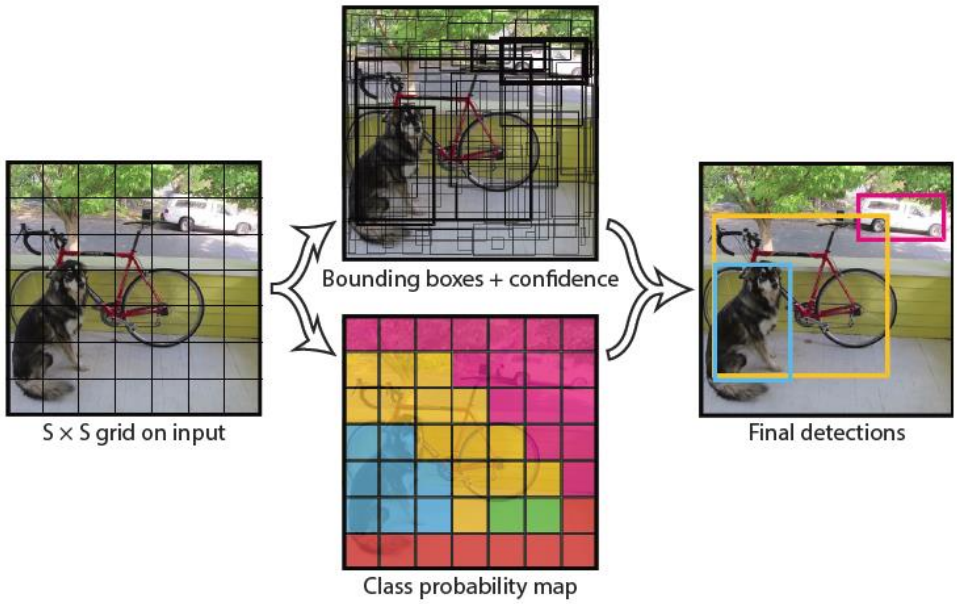


Figure 4–2: A visualisation of the yolo pipeline [31]

After the first introduction of the yolo model back in 2016, several other iterations have been developed. Several iterations are written and developed by different authors, especially not by the original author who quit the object detection scene due to ethical concerns. Only a model can call itself yolo with the permission of the original creator of the first model.

- Yolov2: introduced back in 2016 [41], Yolov2 improved upon the original model by using a different network architecture and training procedure, resulting in higher accuracy. It also introduced the use of anchor boxes for handling multiple scales of objects, as mentioned above.
- Yolov3: introduced in 2018 [42] yolov3 further improved upon yolov2 by using a more efficient network architecture and adding features such as multi-scale training, which allows

the model to learn to detect objects at different scales. It also introduced the use of residual connections, which allow the network to learn more complex features.

- Yolov4: introduced in 2020 [43] the model made significant improvements to the network architecture and training procedure. The model backbone was changed to one requiring less connections, resulting in higher accuracy and faster performance. It also introduced mosaic data augmentation which combines images. This model was the first model to introduce the bag of freebies, which are augmentations that make the model more accurate without sacrificing inference speed.
- Yolo5 [44] introduces in June 2020 improved by using a more efficient network architecture and training procedure, resulting in higher accuracy and faster performance. Prior to yolov5 all yolo models were built in TensorFlow, however yolov5 was the first model developed in Pytorch.
- YoloR [45] released in 2021. The goal of this model was to create a model that can combine implicit and explicit knowledge into general representations. This model tries to mimic the mind, as we combine pas experiences with current senses to process never seen data.
- Yolov6 [46] released in 2022 focused on increasing the efficiency and compactness of the model, while maintaining high accuracy. The head of the model is decoupled, which means there are additional layers separating the boxy form the head.
- YOLOv7 [37]: introduced in 2022 made further improvements to the network architecture and training procedure, resulting in higher accuracy and faster performance. It has an optimized architecture and optimised loss function. By focussing on all ‘improvements’ other models made the authors managed to create multiple improvements in both the backbone and head of the model. The repository also supports instance segmentation and pose estimation. YoloV7 was at the time of publishing the fastest and most accurate model.

Overall, Yolo has undergone significant evolution since its introduction, with each version representing an improvement over the previous version in terms of accuracy, speed, and efficiency. It remains a popular choice for object detection tasks due to its continuous new load of iterations. The Yolo family is currently one of the most popular object detection models.

Chapter 5 - Execution - The training process

This chapter will focus mainly on the steps taken in the training process. The training process of an CNN is not linear, it really is about repeating and testing this process. There is a certain rule in AI, and machine learning in general: 'garbage in – garbage out' [47]. This means if there are already errors within the training data, the resulting CNN will also produce the same nonsense as the input data. Therefore, it is critical to find good performance, and check what can be changed in the training dataset and model choice to get the best performance from the CNN.

5.1 - setting up environments

First start with how the project is made, for both project Anaconda is used. Anaconda is [48] a ready to use python environment, which is specifically developed for scientific purposes. This makes it possible to create certain python environments. These environments enable us to choose specific python versions and install certain libraries. These libraries are only installed within the created environment, which means each environment can have its own setup. Additionally, anaconda has a some widely used pre-installed IDE's such as jupyter and Spyder.

For this project two environments are created, (yolov7) and (maskRCNN). These environments have different dependencies installed. Of course, the biggest requirements are the ones that come with the CNN that is used. For my laptop to be able to train on the graphics card, which is proven to be the faster method of training, CUDA is used [49]. This is a package of drivers and loaders that are developed by Nvidia. In essence CUDA used the powers of an GPU (Graphics processing unit), these devices generally have high processing power and multiple computing cored. This makes it the ideal for deep learning, and especially training a CNN.

For the implementation of yoloV7 Pytorch is installed [50]. PyTorch is a versatile and flexible machine learning library that provides a wide range of functionalities for both research and production. It is widely used for tasks such as deep learning, computer vision, reinforcement learning, and generative models. Furthermore, pytorch uses the latest version of cuda, which is supported by the graphics card in the laptop that is used for training. For an overview of each library installed in the environment see appendix A.

The second environment has TensorFlow installed. [51] Originally TensorFlow was intended for Google's internal use. TensorFlow uses earlier described tensors, the library enables the coder to make computations with these tensors.

The implementation of maskRCNN was not as easy as expected. The model 'relatively old' as the computer vision landscape is continuously changing newer and newer models are created every year. And every new iteration is an improvement on the previous one. Different libraries, and different library versions are used.
maskRCNN was state-of-the art back in 2017, back when tensorflow was in their 1.- era. Currently there is a new and improved version. Tensorflow 2.0 just works slightly different, certain methods are changed, they require different inputs ore are overall not used anymore.

'just install and use the older version' is the obvious answer to this problem. However a simple solution was not applicable. For the original maskRCNN repo to function, tensoflow 1.0 is a requirement, however the graphics card in the laptop used for this project laptop is not supported by this version. Therefore, some workaround must be made. Luckily maskRCNN is an open-source

project, which enables other developers to look trough and change the code. There is a repo on github from the maskRCNN model that luckily supports cuda 2.* this meant that the model could still be used.

One problem again, this repo is also quite dated (2 years old) therefore again some specifics could have been changed. Via trail and error changing library versions the environment was eventually running the maskRCNN model. This meant that for this model the training could begin. For the specific environment file on the requirements overview see appendix**INSERT PLEASE.** This file shows which version of which library is used.

5.2 – Training dataset

Stomata detection has been a research topic throughout the years. In prior chapters the conclusion can be made that stomata detection using CNN is not something new. Therefore, there should exist datasets from stomata. Although there is plenty of research, and thus data, this data is mainly from leaf imprints. These imprints are gathered via the nail polish method, shown in the Figure 5–1

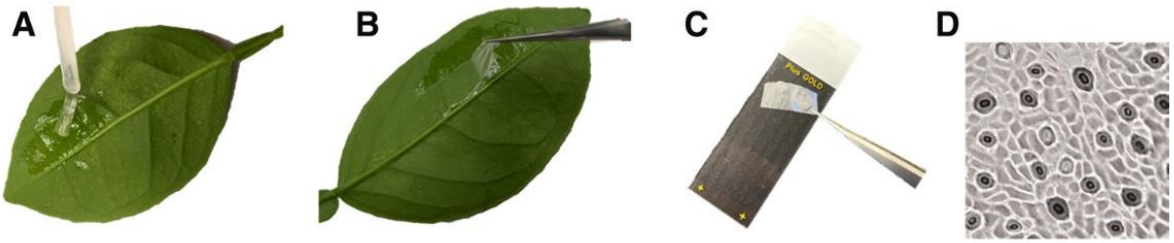


Figure 5–1: The nail polish method to gather leaf epidermal prints. A Nail varnish is applied on the abaxial side of the leaf. B The dried varnish is peeled of the leaf. C The peel is mounted on glass for microscopic analysis. D The image captured at 20X magnific [5]

This does not mean that these images are useless, as the clear features of stomata are still visible on the imprint. Therefore, the dataset will also include the imprints to 'boost' the number of images and therefore stomata in the overall dataset. The following dataset is used as an initial starting dataset. The dataset has multiple classed namely open and close. However, these classes are all remapped into 'stoma'.

Besides the already available dataset another data is created. This dataset only consists of images captured by the target hardware, the microscope camera from the platanna team. These images are delivered as a stack of images, with a total of 119 images per stack. Although the stack does not change that much, the stomata in the images do. Therefore around 22 images of each stack are extracted and annotated.

The stack can be seen as a full day/night cycle, where the stomata open and close. With these images the model can be trained on both closed and opened stoma, as these tend to differ. The online found dataset mainly shows open stomata, however, also contains closed stomata. This found database namely had both open and closed annotation. For the training of the models for this project, these open and close classes are overwritten with the class stomata.

For annotation the online program Roboflow [52][7] is used. Roboflow is an easy-to-use online tool for annotation. It has certain key features that make it interesting to use. Such as annotations assist, which can load up a model to pre-detect and annotate your data. Furthermore, it can export your dataset in different annotation styles. For YOLO the annotation style consists of text accompanied with the images. In the text are the locations, width and height of the bounding boxes saved. For mask-RCNN the export style is Pascal VOC. Figure 5–2 shows a screenshot of the roboflow interface, it the boxes are the currently drawn bounding boxes on the image.

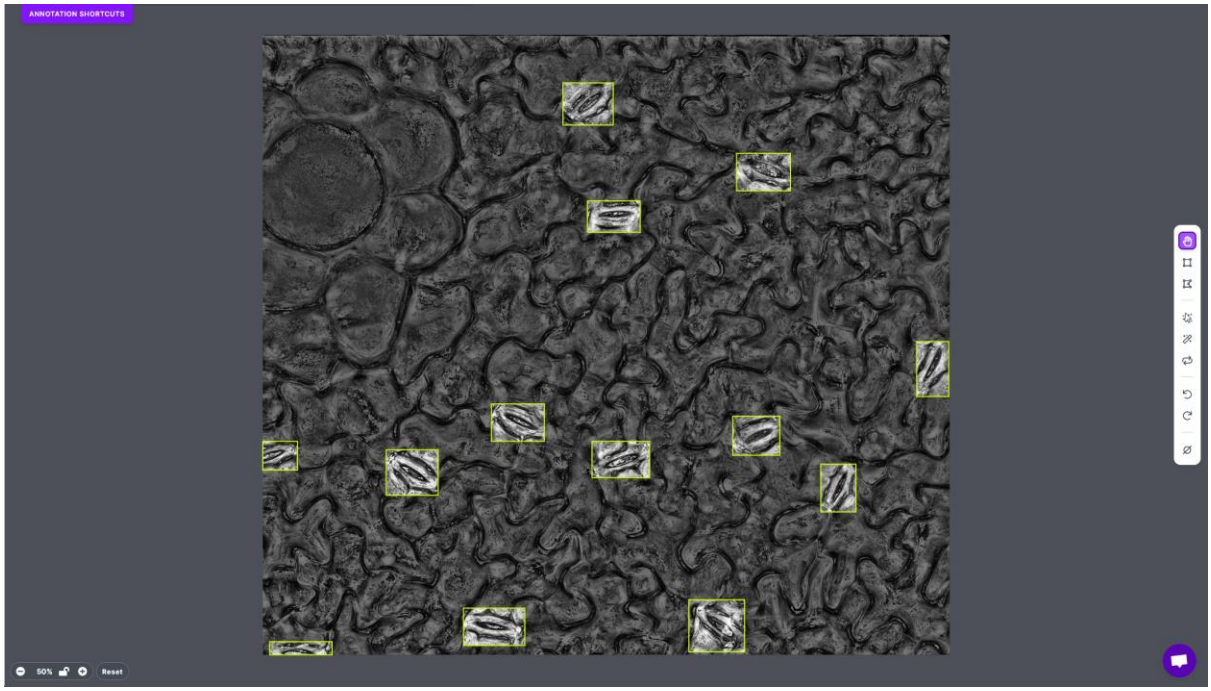


Figure 5–2: A example of the roboflow user interface

Annotation of the stomate was a long and sometimes tedious task. This task is especially hard for stomata from the tomato images. These stomata are small, and sometimes look like neighbouring cells. In other cases, the other way around was true, where some regions of cells really looked like stomata. The chrysant images are a bit easier to detect as these images have clearer stomata within them.

5.3 - Training methodology for yolov7

YoloV7 is currently the state-of-the art object detection model, by the original authors of yolo. This means that there are not many statistics on which model version works the best. Therefore, training will be done on different 'sub' models of yoloV7. These are the following yoloV7, yoloV7-tiny, yoloV7x. yolov7 tiny is the smallest, only being around 12.5mb large. Yolov7x is the largest model that is tested, which is around 140mb. Yolov7 (default) is around 75mb large.

The other variable in this research is the train, validation, and test dataset. As training is a process that need large amounts of data. Finding a data setting for optimal training might be a useful asset, as training is an energy and time-consuming task, albeit for the training hardware.

A set of different training test and validate datasets is created. The first training set is the 'default' set. Table 5-I: Table 5 I: images in defrault trainingTable 5-Ishows the train, validate and test distribution of the dataset used for the first training. This training will be called the 'default'.

Class	Total	Training	Validation	Test
Stoma	1693	1200	394	99
	100%	71%	23%	6%

Table 5-I: Table 5 I: images in defrault training

The second dataset is in the training part a hard copy of the first training dataset. The only difference is within the validation part. Here all non-target images are stripped.

Class	Total	Training	Validation	Test
Stoma	1387	1200	22	99
	100%	86%	2%	12%

Table 5-II: Target training dataset

The final dataset only contains target images in both the training and validations part of the dataset. The resulting distributions is shown in the table below. Trained models with this dataset all have transfer at the end of the model name

Class	Total	Training	Validation	Test
Stoma	234	204	30	99
	100%	82%	12%	6%

Table 5-III: Transfer training dataset

The biggest difference between the datasets is the size and number of images in the dataset. The target dataset has a quite low amount of validation images compared to the training images. This might have an impact on the overall model performance.

5.4 - Training the default dataset

The training is carried out on a windows system, using CUDA 11.8 drivers for Windows. The system has the following specifications: A Nvidia RTX3070 Laptop GPU (8GB Vram), AMD Ryzen 7 5800H with Radeon Graphics processor. 16GB of ram
The models are trained with Python version 3.8 in Anaconda3 for Windows. The environment is the earlier described yolov7.

For the first training sessions the three Yolo models are trained with the following command. Note that this command is an example. See appendix **INSERT** for the full list of commands. Table 5-IV shows an overview of training input. The batch size is picked to maximize the GPU memory usage, a bigger batch size means that more images can be fed to the GPU in a shorter amount of time. In the end the overall training time will therefore be optimal.

python train.py --workers 2 --device 0 --batch-size 8 --data ./data/data.yaml --img 960 960 --cfg ./cfg/training/yolov7.yaml --weights yolov7_training.pt --name yolov7-960-default --hyp data/hyp.scratch.p5.yaml --epoch 300

Name	weights	batch size	workers	Epoch
yoloV7-default	Yolov7_training.pt	5	2	300
yoloV7-tiny-default	Yolov7-tiny.pt	26	2	300
yoloV7X-default	Yolov7x_training.pt	3	2	150

Table 5-IV: an overview of input parameters for the train.py file

During the training of the Yolo CNN weights are changed every epoch, just the same as for every other CNN. After each epoch a small test is conducted, this test is done with the validation set, to get a fitness value. According to the test the weights are changed again and again, up until the 300th epoch is reached. As an input value, batch size has the highest impact on the training time and memory needed for training. Figure 5–3 shows two different batch files, with two different batch sizes.

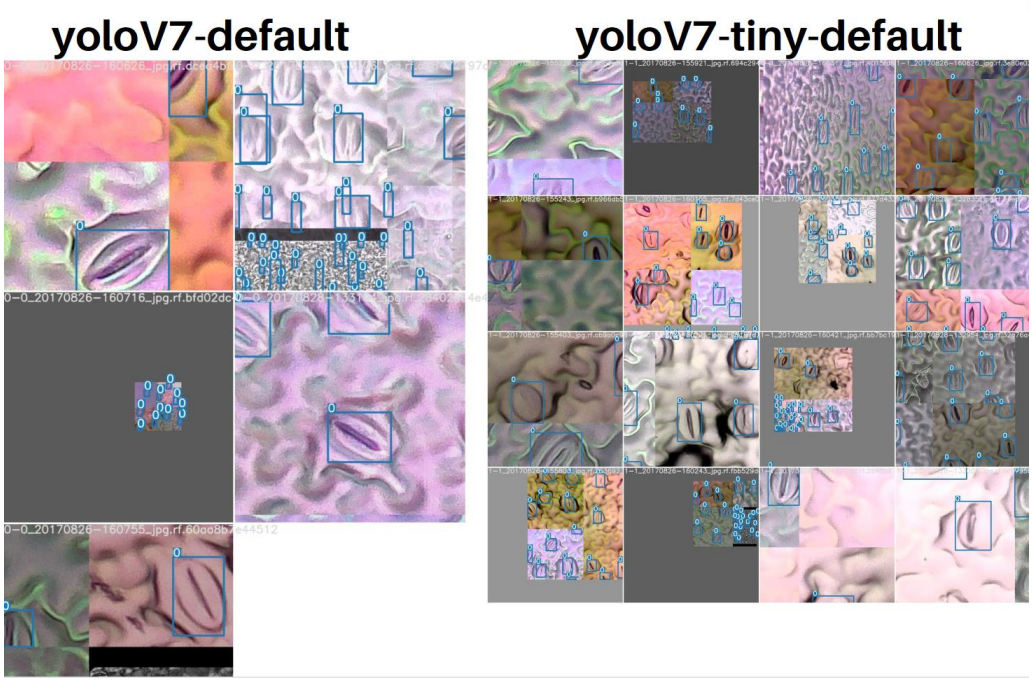


Figure 5–3: the left part shows one of the many training batches for yolov7-default. The right side shows the images per training batch for yolov7-tiny. The blue boxes with label 0 are the bounding boxes.

With a batch size of 8 it took the model 11,54 hours to reach the finish the 300th epoch. Note that the amount of training images is quite high. The time per epoch is around 2 minutes and 38 seconds. The table below shows overall training data and the results of the best epoch. This result is based on the validation part of the dataset. This is true for each table that will be shown below.

The difference between the validation and train dataset is mainly how Yolo this handles. As seen in Figure 5–3 the images in the training batches are overlapping, cropped etc. These all change the nature of the image. The validation dataset does not change. Figure 5–4 shows on the left the input image with bounding boxes (ground truth). And the right part of the image shows what the prediction of the model is. The results are compared to the ground truth, and the following metrics are produced. Precision, recall, mAP.05 (ioU of 50%) and mAP 0.95 (ioU of 95%).

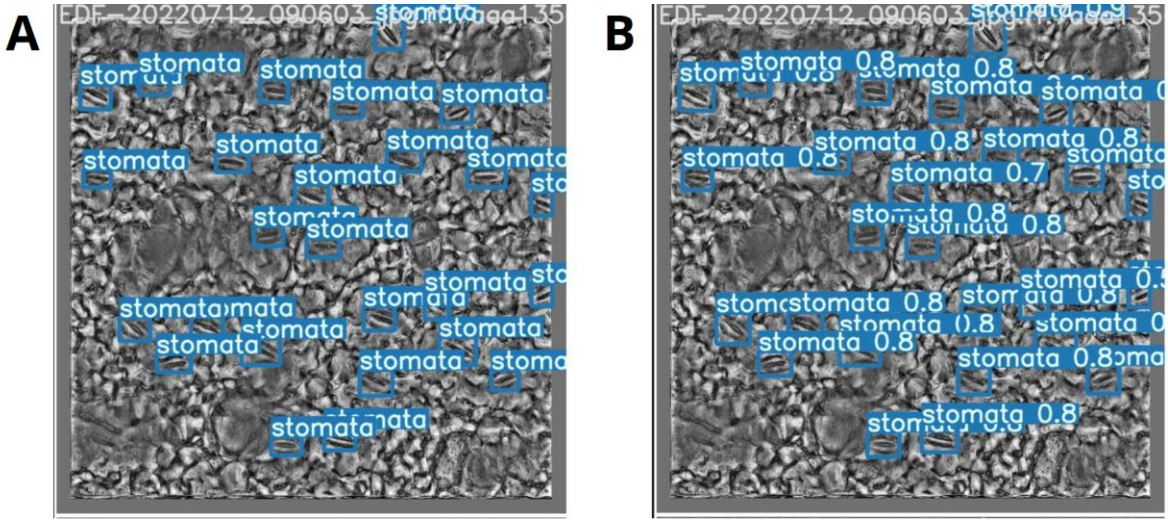


Figure 5–4: A, ground trugh B, predictions

Table 5-V shows these metrics, for the most optimal epoch from the training session for the default dataset.

Default							
Name	**Precision**	**Recall**	**mAP@.5**	**mAP@0.95**	**Total epoch**	**Batch-size**	**Train- time**
yoloV7-default	0.94	0.969	0.979	0.7	300	8	11h 54m
yoloV7-tiny-default	0.94	0.961	0.975	0.684	300	26	4h 58m
yoloV7X-default	0.942	0.964	0.979	0.695	150	3	7h 31m

Table 5-V: A table of the metrics on which yolov7-default is trained and validated(left side). The right part shows the total epochs in the training session the batch size and the total training session time.

The first training session directly shows some promising results. Although the mAP 0.95 is not extremely high. The mAP 0.5 reaches around 0.98 which shows that the bounding boxes are placed quite nicely around the stomata. The precision and recall both reach an significantly high value. Meaning that only stomata are detected, and that there are barely any misses. From the training metrics Figure 5–5 clearly shows that this model is almost starting to over fit. This means that the model performance is starting to fall, in most cases the recall or precision will start to reduce as the model tries to improve either one of the metrics. The graphs of each epoch results can give an meaningful insights, if the model is overfitting. When a model over fits in during the training of yoloV7 is does not mean that the best epoch is bad. YoloV7 picks the best epoch based on set metrics, which for this project is equally distributed amongst the metrics.

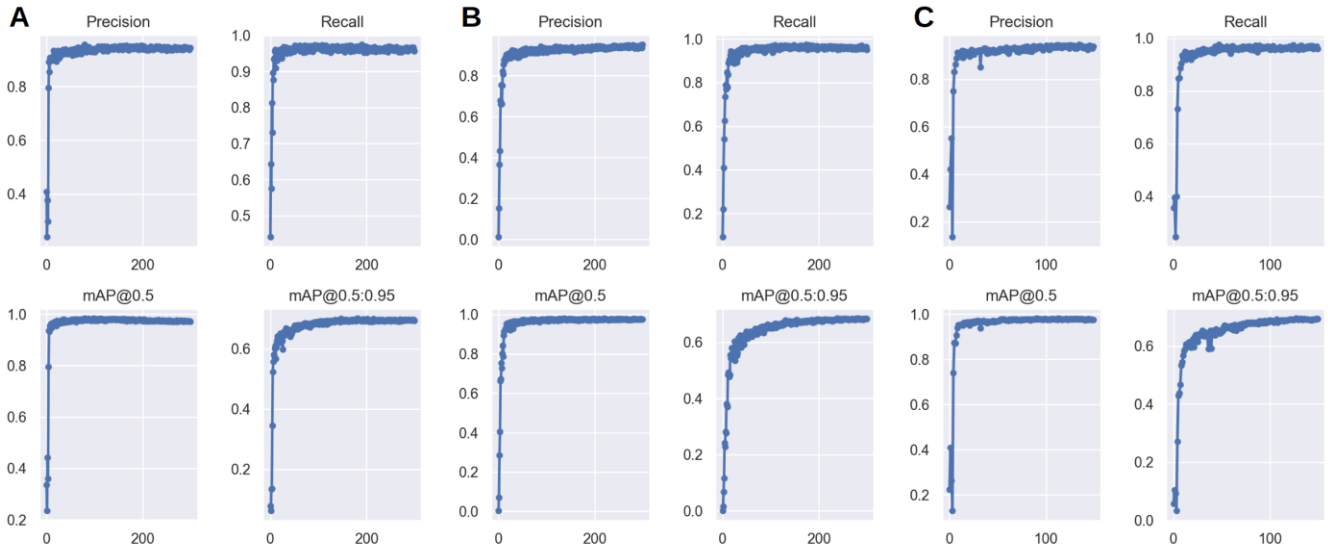


Figure 5–5: a. yoloV7 training results b. yoloV7-tiny training results C. yoloV7x training results

For the second training session only target images made up the validation part of the dataset. Although the amount of validation images were small, the training still took long since the test set was not changed. Table 5-VI shows the results op the best epoch, together with the total amounts of epochs.

Target							
Name	**Precision**	**Recall**	**mAP@.5**	**mAP@0.95**	**Total epoch**	**Batch-size**	**Train- time**
yoloV7-target	0.961	0.962	0.968	0.573	450	5	16h 13m
yoloV7-tiny-target	0.947	0.946	0.959	0.519	450	26	6h 45m
yoloV7X-target	0.954	0.957	0.966	0.548	300	3	13h 43m

Table 5-VI: A table of the metrics on which yolov7-target is trained and validated (left side). The right part shows the total epochs in the training session the batch size and the total training session time.

Even though the total amount of epochs is larger then the yoloV7 model the training results do not differ that much. In this case all metrics are slightly lower, this can be a result of the difference in validation. As seen Table 5-II the validation set has a small fraction of the total image pool, this might mean model simply cannot get a fair evaluation of its own performance. And therefore, cannot train as targeted as the 'default' training.

To fully understand the metrics of each of the best epochs, the full training sessions for each model is analysed again. Figure 5–6: A The results for yoloV7. B The results for yoloV7-tiny. C The results for yoloV7x shows the all the graphs that result from the training sessions. In comparison to the 'default' training the target training shows a slower increase for all metrics. The 'default' models the metrics quickly rise to values of 0,9 (except mAP0.95), the metrics from the target training reach these values simply later. In addition, the slope of the curve is less steep, which indicated it took a longer time to learn the features of stomata.

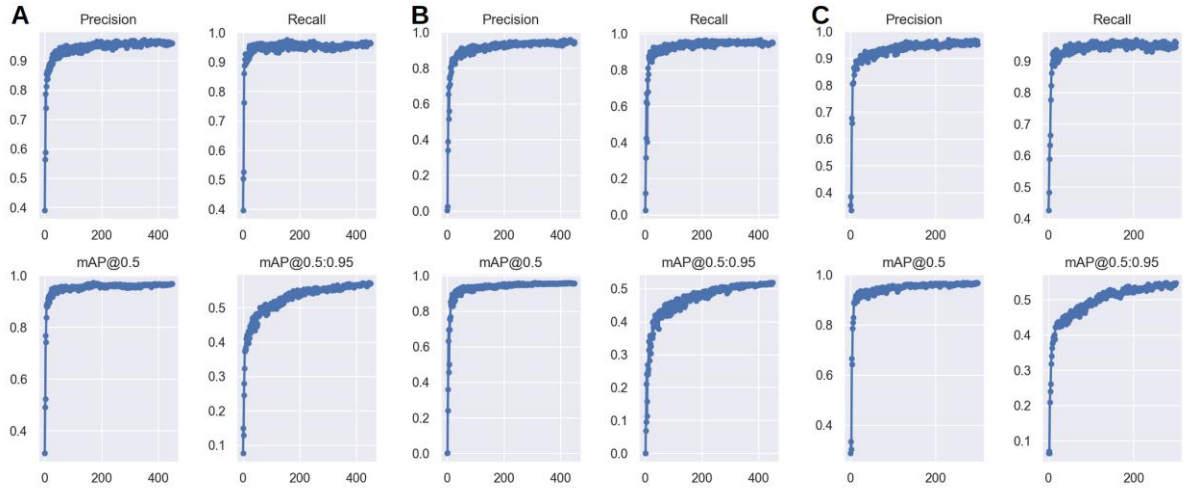


Figure 5–6: A The results for yoloV7. B The results for yoloV7-tiny. C The results for yoloV7x

The next training session is named **'default-transfer'**. The best model that resulted from the default training is trained again. Now with transfer learning. Transfer learning is proven to benefit model performance according to this paper [42]. Although this paper is focussing on yoloV3, an improvement can occur for yolov7 as well

Default-Transfer							
Name	**Precision**	**Recall**	**mAP@.5**	**mAP@0.95**	**Total epoch**	**Batch-size**	**Train- time**
yoloV7-default-transfer	0.977	0.973	0.987	0.631	450	5	3h 10m
yoloV7-tiny-default-transfer	0.948	0.914	0.959	0.538	450	26	1h 38m
yoloV7X-default-transfer	0.954	0.968	0.965	0.625	450	3	4h 22m

Table 5-VII: : A table of the metrics on which yolov7-default-transfer is trained and validated (left side). The right part shows the total epochs in the training session the batch size and the total training session time.

If the results in Table 5-VII and the original Table 5-V are compared, a clear increase is evident. Although mAP0,95 seems to have reduced drastically. This phenomenon can be a result of the change in validation images. The validation images for the 'default' dataset have way bigger stomata Figure 5–7 shows the difference, on the left side there are a few clear stomata in the image, which means it is way easier for yolo to have a high ioU. The right side of the image clearly shows multiple smaller stomata. This makes it for yolo harder to refine good bounding boxes.

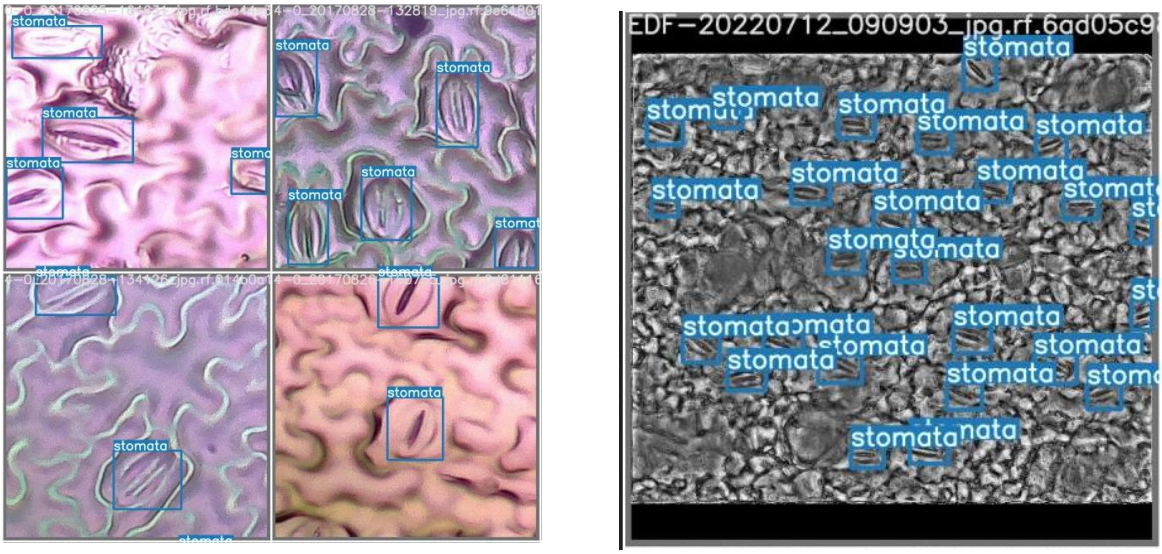


Figure 5–7: default and target ground truth bounding boxes.

This does not mean that the mAP0,95 metric has only decrease during the training, from the start this metric was lower in comparison to the best epoch of the default dataset. Figure 5–8 shows the graphs f each metric for each epoch during training.

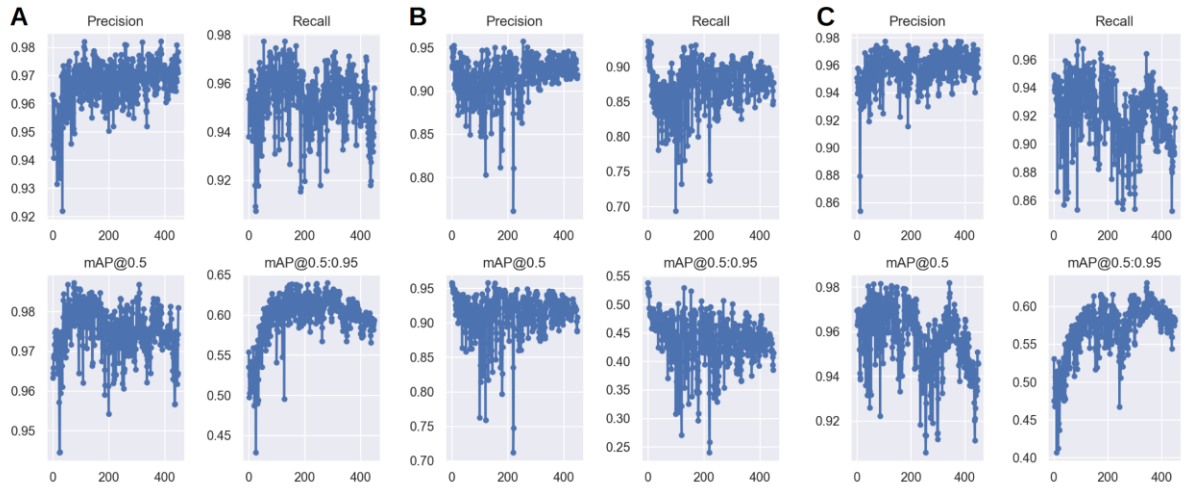


Figure 5–8: Default transfer training results A The results for yoloV7. B The results for yoloV7-tiny. C The results for yoloV7x

It is obvious that during the 'default' some metrics already reached a near peak value, this is especially true for the yoloV7-tiny model. This model does not benefit the transfer learning for any metric, all it does is overfitting. This is obviously shown by the slope of graph B going down. For the other two models the results are more interesting. Both models have an increase in precision, as both graphs A and C show. The recall on the other hand shows a slight increase for the first few 100 epochs and starts to decrease afterwards. Again, this is a sign of over fitting. Besides the recall, both mAP0,5 and 0,95 starts to show signs of overfitting. For mAP0,5 around the 100th epoch, and mAP0,95 around the 200th epoch.
Overall, it seems that yoloV7 and yoloV7x have had some benefit with the transfer learning, although it is hard to determine since the validation sets have been different.

The fourth and final training session is the transfer learning on the best models of the already trained target dataset. As earlier described the transfer dataset only contains target images for validation. In essence the target-transfer training session will result in a model only validated on the target images. This means the modal only received images from the microscope.
Table 5-VIII shows a clear increase for every metric. In essence the validation part of the learning process is just lengthened with a training set of images that only contains the target. Again yolov7-tiny shows the smallest increase for all metrics.

Target-Transfer							
Name	**Precision**	**Recall**	**mAP@.5**	**mAP@0.95**	**Total epoch**	**Batch-size**	**Train- time**
yoloV7-target	*0.961*	*0.962*	*0.968*	*0.573*	*450*	*5*	*16h 13m*
yoloV7-target-transfer	0.972	0.977	0.987	0.645	450	5	3h 04m
yoloV7-tiny-target	*0.947*	*0.946*	*0.959*	*0.519*	*450*	*26*	*6h 45m*
yoloV7-tiny-target-transfer	0.952	0.91	0.954	0.543	450	26	1h 20m
yoloV7X-target	*0.954*	*0.957*	*0.966*	*0.548*	*300*	*3*	*13h 43m*
yoloV7X-target-transfer	0.975	0.958	0.988	0.642	450	3	4h 1m

Table 5-VIII: Table 5 VII: : A table of the metrics on which yolov7-target-transfer is trained and validated (left side). The right part shows the total epochs in the training session the batch size and the total training session time

As for other training sessions, the graph produced by the algorithm is studied further. shows the metrics for each epoch during training.

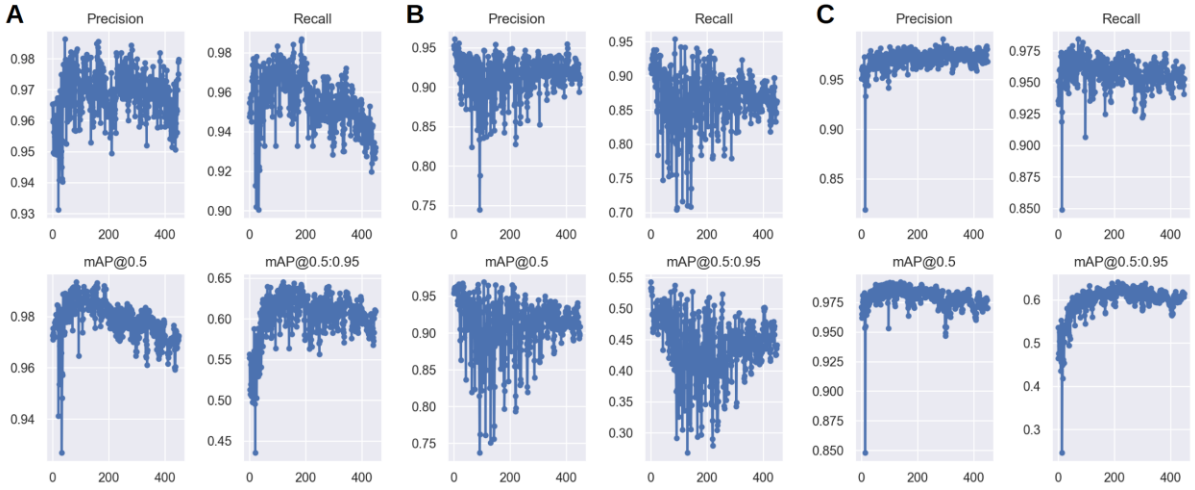


Figure 5–9: : target transfer training results A The results for yoloV7. B The results for yoloV7-tiny. C The results for yoloV7x

The graphs for each model show the same trends as with the 'default-transfer' session. Again, the yolV7-tiny model (B) does not have an increase in all metrics. To be fair, a few epochs form the Recall of B show a higher initial value, however these are the exception. This clearly indicated that the tiny model is already overfitting from the start. YoloV7x (C) shows quite an interesting curve, there is a big dip around the 10th epoch. All metrics decrease with 10%, however the next epoch the values rise slowly again. To continue with C, all metrics decrease over time, after a small rise until the 150th epoch, for most metrics.

To stay fair, these graphs all have a varying scale, this means that comparing might have been an unfair process. However, in appendix **C** is a set of excel graphs all with the same scale, and all made form the output.txt file for each training session and each model. For reference, all graphs are also scaled within the last 10% range, to see the curve more precise for the final epochs.

From all the trained models the yoloV7x-target-transfer is the most promising. With a total precision of 0,975 the models should find only stomata. Additionally, the high recall of 0,958 there are not many false negatives. However, some of the models had a different validation and training. Therefore, it is only fair to compare all models on a true never seen test dataset.

Figure 5–10 shows an output of the model note that the bounding boxes are quite small, as the model resizes the image to 960x960 before detection, and after detection back to the original size. The bounding boxes are placed quite well on the stomata. The stomata on the edges also are being detected. This is due to the moaisic cut&mix what happens when the training batches are formed.

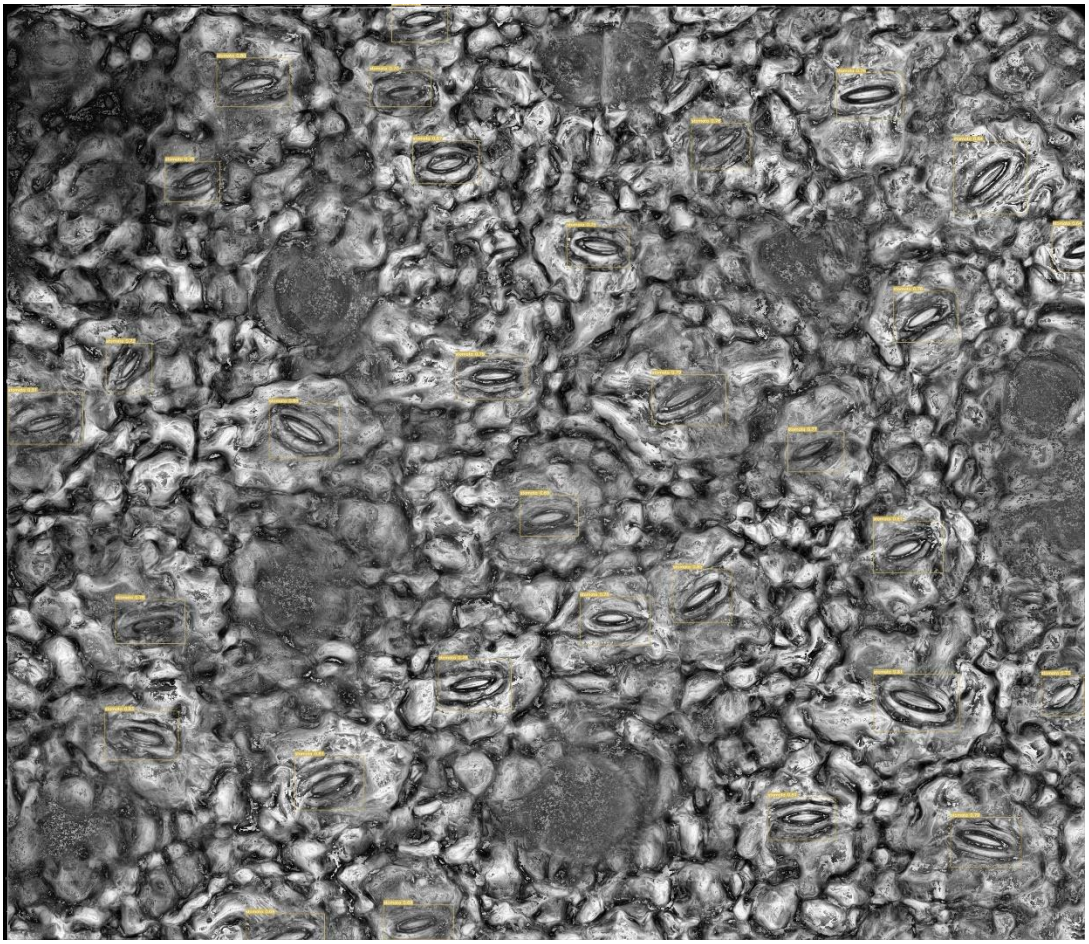


Figure 5–10: Image form a detection done by yoloV7-default

5.4 – Training of Mask-RCNN

Mask-Rcnn had a rather different approach for training. Mask-Rcnn produces other metrics during training, and the training of yoloV7 was a bit more **VOORKEUR ENGELS .** However mask-rcnn was trained with the default training methodology, and the transfer learning methodology. The transfer learning training in mask-rcnn had a different type of bounding boxes. Here the bounding boxes were changed into masks, this process took some time, and is sadly not that well researched within this paper.

The focus of the project shifted more to creating a more elaborate training and validation method for yolov7, which overshadowed mask-rcnn. However, this decision was made with well knowing that mask-rcnn is an older model with well documented research on stomata detection.

Figure 5–11 shows some nice images from the mask-rcnn model. But I highly recommend not using this model anymore, given the added complexity for newer hardware.

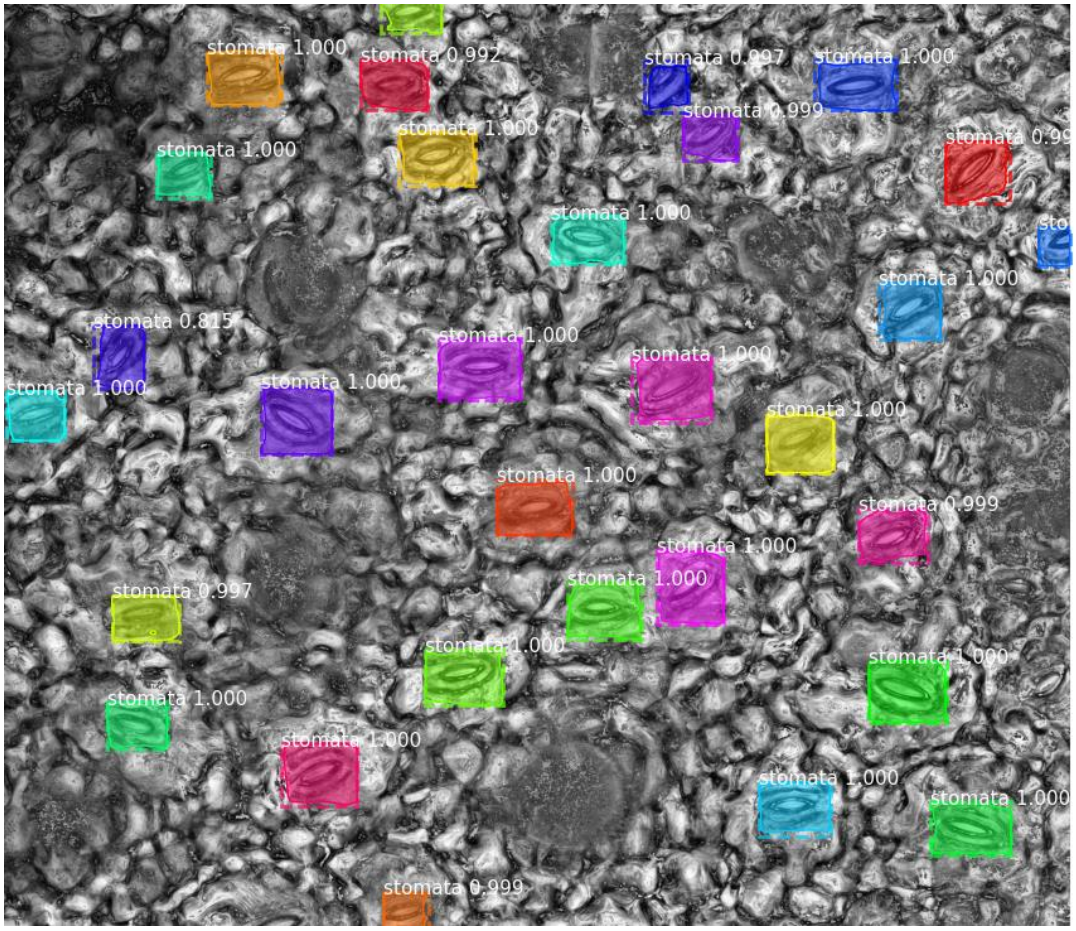


Figure 5–11: A detection output from mask-rcnn

5.4 - Using stacks to increase precision.

A lot of tests on the models are conducted during this project. False positives occurred now and then, which are preferably filtered out when this model is used for actual real-world situations. Therefore, there is a need to improve the detections without extra training.

There is an option to improve this. If the data is presented in a stack there is a possibility to use the knowledge of all images collectively to boost the overall performance. This means that for a stack or a full set of images one final image with bounding boxes can be made.

The current inference class of yoloV7 is detect.py [53] this function detects the images fed to the algorithm, together with certain pre-sets the user puts in when calling the detect method. There is an option to save each detection made on an image in a .txt file, however this is per image, and not per stack. Therefore, a list of each bounding box in each image is made. This is done through a multidimensional list : *[[image[bounding boxes]]]*. The bounding boxes are saved in an xyxy manner, which means that the xy of the upper left corner, and the xy of the lower right corner is saved. With this a rectangle can be made.

After the whole stack is processed the list is filled with all the bounding boxes for each image. Note that this list only contains coordinates so no confidence levels or ioU levels.
Now the following code is written to filter out non overlapping bounding boxes. For every image, loop trough every bounding box. Check for every bounding box if there is another bounding box in

any image that overlaps. If they overlap the count. Then only return a list for every bounding box that exceeds a minimal amount of overlap (images), and the number of overlaps. See appendix C for the created code.

Then the final list is drawn on the first input image as shown in Figure 5–12. The green squares are quite thick, as the image does show every bounding box that overlaps. This is happening since the model does not predict every bounding box at the exact same location. There are also other green boxes shown, which accidently happen to overlap. Therefore, the current code is not robust enough to be tested on an increase in precision.

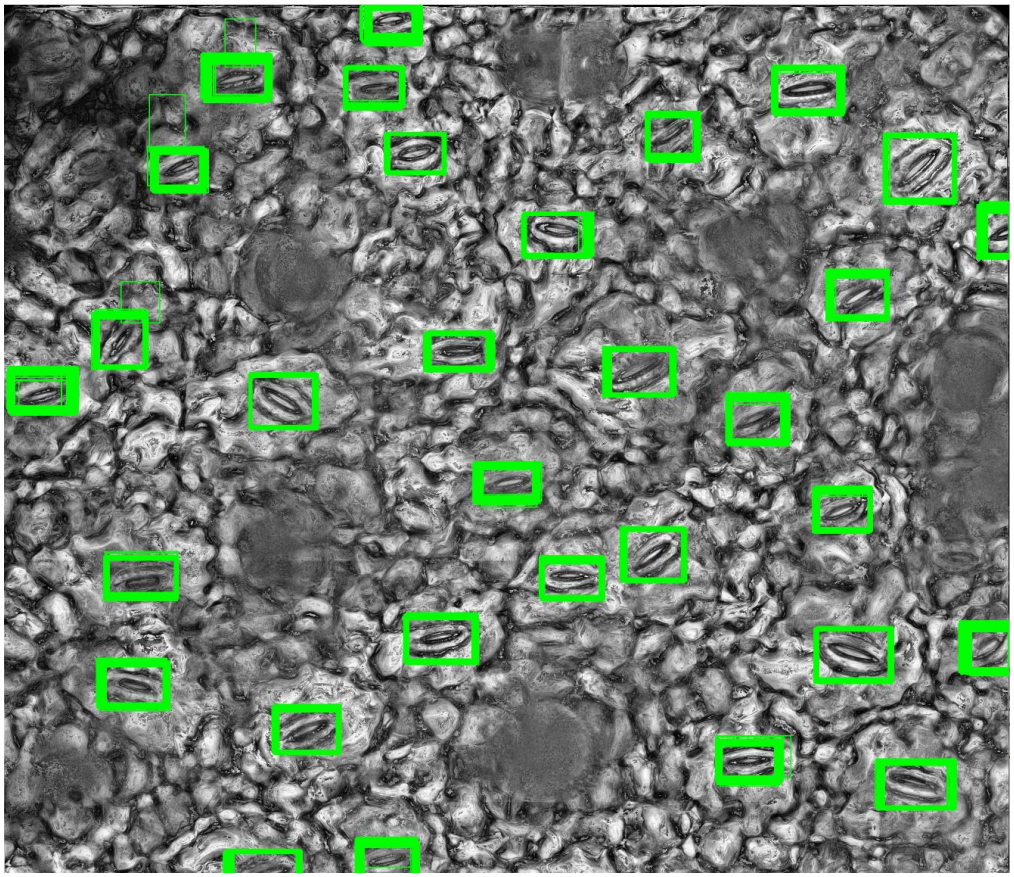


Figure 5–12: Output of the find overlap drawn on a detection

A possible solution for this is to create an augmented non max suppression. Where the images or all predictions are fed to that specific method, together with a certain threshold level. The non max suppression will again filter out all the overlapping images and return only the predictions/bounding boxes for each stoma.

Chapter 6 – Evaluation of yoloV7

The final part of this Thesis is focussed on testing the models created during the project. The test phase is a critical point in the research and creation of a model. Since this is the first moment the model will be tested on a real-world case, ideally one with the applied use in mind.

From earlier chapters it is stated that the testing images should be never-seen images. By having never seen images the detections are a true result of feature detection. This means a new set of images has been annotated from the microscope images. The species are tomato, chrysant and a new one cucumber. To fully test how each model performs a test will be performed on all possible test datasets, ranging from full to only cucumber.

After the testing an evaluation will be done. The evaluation will firs observe trends amongst models, state these observations. The best performing model can be selected from the list by simply looking at the model with the highest metrics. However, there are possibly more interesting effects of training on different models. Therefore, a statistical analysis will be done to statistically validate observations made.

6. 1 - Test set up

To be more precise, a set of 9 stacks are collected, a stack is a set of images all taken from the same part of the leaf surface, only at a different moment in time. The three species all have 3 stacks. From each stack 11 images were extracted, the images ranged from open to close stomata. Additionally, the images were also selected on clearness, as some images were out of focus, or not aligned. To ensure a fair stack representation the images taken were roughly evenly spread throughout the stack, this meant that for example with a total stack length of 119, every 12^{th} image was selected.
The stacks have images that were never seen by the model, this means that if the model just trained to recognise the fed training images, it would be noticeable, and the overall performance of the model should be lower. If the model truly learned the features of stomata, the performance should be almost the same for validation tests after each training epoch.

After the image collection the images were annotated using roboflow [52]. Since the images were pure test images, no augmentations are needed, as the test should represent the real-world use case. The annotated images are all tagged within roboflow to make sure specific species can be filtered out afterwards. With this different test on different species can be made.Figure 6–1 below shows the three species for the test dataset, from left to right: *Tomato,* chrysant, cucumber. There is a clear noticeable difference amongst stomata between images. The chrysant clearly has the biggest stomata. The stomata in cucumber images are a bit more rectangular, and finally the stomata from the tomato images are small.

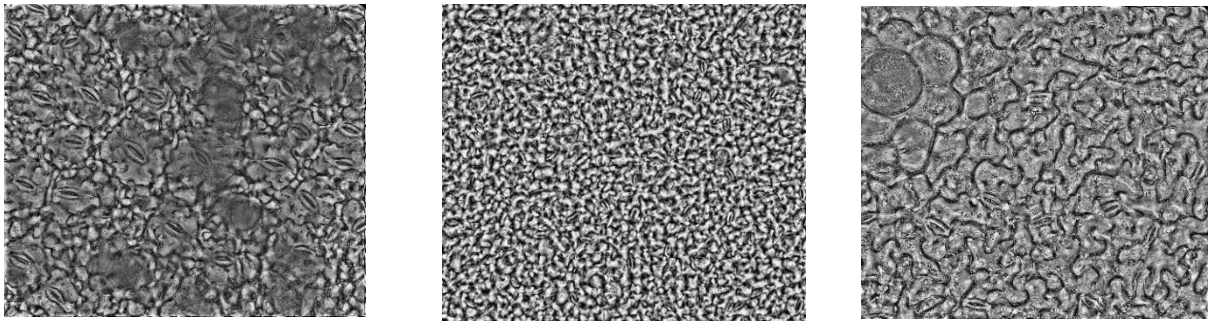


Figure 6–1:The three species, from left to right: chrysant, tomato, cucumber

After the images were annotated the testing could begin. During the training chapter is described multiple models reached via multiple methods were trained. By performing a test on the models, a true performance metric can be created, and models can be compared. By comparing the models, the research questions will be answered.

Yolov7 has a specific function in the file test.py to test the images, after each test the metrics are shown in the Terminal. Each test creates additional charts, that show how the model performs on different confidence levels. It also shows a final, precision recall curve, which really indicated how good or poor a model is. The first sets of tests are on the different training methods.

During testing some interesting results rolled out, the full test dataset gave results that were worse during validation. The results were observable lower, after looking at the number of labels per plant species, a revealing discovery was made. The tomato has 2007 which might be impactful as well. Therefore, the test will be performed on multiple tests set, to see if this impact is significantly impacting the results of the tests. Each test set has an excluded specie, or only exists of 1 specie overall. This created the following test setup, where each model, each training method, and each test_set is tested. Resulting in 3*4*6 (72) datapoints. Table 6-I shows the full test setup

Model_type	**Training method**	**Test_data**	**Images**	**Labels**
3	4	6		
YoloV7	Default	All	99	3701
YoloV7-tiny	Target	No_sl	66	1694
YoloV7X	Default- transfer	No_Kom	66	3065
	Target - transfer	Only_CH	33	1058
		Only_Kom	33	636
		Only_SL	33	2007

Table 6-I: Test design for evaluation

The table below shows results of the tests, note that both the full test dataset (left) and the no sl test dataset is shown(right). For the table with the full results see appendix D. The boxplots shown in this chapter are only from AP0,5 and/or AP0,95. The full set of boxplots for all metrics are found in appendix E

ALL					**No SL**			
Default	**P**	**R**	**Map0.5**	**mAP0.95**	**P**	**R**	**Map0.5**	**mAP0.95**
Yolov7	0,883	0,799	0,858	0,399	0,973	0,874	0,927	0,479
Yolov7-tiny	0,884	0,803	0,874	0,403	0,941	0,877	0,921	0,474

Yolov7-x	0,889	0,751	0,837	0,394	0,965	0,911	0,947	0,501

Target	**P**	**R**	**Map0.5**	**mAP0.95**	**P**	**R**	**Map0.5**	**mAP0.95**
Yolov7	0,868	0,713	0,791	0,361	0,944	0,818	0,893	0,463
Yolov7-tiny	0,862	0,79	0,837	0,378	0,924	0,858	0,9	0,467
Yolov7-x	0,904	0,72	0,818	0,377	0,973	0,832	0,91	0,471

Default-Transfer	**P**	**R**	**Map0.5**	**mAP0.95**	**P**	**R**	**Map0.5**	**mAP0.95**
Yolov7	0,875	0,727	0,801	0,375	0,94	0,78	0,86	0,455
Yolov7-tiny	0,879	0,81	0,882	0,415	0,944	0,878	0,932	0,482
Yolov7-x	0,859	0,707	0,77	0,348	0,907	0,766	0,848	0,437

Target-Transfer	**P**	**R**	**Map0.5**	**mAP0.95**	**P**	**R**	**Map0.5**	**mAP0.95**
Yolov7	0,864	0,775	0,822	0,366	0,962	0,769	0,834	0,439
Yolov7-tiny	0,853	0,806	0,847	0,397	0,941	0,852	0,906	0,476
Yolov7-x	0,874	0,758	0,83	0,377	0,907	0,766	0,848	0,437

6.2 – The best possible training method

Let’s start this descriptive analysis with finding the best training method from the 4 proposed methods. As the data has many independent variables the analysis needs to be split up, otherwise creating, and analysing data visualisations is near impossible. To get fair comparison boxplots and heatmaps are made, to analyse the data distributions and the means. The boxplot Figure 6–2 shows the mAP at 0,5 ioU and mAP 0,95 at ioU per training method, note that this boxplot contains all test data

entries.

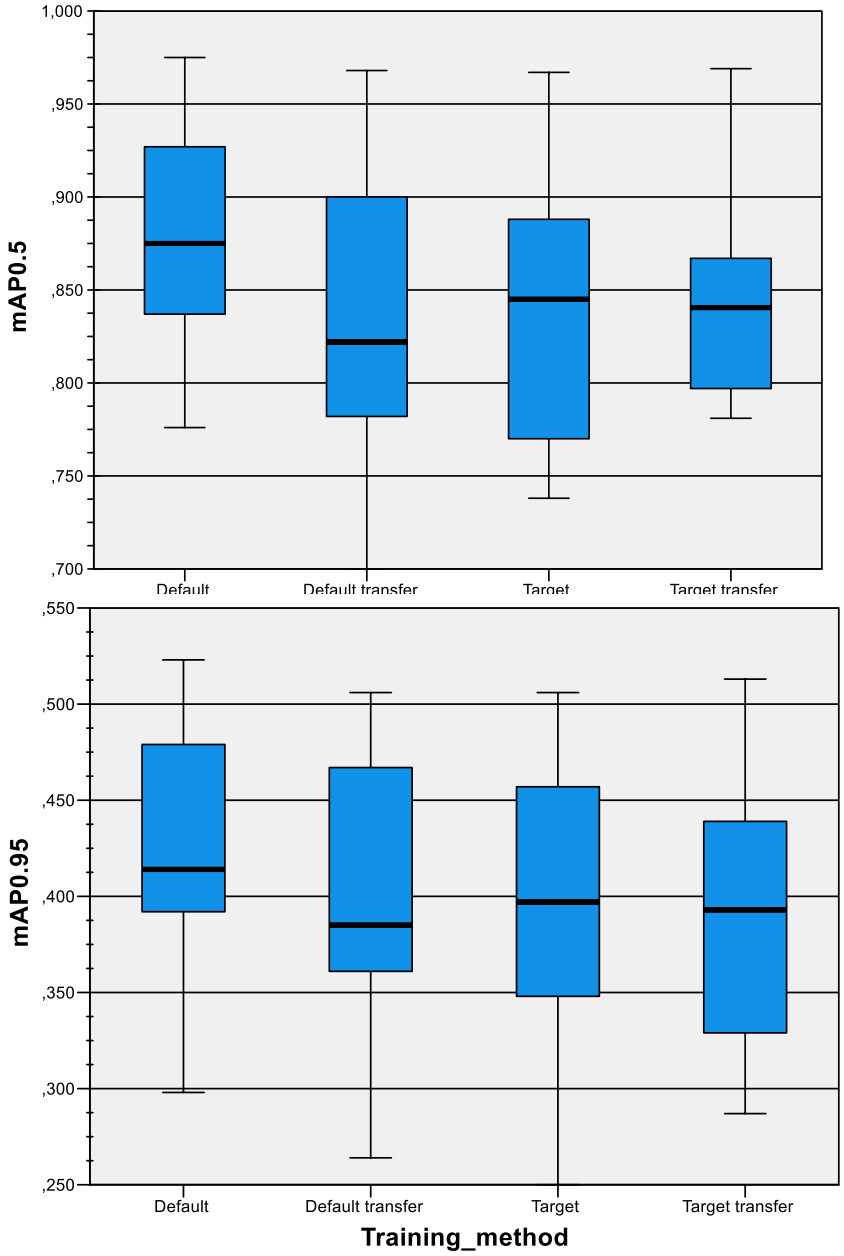


Figure 6–2: Boxplot of AP 0,5 and 0,95 per training method

The figure Figure 6–2 shows a clear higher median for the Default training method, this is the case for both metrics. With a higher median the assumption that all values are higher in the default training compared to the other training methods. The whole box, thus 50% of the values from default is higher compared to the other boxed. The highest value (peak of the whisker) is slightly the highest in the default training, although target training is close 2nd to default training. These observations lead to the final assumption that there is some indication that the default training method proves to be the best training method for the test dataset

6.4 - The most optimal model

The next step to determine the best model is to see which model performed the best overall. There are two approaches possible. The first approach is to look at each model individually, and not group/fiter

on a specific training method. The second approach is to group/filer on training method. Both approaches will be executed, to completely understand the results of all tests.

The first approach will look at the boxplots of each model, there will be no filtering or grouping. This means that all training methods and all test sets are included. Figure 6–3 contains the mAP at 0,5 ioU and mAP 0,95 at ioU per training model.

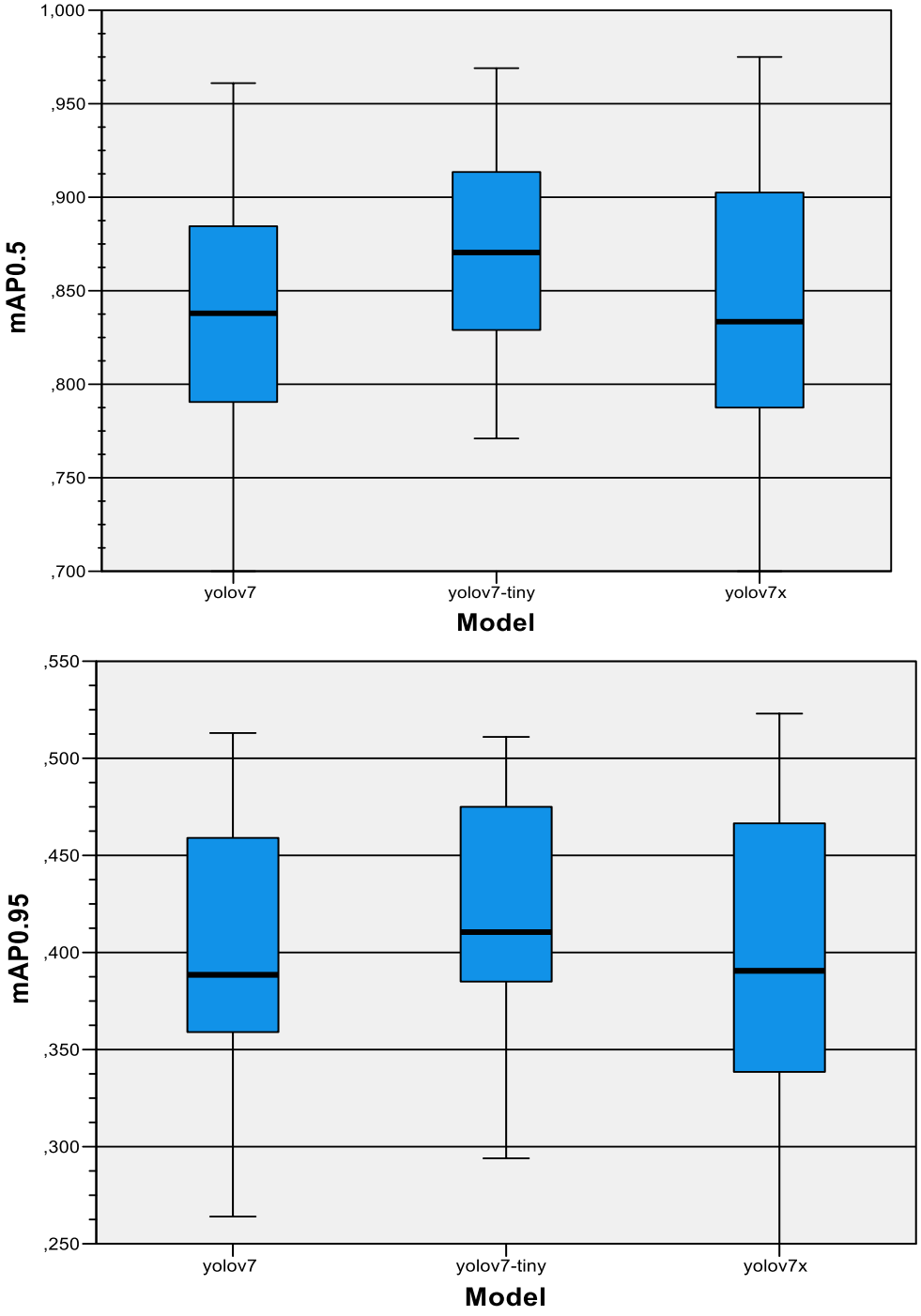


Figure 6–3: Boxplot of AP 0,5 and 0,95 per model

Each box plots some interesting things. The median of yolov7-tiny is in both plots higher than the other models. The box itself is also reaching higher than the other models. This indicated that on average yolov7-tiny scores better on AP then the other yolo model. However, looking at the top of the whiskers another interesting result can be observed. The yolov7x models seems to score the best for both AP metrics. The model has quite a high variance, as the bottom of the whisker is not visible on

this image. This can indicate that on average the yolov7-tiny model generalises better, which means that it learns more general stomata features. And on the other the bigger yolov7x model learns more specific features, which impacts its performance, if exceptional cases are presented.

The second approach will look at the boxplots of the test results grouped by learning method. By grouping this variable, the effect of the training method on individual model performance can be observed. Again, the plot is made with the AP0,5 and AP0,95.

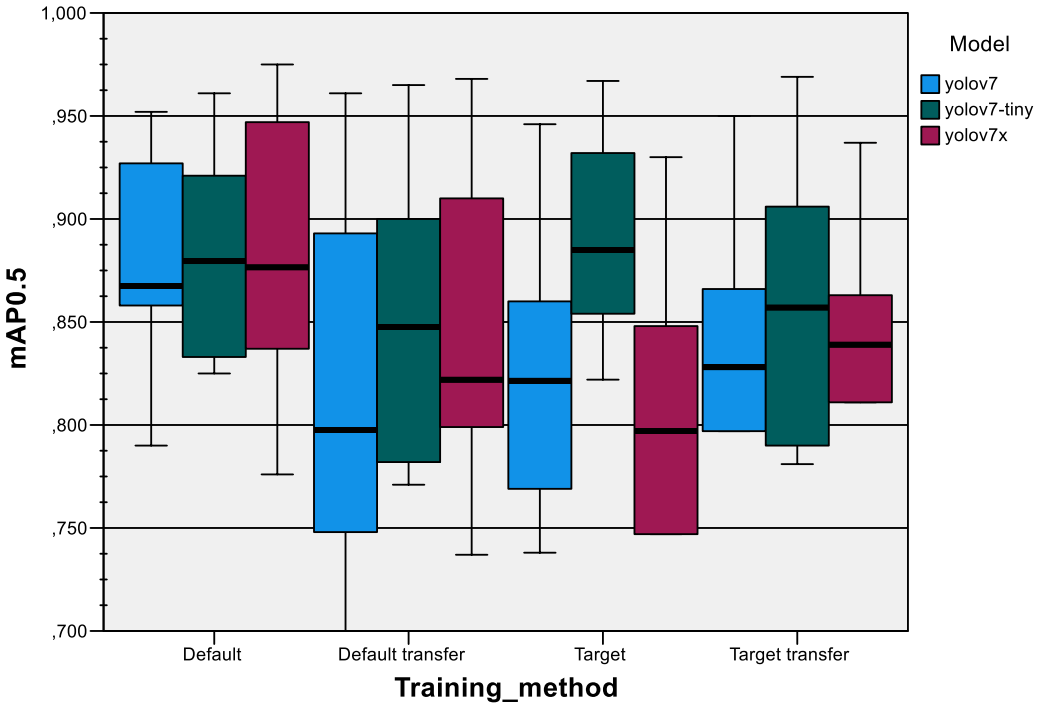


Figure 6–4: Boxplot of AP0,5 for test result grouped by model

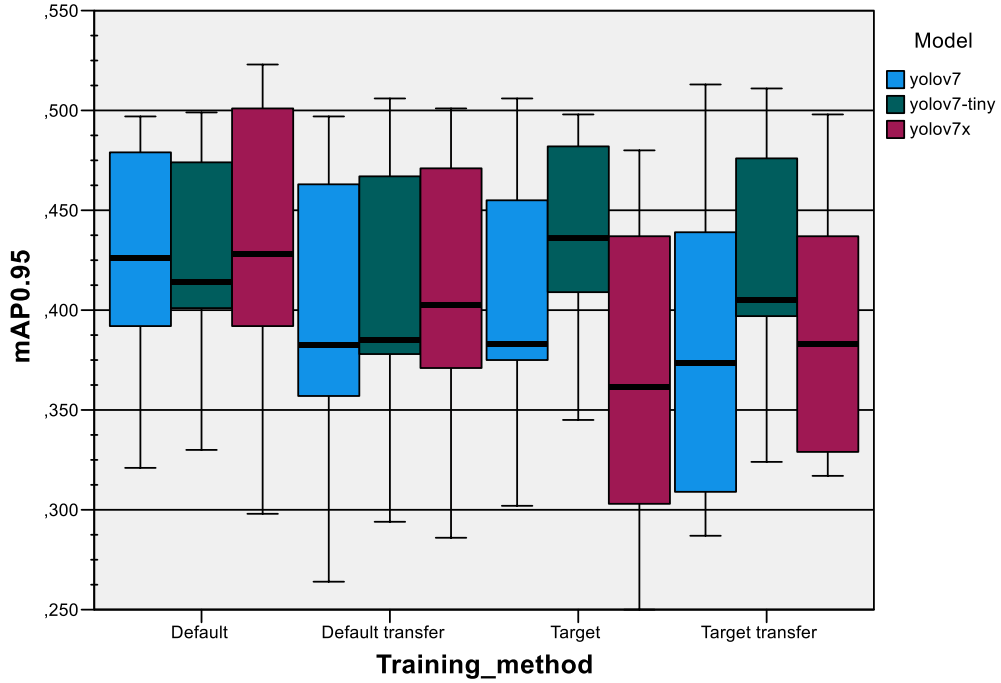


Figure 6–5: Boxplot of AP0,95 for test result grouper per training method

We start with observing the images in general. In Figure 6–4 and Figure 6–5the model yolov7-tiny shows indications of the highest average, this is also reflected in this image. Throughout each training method the boxes for yolov7-tiny model all are all constantly on the same relative height. With the in the targeted training the box even surpasses the other models. The median form the yolov7-tiny model at an AP of 0,5 is in all cases the highest. In the cart from AP 0,95 this is not the case. Here the median is only the highest for the cases with target and target transfer as training method. Again, this observation indicates the prior stated finding, yolov7-tiny is better in generalizing. The model scores for each training method consistently average, which is not the case for the yolov7x and v7 models. Again, v7x shows the highest variance in the charts.
The whiskers reveal quite an interesting observation. The highest value for yolov7x is within the default training, all the other training types show a lower peak value. These peak values are even lower then yolov7-tiny and in some cases the lowest of the three models. From this the earlier observation that yolov7x is possibly the best model starts to score a bit worse. If the best models are selected on pure performance, then yolov7x is still a good competitor.

5.5 - Analysis of impact of plant species.

In all the prior showed graphs the data still covers all the testing datasets. However, the test dataset also had an influence on model performance, as it is still an independent variable. Therefore, a series of observations will be made on the different test_sets. First, we start with a general observation. Figure 6–6 is the boxplot of all test results by test_set, note that the this contains all training methods and all models.

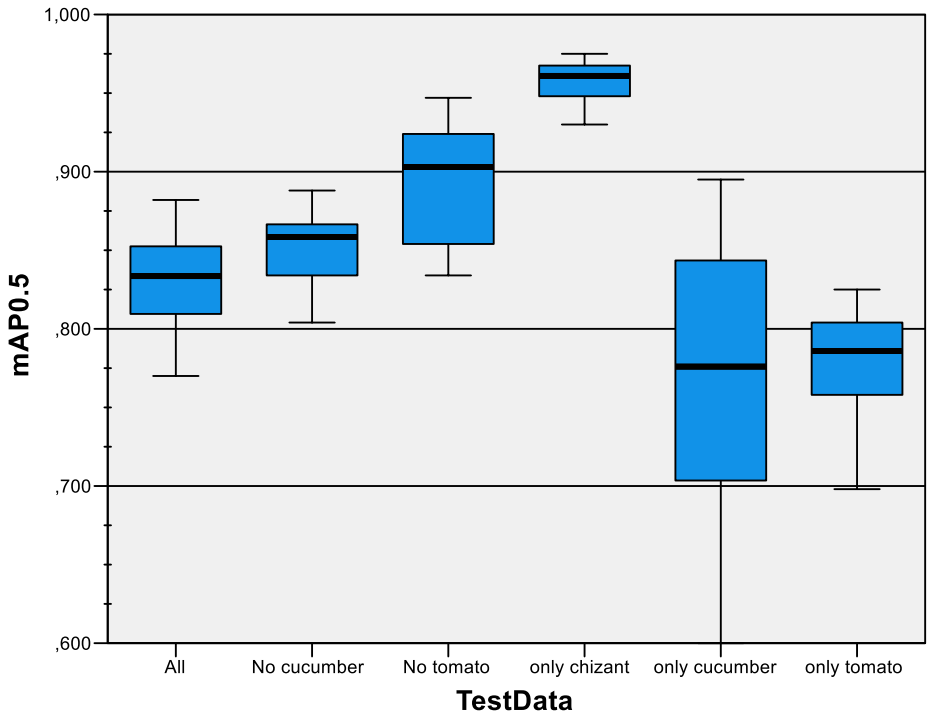


Figure 6–6: Boxplot of AP 0,95 per test_set

A quick observation can be made from the image, the only chrysant dataset has the highest average precision. The median is the highest from each test dataset. The scores from all set are varying a lot, this is also the case for only cucumber. Therefore, it can be assumed that this variance in all is mainly caused by the only cucumber test set. Figure 6–6 does not show the impact of each model. Therefore Figure 6–7 is made.

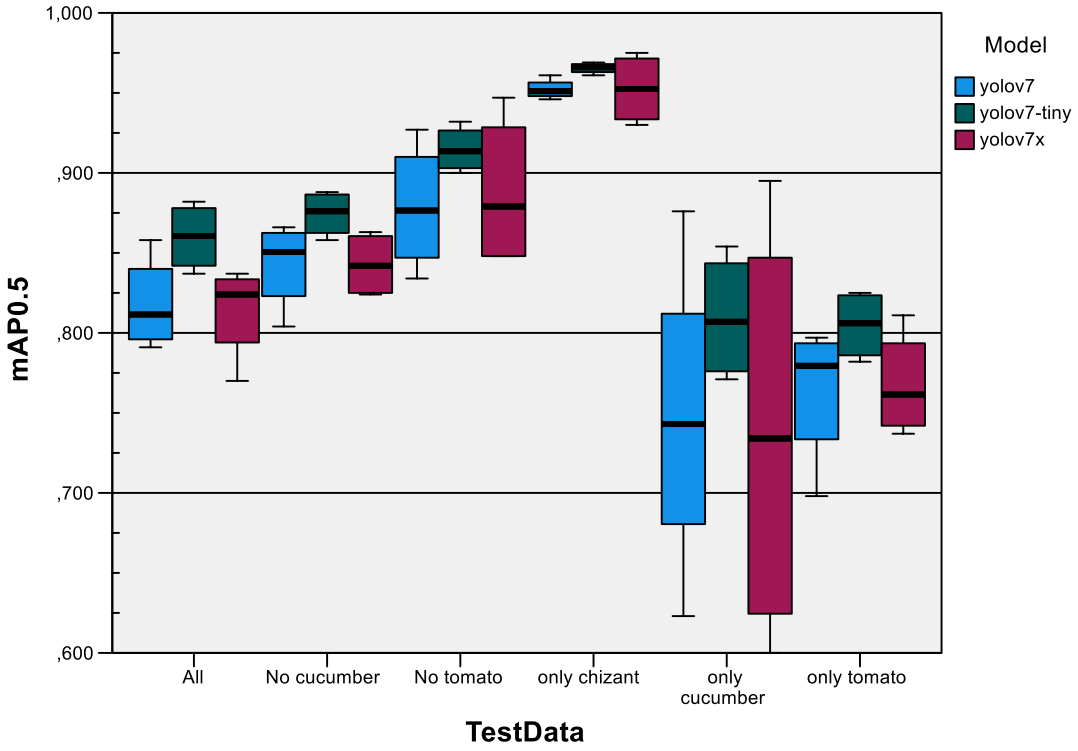


Figure 6–7: Boxplot of AP0,5 per test_set grouped by model

The figure does give an insight full overview of what each model scores for average precision 0,5 at each test set for every training method. From this some new observations can be deducted and combined with earlier images some stronger assumptions are created. First start with the earlier observation of yolov7-tiny performing quite well on average. In IMG INSERT yolov7-tiny again scores on average quite high, in some cases even the highest. For all, no cucumber, and only tomato the model scores the highest. The median is even well above the whiskers of yolov7x. This again strengthens the assumption that on average yolov7-tiny scores the best.

Yolov7x on the other hand does have some interesting whiskers (top values). Without tomato the model seems have the highest score on average precision. This is also the case for cucumber. However, cucumber does have quite a high variance for yolov7x. From

The three species are shortly covered to observe and reason results for each specific specie.

5.5.1 - Observations chrysant

This specie shows the best performance, for almost every training method. Yolov7x on default training is has the highest performance, with a mAP 0,5 of 0,975 and 0,95 of 0,523. The precision and recall are also well above 0,95. Again yolov7-tiny score quite high on average for every training method. The high scores are truly phenomenal, however not entirely surprising. The models all have trained quite a lot on the chrysant, the used dataset contained this specie explicitly. Additionally, this specie is generally known for its recognisable stoma shape, and therefore the most trained on/ benchmarked specie within this research field.

If the model is picked that performed the best on this specie, yolov7x would be the best candidate. The model has both a high precision, recall and average precision for both IoU levels.

ONLY CH					
Default	**P**	**R**	**F1**	**Map0.5**	**mAP0.95**
Yolov7	0,975	0,94	0,95718	0,952	0,497
Yolov7-tiny	0,969	0,95	0,959406	0,961	0,499
Yolov7x	0,981	0,955	0,967825	0,975	0,523
Target	**P**	**R**	**F1**	**Map0.5**	**mAP0.95**
Yolov7	0,973	0,926	0,948918	0,961	0,497
Yolov7-tiny	0,979	0,937	0,95754	0,965	0,506
Yolov7x	0,981	0,944	0,962144	0,968	0,501
Default transfer	**P**	**R**	**F1**	**Map0.5**	**mAP0.95**
Yolov7	0,968	0,909	0,937573	0,946	0,506
Yolov7-tiny	0,978	0,945	0,961217	0,967	0,498
Yolov7x	0,955	0,892	0,922426	0,93	0,48
Target transfer	**P**	**R**	**F1**	**Map0.5**	**mAP0.95**
Yolov7	0,98	0,915	0,946385	0,95	0,513
Yolov7-tiny	0,976	0,945	0,96025	0,969	0,511
Yolov7x	0,952	0,904	0,927379	0,937	0,498

Table 6-II: Test results for only chrysant

5.5.2 - Observations cucumber

From all the test images cucumber was a never seen plant species. The result of this is quite visible for all the different models. All models show a high variance in Figure 6–7 and this specie has the lowest median.

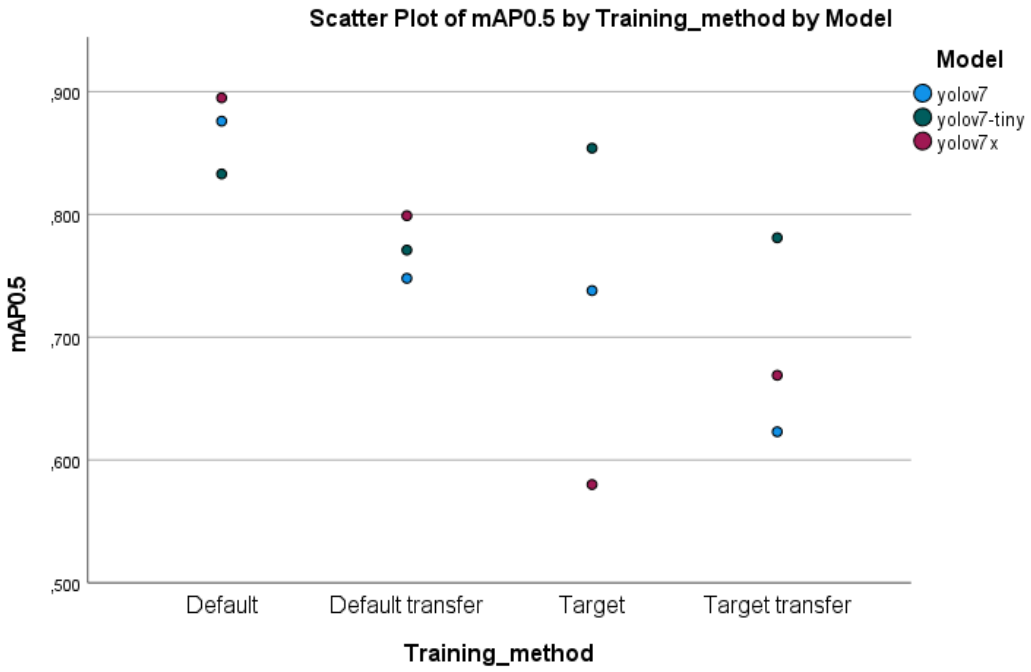


Figure 6–8: Scatterplot of mAP0,95 grouped by training method, coloured by model and filtered on only_sl

This scatterplot from Figure 6–8shows how each training method influenced each model on the cucumber dataset. Again the default in combination with yolov7x seems to be the best pick, as it scores on average precision quite nicely. The scatterplot reveals that the high variance as seen in Error! Reference source not found. is caused by both the target trained models.
ombining observations one can argue that the default trained models generally generalise more on stomata shape, and less on the specific features from the stack images. This statement can be backed up by the drop for all models after transfer learning, where the stack images were used for training and validation.

Overall cucumber scores respectively well, albeit that for the targeted learning the performance is relatively low. This shows that other species can be recognised, with a clear drop in performance as this drop in performance is visibly there.

5.5.3 - Observations tomato

There is a clear increase in all metrics between the tests with and without the tomato images. For each training method and each model there is a clear increase in precision recall mAP0,5 and mAP0,95. This is quite an interesting result with multiple possible reasons, since tomato is in the training dataset the argument that this specie is never seen cannot be made. Figure 6–7 further indicated the difference between the ALL and No_sl test dataset. In this figure the only_sl also indicates that the tomato images are clearly reducing the performance of all models.

- **The tomato images are not fairly represented**
  This is true for the default dataset, where a lot of images are form chrysant stomata, as these are the most recognisable and well-studied stomata. For the other training methods however, there were 3 sets of tomato images in the training set, with the additional given images with tomato are small.
- **The drawn bounding boxes are not correct**
  This can be a valid reason. The stomata are small on the tomato images, Figure 6–9 shows how small. As a nonprofessional biologist, it is hard to successfully find all stomata, and also only find stomata. Together with the fact that there are more stomata in tomato images in comparison to other images, one can state this assumption must be considered.

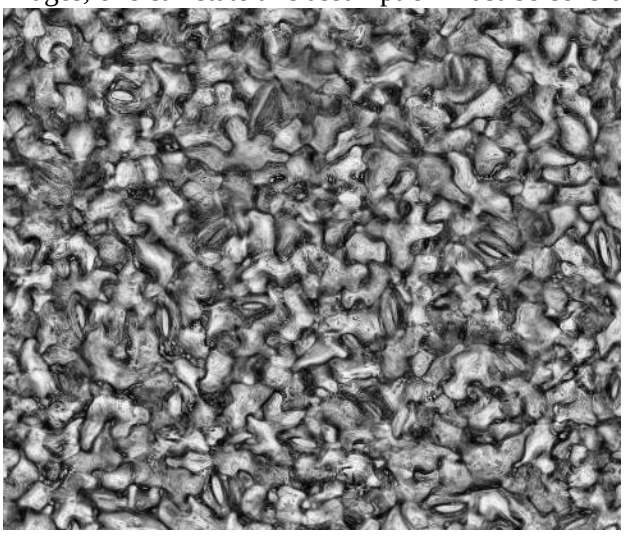


Figure 6–9: A 4x zoomed segment of tomato image from microscope

- **Tomato has small stomata, therefore hard to detect**.
  From all reasons the fact that the stomata in the tomato images are small is quite evident. However besides hard to find, there is another flaw. The Yolo models might have a direct influence on this as well. During training the images are all compressed to a square of 960x960, this is done to reduce the computational load. The GPU used was only limited to use this resolution, therefore all images were trained on a smaller scale then the 'real' input resolution. When the stomata in the image are already small, a compression will even further reduce the pixel count of a single stomata. Figure 6–10 below shows the direct effect of this compression on a stoma in a tomato image.

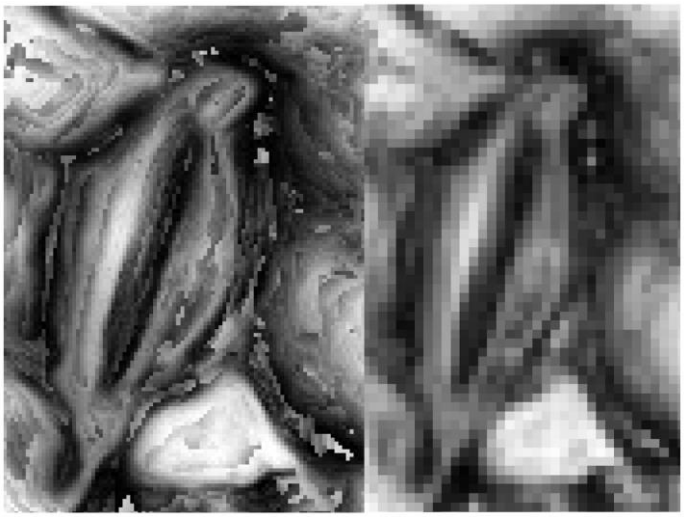


Figure 6–10: Left side original input image, right side the image cropped by yolo

5.7 – Statistical analysis

Prior parts of this chapter only focussed on observations. The observations made are statistically proven yet. This chapter will attempt to test if model has influence on the test results, and if training method has influence on test results.

The results from the test sessions are not all dependent on each other. This means that precision and recall are independent. However, both AP metrics are dependent both on precision and recall but also dependent on IoU. This created a quite complex situation, since there are also 3 different independent variables, namely model, training method and test_set. First, we do a three-way factorial ANOVA on precision and recall.

To be able to perform an ANOVA test certain assumptions must be met. From a descriptive analysis in spss the only the dependent variables are tested. The first test of normality without setting any independent variables already showed something concerning, as there is already a slight indication of no normal distribution. The 2nd test on normality with model set as an dependent variable already showed that the distribution was not normal as seen in Table 6-III

Tests of Normality

		Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Model	Statistic	df	Sig.	Statistic	df	Sig.
Precision	yolov7	,173	24	,063	,902	24	,024
	yolov7-tiny	,148	24	,190	,941	24	,173
	yolov7x	,086	24	,200*	,954	24	,332
recall	yolov7	,089	24	,200*	,970	24	,667
	yolov7-tiny	,149	24	,181	,934	24	,119
	yolov7x	,130	24	,200*	,956	24	,358
ThmAP0.5	yolov7	,095	24	,200*	,963	24	,510
	yolov7-tiny	,096	24	,200*	,955	24	,342
	yolov7x	,113	24	,200*	,950	24	,273
mAP0.95	yolov7	,102	24	,200*	,958	24	,391
	yolov7-tiny	,113	24	,200*	,952	24	,302
	yolov7x	,099	24	,200*	,967	24	,592

Table 6-III: Normality test with model as independent variable

The test on normality is not met, which means that for each model there is no normal distribution. Which makes is impossible to do an ANOVA as one of the assumptions is violated. Therefore, a non-parametric test is used. This however means that the statistical significance is weaker if the null hypothesis is rejected. The non-parametric test used is the Kruskal-Wallis test. The tables below show the tests output from spss in grouped by model, training method and test_set

Test Statistics[a,b]

	Precision	recall	mAP0.5	mAP0.95
Kruskal-Wallis H	,016	3,666	3,134	1,890
df	2	2	2	2
Asymp. Sig.	,992	,160	,209	,389

a. Kruskal Wallis Test

b. Grouping Variable: Model

Test Statistics[a,b]

	Precision	recall	mAP0.5	mAP0.95
Kruskal-Wallis H	2,606	3,934	4,457	2,100
df	3	3	3	3
Asymp. Sig.	,456	,269	,216	,552

a. Kruskal Wallis Test

b. Grouping Variable: Training_method

Test Statistics[a,b]

	Precision	recall	mAP0.5	mAP0.95
Kruskal-Wallis H	57,977	46,969	51,954	58,056
df	5	5	5	5
Asymp. Sig.	<,001	<,001	<,001	<,001

a. Kruskal Wallis Test

b. Grouping Variable: TestData

The first Kruskal-Wallis H test showed that there was no statistically significant difference in precision, recall, mAp0,5 and mAP0,95 between the different models.
The second Kruskal-Wallis H test showed that there was no statistically significant difference in precision, recall, mAp0,5 and mAP0,95 between the different training methods.
The third Kruskal-Wallis H test showed that there is statistically significant difference in precision, recall, mAp0,5 and mAP0,95 between the test methods.

This means that there is only a significant differenct between the test_sets. There is no significant difference on model and training_method. Therefore the observations made are purely observations and not statistically proven.

5.6 - Evaluation - to conclude

The statistical analysis concluded that the differences between models and training methods are not significant, this can potentially be solved by performing more tests with different stacks. For now, the observations are discussed but note no statistical significance is proven. The boxplots presented in this chapter aid the evaluation of the tests performed on each model. With the made assumptions throughout this chapter one can pivot towards a certain model in the selection of the models. Before a specific model is labelled as the 'perfect' one the observations made are listed below.

- The default training generally is the best training method.
- Yolov7-tiny scored high on AP0,5 and AP0,95 on average.
- Yolov7x has the highest AP for an ioU 0,5 and 0,95 with default training and chrysant as test set.
- Yolov7x with default training scores the best on cucumber.
- Tomato reaches the limits of yolo, especially for yolov7x.

Defining the best model can be done on multiple ways. One way is to simply look at the best performance for all dataset and conclude that is the best possible model. However, one can also argue that differences are amongst the stomata, especially with the tomato stomata, as the size does vary a lot compared to cucumber and chrysant.

With the first methodology the preferred model is yolov7-tiny training with default training. This model scored consistently average for every test_set. YoloV7x trained with the default dataset is the picked model with the second way of reasonig, the model scores perfectly for chrysant, and relatively nice for cucumber, with the only weakness tomato.

In the end, incorporating tomato negatively impacted the overall performance of the model, hence the second approach proved to be the most effective and should be adopted. The yolov7x model learned the stomata shape the best, and different more smaller shapes are simply harder for yolov7 overall.

Chapter 7 - Conclusion

Precision agriculture is a relative and important topic. Climate change is changing the environment, resulting in more precise and sensor driven agriculture. If it is known what exactly a plant needs, or a plant can be bread to specifically live in a certain environment, the emerging problem of food production in the world could be solved. Automatic stomata detection can speed up current research. Stomatal behaviour, and stomatal density and other aspects are still not fully understood. However, they are certainly a key aspect of precision agriculture, therefore understanding stomata is of the upmost importance. Detecting stomata by hand is currently a repetitive and time-consuming task if this can be sped up, the research output of this field can drastically increase.
The landscape of object detection has changed significantly throughout the last years, there has been a paradigm shift from computational modals towards deep learning methods. Deep learning combines the need for human feature extraction and classifiers into one automated pipeline. Currently there are automated programs that can detect and count stomata, however these are still in development.

The first research question is: How can existing knowledge on computer vision techniques be used to create a real time automatic stomata detection algorithm?

From the first chapters it is found that with yoloV7 real time stomata detection is possible, as YoloV7 has fast and accurate inference. Additionally, the mask_rcnn model is used, however this model had no real evaluation due to time constraints.

The second part of this project explored the trainability and usability of YoloV7 for stomata detection, all to answer the second research question: Can a deep learning algorithm be trained that is able to detect stomata on different plant breeds with a precision of 95%?

YoloV7 is currently the state-of-the-art model therefore, a more elaborate training and testing method was developed. Object detection with convolutional neural networks seems like a daunting task. With a structured and well though workflow this task is possible. The workflow ranges from data collection to model evaluation. The collected data should be as diverse as possible, where image count and difference between images is a key detail. For this research an open-source image library is used for the default targeted training. The models are also trained on additional images gathered directly from the microscope.

With the dataset used from roboflow in combination with the images form the microscope a yolov7 model can detect stomata with a high precision.

The default dataset and training from this project has a strong indication to be the best method. This is mainly caused by the number of different images within the dataset; this enables the model to clearly extract key stomatal features. In combination with the targeted images in the dataset, the model can get a strong understanding of the microscope images.

During this project multiple yoloV7 models are trained and tested and evaluated. The evaluation of each model is done with a test set containing never seen images from three different species. Chrysant and tomato, are the species that were used during training. Tomato was a never seen specie presented to the model in the test dataset.

There are two good competitors for the best model. Both models are trained with default training. For a more average model that scores well on all test images, the yolov7-tiny model is the most optimal. Table 7-I shows the scores for the two models.

	All	P	R	F1	Map0.5	mAP0.95
Yolov7-tiny		0,884	0,803	0,842	0,874	0,403
Yolov7-x		0,889	0,751	0,814	0,837	0,394
	Only CH					
Yolov7-tiny		0,969	0,95	0,959	0,961	0,499
Yolov7-x		0,981	0,955	0,968	0,975	0,523
	Only Kom					
Yolov7-tiny		0,896	0,748	0,815	0,833	0,425
Yolov7-x		0,938	0,84	0,886	0,895	0,462
	Only tom					
Yolov7-tiny		0,816	0,76	0,787	0,825	0,33
Yolov7-x		0,788	0,74	0,763	0,776	0,298

Table 7-I: Test results for default training

The final research question was not answered during this project. Due to time constraints and the amount of testing conducted on the yolov7 models, the task of combining stack images is not finished. However, there is a method which is developed that filters out certain bounding boxes, but not robust enough for testing. By counting the overlap and only save the overlapping bounding boxes some false positives can already be filtered out. With an altered nom max suppression this method can become more robust, and suitable for testing.

6.2 - Discussion & future research

The project really showed some direct influences of different training methodologies. Training has a real influence on refining your model, as the perfect training in combination with the perfect model can really squeeze out the last bit of possible improvement.

One of the reasons why the test dataset scored lower than the training is the difference in data, and overall data availability. With the stack structure the model receives quite repetitive images, although they are not entirely similar the bounding box density is not uniformly distributed throughout the whole dataset. YoloV7 luckily uses the cut mix manner to create unique training images. Future research should try to test the influence of increasing the amount of truly different stacks. There may be no influence at all, which means that the models trained in this project are on peak performance on the targeted images.

During this project the focus was more on object detection rather than segmentation. Yolov7 does have segmentation possibilities. Future research can create a model that is able to segment, based on the default training methodology created by this project. There is a version of the microscope dataset that contains masks, therefore some annotation steps can be skipped.

The tomato images reduced model performance drastically for all types of training. yoloV7 is known for bad detections on small images, in addition with the rescaling this can indicate that the limit of the model is reached with the tomato images. Future research can consider using Slicing Aider Hyper inference [54], for the tomato images. This reintroduces a sliding box to the inference. In essence the image is cut into multiple images, and all inferenced on their own. From the github page this method seems to increase detection of smaller objects relatively well.

In this project yoloV7 is used, at the start of this project up until the beginning of the year this model was the state-of-the-art. Currently yoloV8 [55] is released, this model is from the same authors as yoloV5. yoloV5 is created with a more business driven approach, where usability is a key aspect, this model has many tools developing and training yoloV5. YoloV8 has this same approach since it is made by the same authors. Yolov8 claims to have a higher score on the coco dataset (a widely

accepted test to validate a created model). Ease of deployment is something that yoloV7 does not have, therefore future research can focus on re-doing the default training on yoloV8 and compare results.

The yolov7 models will all be handed over to Tom together with a small GUI. The GUI can be developed further to also accept segmentation models, since yolov7 does support this. Additionally, there a future project can refine the method to increase overall accuracy even more. And add possible functionality to the GUI.

Chapter 8 Works Cited

[1] G. Leng, Q. Tang and S. Rayburg, "Climate change impacts on meteorological, agricultural and hydrological droughts in China," 3 2015. [Online].

[2] B. A. Keating, M. Herrero, P. S. Carberry, J. Gardner and M. B. Cole, "Food wedges: Framing the global food demand and supply challenge towards 2050," *Global Food Security,* vol. 3, no. 3-4, pp. 125-132, 11 2014.

[3] Plantenna, "plantenna," 4Tu, [Online]. Available: https://www.4tu.nl/plantenna.

[4] M. M. Blanke and A. R. Belcher, "Stomata of apple leaves cultured in vitro," 1989.

[5] L. Costa, L. Archer, Y. Ampatzidis, L. Casteluci, G. A. Caurin and U. Albrecht, "Determining leaf stomatal properties in citrus trees utilizing machine vision and artificial intelligence," *Precision Agriculture,* vol. 22, no. 4, pp. 1107-1119, 8 2021.

[6] A. P. Singh and L. M. Srivastava, "The Fine Structure of Pea Stomata".

[7] R. Hedrich and S. Shabala, "Stomata in a saline world," *Current Opinion in Plant Biology,* vol. 46, pp. 87-95, 12 2018.

[8] "Stomata," Science facts, [Online]. Available: https://www.sciencefacts.net/stomata.html.

[9] R. P. Spiegelhalder and M. T. Raissig, "Morphology made for movement: formation of diverse stomatal guard cells," *Current Opinion in Plant Biology,* vol. 63, 10 2021.

[10] M. Koornneef and D. Meinke, "The development of Arabidopsis as a model plant," *Plant Journal,* vol. 61, no. 6, pp. 909-921, 3 2010.

[11] L. a. C. Cortes, "THE MNIST DATABASE of handwritten digits," 1999. [Online]. Available: http://yann.lecun.com/exdb/mnist/.

[12] S. Visalpara, "How Do Computers See an Image," 15 January 2016. [Online]. Available: https://savan77.github.io/2016-01-15-image/.

[13] L. Deng and D. Yu, "Deep learning: Methods and applications," *Foundations and Trends in Signal Processing,* vol. 7, no. 3-4, pp. 197-387, 2013.

[14] F. Chollet, "M A N N I N G".

[15] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way," 15 December 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[16] Y. Lecun, L. Eon Bottou, Y. Bengio and P. H. Abstract|, "Gradient-Based Learning Applied to Document Recognition".

[17] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks".

[18] O. Russakovsky, "ImageNet Large Scale Visual Recognition Challenge," 9 2014. [Online]. Available: http://arxiv.org/abs/1409.0575.

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going Deeper with Convolutions," 9 2014. [Online]. Available: http://arxiv.org/abs/1409.4842.

[20] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 9 2014. [Online]. Available: http://arxiv.org/abs/1409.1556.

[21] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition".

[22] G. Huang, Z. Liu, L. Van Der Maaten and K. Q. Weinberger, "Densely Connected Convolutional Networks".

[23] C. Kulkarni, "Learning Rate Tuning and Optimizing," 28 February 2018. [Online]. Available: https://medium.com/@ck2886/learning-rate-tuning-and-optimizing-d03e042d0500.

[24] S. Ruder, "An overview of gradient descent optimization algorithms," 9 2016. [Online]. Available: http://arxiv.org/abs/1609.04747.

[25] 21 Dec 2022. [Online]. Available: https://hasty.ai/docs/mp-wiki/metrics/precision-recall-curve-and-auc-pr.

[26] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick and P. Dollár, "Microsoft COCO: Common Objects in Context," 5 2014.

[27] M. William da Silva Oliveira, N. Rosa, D. Casanova, L. Felipe Souza Pinheiro, N. Rosa da Silva, R. Marta Kolb and O. Martinez Bruno, "Automatic Counting of Stomata in Epidermis Microscopic Images Spatially explicit models in data science and vice versa View project Image analysis View project Automatic Counting of Stomata in Epidermis Microscopic Images," 2014.

[28] H. Laga, F. Shahinnia and D. Fleury, "Image-based plant stornata phenotyping," *2014 13th International Conference on Control Automation Robotics and Vision, ICARCV 2014,* pp. 217-222, 2014.

[29] K. T. Duartě, M. A. De Carvalho and P. S. Martins, "Segmenting high-quality digital images of stomata using the wavelet spot detection and the watershed transform," *VISIGRAPP 2017 - Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications,* vol. 4, pp. 540-547, 2017.

[30] A. H. Aono, J. S. Nagai, G. d. S. Dickel, R. C. Marinho, P. E. de Oliveira, J. P. Papa and F. A. Faria, "A stomata classification and detection system in microscope images of maize cultivars," *PLoS ONE,* vol. 16, no. 10 October, 10 2021.

[31] S. Meeus, J. Van den Bulcke and F. wyffels, "From leaf to label: A robust automated workflow for stomata detection," *Ecology and Evolution,* vol. 10, no. 17, pp. 9178-9191, 9 2020.

[32] W. Song, J. Li, K. Li, J. Chen and J. Huang, "An automatic method for stomatal pore detection and measurement in microscope images of plant leaf based on a convolutional neural network model," *Forests,* vol. 11, no. 9, 9 2020.

[33] X. Liang, X. Xu, Z. Wang, L. He, K. Zhang, B. Liang, J. Ye, J. Shi, X. Wu, M. Dai and W. Yang, "StomataScorer: a portable and high-throughput leaf stomata trait scorer combined with deep learning and an improved CV model," *Plant Biotechnology Journal,* vol. 20, no. 3, pp. 577-591, 3 2022.

[34] C. Li, C. Xu, C. Gui and M. D. Fox, "Level set evolution without re-initialization: A new variational formulation," *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005,* vol. I, pp. 430-436, 2005.

[35] T. Dai, J. Zhang and K. Li, "Microscopic image recognition method of stomata in living leaves based on improved YOLO-X," 2022.

[36] K. He, G. Gkioxari, P. Dollár and R. Girshick, "Mask R-CNN".

[37] C.-Y. Wang, A. Bochkovskiy and H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," 7 2022.

[38] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 11 2013.

[39] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," 6 2015. [Online]. Available: http://arxiv.org/abs/1506.01497.

[40] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 6 2015.

[41] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," 12 2016. [Online]. Available: http://arxiv.org/abs/1612.08242.

[42] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," 4 2018. [Online]. Available: http://arxiv.org/abs/1804.02767.

[43] A. Bochkovskiy, C.-Y. Wang and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," 4 2020. [Online]. Available: http://arxiv.org/abs/2004.10934.

[44] G. Jocher, ""YOLOv5 by Ultralytics," 29 May 2020. [Online]. Available: https://github.com/ultralytics/yolov5/blob/master/CITATION.cff.

[45] C.-Y. Wang, I.-H. Yeh and H.-Y. M. Liao, "You Only Learn One Representation: Unified Network for Multiple Tasks," 5 2021. [Online]. Available: http://arxiv.org/abs/2105.04206.

[46] C. Li, L. Li, H. Jiang, K. Weng, Y. Geng, L. Li, Z. Ke, Q. Li, M. Cheng, W. Nie, Y. Li, B. Zhang, Y. Liang, L. Zhou, X. Xu, X. Chu, X. Wei and X. Wei, "YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications," 9 2022.

[47] H. Sanders and J. Saxe, "GARBAGE IN, GARBAGE OUT: HOW PURPORTEDLY GREAT ML MODELS CAN BE SCREWED UP BY BAD DATA," 2017.

[48] D. Rolon-Mérette, M. Ross, T. Rolon-Mérette and K. Church, "Introduction to Anaconda and Python: Installation and setup," *The Quantitative Methods for Psychology,* vol. 16, no. 5, pp. S3-S11, 5 2020.

[49] J. Sanders, E. Kandrot and E. Jacoboni, CUDA par l'exemple [une introduction à la programmation parallèle de GPU], Pearson, 2011.

[50] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. D. Facebook, A. I. Research, Z. Lin, A. Desmaison, L. Antiga, O. Srl and A. Lerer, "Automatic differentiation in PyTorch".

[51] USENIX Association., ACM SIGMOBILE., ACM Special Interest Group in Operating Systems. and ACM Digital Library., "TensorFlow: A System for Large-Scale," USENIX Association, 2005.

[52] Roboflow, "Roboflow," Roboflow, [Online]. Available: https://roboflow.com/.

[53] W. Kin-Yiu. [Online]. Available: https://github.com/WongKinYiu/yolov7/blob/main/detect.py.

[54] F. C. Akyon, S. O. Altinuc and A. Temizel, "Slicing Aided Hyper Inference and Fine-tuning for Small Object Detection," 2 2022. [Online]. Available: http://arxiv.org/abs/2202.06934.

[55] J. Glenn and A. Chaurasia, "YOLO by Ultralytics," Ultralytics, 10 1 2023. [Online]. Available: https://github.com/ultralytics/ultralytics.

Chapter 9 - Appendices

Appendix A: Conda environments

Yolov7

```yaml
name: yolov7_mask

channels:
  - pytorch
  - nvidia
  - conda-forge
  - defaults
dependencies:
  - alabaster=0.7.12=pyhd3eb1b0_0
  - arrow=1.2.3=py39haa95532_0
  - astroid=2.11.7=py39haa95532_0
  - atomicwrites=1.4.0=py_0
  - attrs=22.1.0=py39haa95532_0
  - autopep8=1.6.0=pyhd3eb1b0_1
  - babel=2.9.1=pyhd3eb1b0_0
  - backcall=0.2.0=pyhd3eb1b0_0
  - bcrypt=3.2.0=py39h2bbff1b_1
  - beautifulsoup4=4.11.1=py39haa95532_0
  - binaryornot=0.4.4=pyhd3eb1b0_1
  - blas=1.0=mkl
  - bleach=4.1.0=pyhd3eb1b0_0
  - brotlipy=0.7.0=py39h2bbff1b_1003
  - ca-certificates=2022.12.7=h5b45459_0
  - certifi=2022.12.7=pyhd8ed1ab_0
  - cffi=1.15.1=py39h2bbff1b_0
  - chardet=4.0.0=py39haa95532_1003
  - charset-normalizer=2.0.4=pyhd3eb1b0_0
  - cookiecutter=1.7.3=pyhd3eb1b0_0
  - cryptography=38.0.1=py39h21b164f_0
  - cuda=11.7.1=0
  - cuda-cccl=11.7.91=0
  - cuda-command-line-tools=11.7.1=0
  - cuda-compiler=11.7.1=0
  - cuda-cudart=11.7.99=0
  - cuda-cudart-dev=11.7.99=0
  - cuda-cuobjdump=11.7.91=0
  - cuda-cupti=11.7.101=0
  - cuda-cuxxfilt=11.7.91=0
  - cuda-demo-suite=11.8.86=0
  - cuda-documentation=11.8.86=0
  - cuda-libraries=11.7.1=0
  - cuda-libraries-dev=11.7.1=0
  - cuda-memcheck=11.8.86=0
```

```
  - cuda-nsight-compute=11.8.0=0
  - cuda-nvcc=11.7.99=0
  - cuda-nvdisasm=11.8.86=0
  - cuda-nvml-dev=11.7.91=0
  - cuda-nvprof=11.8.87=0
  - cuda-nvprune=11.7.91=0
  - cuda-nvrtc=11.7.99=0
  - cuda-nvrtc-dev=11.7.99=0
  - cuda-nvtx=11.7.91=0
  - cuda-nvvp=11.8.87=0
  - cuda-runtime=11.7.1=0
  - cuda-sanitizer-api=11.8.86=0
  - cuda-toolkit=11.7.1=0
  - cuda-tools=11.7.1=0
  - cuda-visual-tools=11.7.1=0
  - cudatoolkit=11.3.1=h59b6b97_2
  - debugpy=1.5.1=py39hd77b12b_0
  - decorator=5.1.1=pyhd3eb1b0_0
  - defusedxml=0.7.1=pyhd3eb1b0_0
  - diff-match-patch=20200713=pyhd3eb1b0_0
  - dill=0.3.6=py39haa95532_0
  - docutils=0.18.1=py39haa95532_3
  - entrypoints=0.4=py39haa95532_0
  - flake8=4.0.1=pyhd3eb1b0_1
  - freetype=2.12.1=ha860e81_0
  - glib=2.69.1=h5dc1a3c_2
  - gst-plugins-base=1.18.5=h9e645db_0
  - gstreamer=1.18.5=hd78058f_0
  - icu=58.2=ha925a31_3
  - idna=3.4=py39haa95532_0
  - imagesize=1.4.1=py39haa95532_0
  - importlib_metadata=4.11.3=hd3eb1b0_0
  - inflection=0.5.1=py39haa95532_0
  - intel-openmp=2021.4.0=haa95532_3556
  - intervaltree=3.1.0=pyhd3eb1b0_0
  - ipykernel=6.15.2=py39haa95532_0
  - ipython_genutils=0.2.0=pyhd3eb1b0_1
  - isort=5.9.3=pyhd3eb1b0_0
  - jedi=0.18.1=py39haa95532_1
  - jellyfish=0.9.0=py39h2bbff1b_0
  - jinja2=3.1.2=py39haa95532_0
  - jinja2-time=0.2.0=pyhd3eb1b0_3
  - jpeg=9e=h2bbff1b_0
  - jsonschema=4.16.0=py39haa95532_0
  - jupyter_client=7.4.7=py39haa95532_0
  - jupyter_core=4.11.2=py39haa95532_0
  - jupyterlab_pygments=0.1.2=py_0
  - keyring=23.4.0=py39haa95532_0
```

```
  - lazy-object-proxy=1.6.0=py39h2bbff1b_0
  - lerc=3.0=hd77b12b_0
  - libcublas=11.11.3.6=0
  - libcublas-dev=11.11.3.6=0
  - libcufft=10.9.0.58=0
  - libcufft-dev=10.9.0.58=0
  - libcurand=10.3.0.86=0
  - libcurand-dev=10.3.0.86=0
  - libcusolver=11.4.1.48=0
  - libcusolver-dev=11.4.1.48=0
  - libcusparse=11.7.5.86=0
  - libcusparse-dev=11.7.5.86=0
  - libdeflate=1.8=h2bbff1b_5
  - libffi=3.4.2=hd77b12b_6
  - libiconv=1.16=h2bbff1b_2
  - libnpp=11.8.0.86=0
  - libnpp-dev=11.8.0.86=0
  - libnvjpeg=11.9.0.86=0
  - libnvjpeg-dev=11.9.0.86=0
  - libogg=1.3.5=h2bbff1b_1
  - libpng=1.6.37=h2a8f88b_0
  - libprotobuf=3.20.1=h23ce68f_0
  - libsodium=1.0.18=h62dcd97_0
  - libspatialindex=1.9.3=h6c2663c_0
  - libtiff=4.4.0=h8a3f274_2
  - libuv=1.40.0=he774522_0
  - libvorbis=1.3.7=he774522_0
  - libwebp=1.2.4=h2bbff1b_0
  - libwebp-base=1.2.4=h2bbff1b_0
  - libxml2=2.9.14=h0ad7f3c_0
  - libxslt=1.1.35=h2bbff1b_0
  - lxml=4.9.1=py39h1985fb9_0
  - lz4-c=1.9.3=h2bbff1b_1
  - markupsafe=2.1.1=py39h2bbff1b_0
  - matplotlib-inline=0.1.6=py39haa95532_0
  - mccabe=0.7.0=pyhd3eb1b0_0
  - mistune=0.8.4=py39h2bbff1b_1000
  - mkl=2021.4.0=haa95532_640
  - mkl-service=2.4.0=py39h2bbff1b_0
  - mkl_fft=1.3.1=py39h277e83a_0
  - mkl_random=1.2.2=py39hf11a4ad_0
  - mypy_extensions=0.4.3=py39haa95532_1
  - nbclient=0.5.13=py39haa95532_0
  - nbconvert=6.5.4=py39haa95532_0
  - nbformat=5.7.0=py39haa95532_0
  - nest-asyncio=1.5.5=py39haa95532_0
  - nsight-compute=2022.3.0.22=0
  - numpy=1.23.3=py39h3b20f71_0
```

```
  - numpy-base=1.23.3=py39h4da318b_0
  - numpydoc=1.5.0=py39haa95532_0
  - onnx=1.12.0=py39h1f835cd_0
  - openssl=1.1.1s=h2bbff1b_0
  - packaging=21.3=pyhd3eb1b0_0
  - pandocfilters=1.5.0=pyhd3eb1b0_0
  - paramiko=2.8.1=pyhd3eb1b0_0
  - parso=0.8.3=pyhd3eb1b0_0
  - pcre=8.45=hd77b12b_0
  - pexpect=4.8.0=pyhd3eb1b0_3
  - pickleshare=0.7.5=pyhd3eb1b0_1003
  - pillow=9.2.0=py39hdc2b20a_1
  - pip=22.2.2=py39haa95532_0
  - pluggy=1.0.0=py39haa95532_1
  - ply=3.11=py39haa95532_0
  - poyo=0.5.0=pyhd3eb1b0_0
  - ptyprocess=0.7.0=pyhd3eb1b0_2
  - pycodestyle=2.8.0=pyhd3eb1b0_0
  - pycparser=2.21=pyhd3eb1b0_0
  - pydocstyle=6.1.1=pyhd3eb1b0_0
  - pyflakes=2.4.0=pyhd3eb1b0_0
  - pylint=2.14.5=py39haa95532_0
  - pyls-spyder=0.4.0=pyhd3eb1b0_0
  - pynacl=1.5.0=py39h8cc25b3_0
  - pyopenssl=22.0.0=pyhd3eb1b0_0
  - pyparsing=3.0.9=py39haa95532_0
  - pyqt=5.15.7=py39hd77b12b_0
  - pyqt5-sip=12.11.0=py39hd77b12b_0
  - pyqtwebengine=5.15.7=py39hd77b12b_0
  - pyrsistent=0.18.0=py39h196d8e1_0
  - pysocks=1.7.1=py39haa95532_0
  - python=3.9.15=h6244533_0
  - python-dateutil=2.8.2=pyhd3eb1b0_0
  - python-fastjsonschema=2.16.2=py39haa95532_0
  - python-lsp-black=1.2.1=py39haa95532_0
  - python-lsp-jsonrpc=1.0.0=pyhd3eb1b0_0
  - python-lsp-server=1.5.0=py39haa95532_0
  - python-slugify=5.0.2=pyhd3eb1b0_0
  - pytorch=1.11.0=py3.9_cuda11.3_cudnn8_0
  - pytorch-cuda=11.7=h67b0de4_0
  - pytorch-mutex=1.0=cuda
  - pywin32-ctypes=0.2.0=py39haa95532_1000
  - pyzmq=23.2.0=py39hd77b12b_0
  - qdarkstyle=3.0.2=pyhd3eb1b0_0
  - qstylizer=0.1.10=pyhd3eb1b0_0
  - qt-main=5.15.2=he8e5bd7_7
  - qt-webengine=5.15.9=hb9a9bb5_4
  - qtawesome=1.0.3=pyhd3eb1b0_0
```

```
- qtconsole=5.3.2=py39haa95532_0
- qtpy=2.2.0=py39haa95532_0
- qtwebkit=5.212=h3ad3cdb_4
- requests=2.28.1=py39haa95532_0
- rope=0.22.0=pyhd3eb1b0_0
- rtree=0.9.7=py39h2eaa2aa_1
- setuptools=65.5.0=py39haa95532_0
- sip=6.6.2=py39hd77b12b_0
- six=1.16.0=pyhd3eb1b0_1
- snowballstemmer=2.2.0=pyhd3eb1b0_0
- sortedcontainers=2.4.0=pyhd3eb1b0_0
- soupsieve=2.3.2.post1=py39haa95532_0
- sphinx=5.0.2=py39haa95532_0
- sphinxcontrib-applehelp=1.0.2=pyhd3eb1b0_0
- sphinxcontrib-devhelp=1.0.2=pyhd3eb1b0_0
- sphinxcontrib-htmlhelp=2.0.0=pyhd3eb1b0_0
- sphinxcontrib-jsmath=1.0.1=pyhd3eb1b0_0
- sphinxcontrib-qthelp=1.0.3=pyhd3eb1b0_0
- sphinxcontrib-serializinghtml=1.1.5=pyhd3eb1b0_0
- spyder=5.3.3=py39haa95532_0
- spyder-kernels=2.3.3=py39haa95532_0
- sqlite=3.39.3=h2bbff1b_0
- text-unidecode=1.3=pyhd3eb1b0_0
- textdistance=4.2.1=pyhd3eb1b0_0
- three-merge=0.1.1=pyhd3eb1b0_0
- tinycss=0.4=pyhd3eb1b0_1002
- tinycss2=1.2.1=py39haa95532_0
- tk=8.6.12=h2bbff1b_0
- toml=0.10.2=pyhd3eb1b0_0
- tomli=2.0.1=py39haa95532_0
- tomlkit=0.11.1=py39haa95532_0
- torchaudio=0.11.0=py39_cu113
- torchvision=0.12.0=py39_cu113
- tornado=6.2=py39h2bbff1b_0
- typing-extensions=4.3.0=py39haa95532_0
- typing_extensions=4.3.0=py39haa95532_0
- tzdata=2022f=h04d1e81_0
- ujson=5.4.0=py39hd77b12b_0
- unidecode=1.2.0=pyhd3eb1b0_0
- urllib3=1.26.12=py39haa95532_0
- vc=14.2=h21ff451_1
- vs2015_runtime=14.27.29016=h5e58377_2
- watchdog=2.1.6=py39haa95532_0
- wcwidth=0.2.5=pyhd3eb1b0_0
- webencodings=0.5.1=py39haa95532_1
- whatthepatch=1.0.2=py39haa95532_0
- wheel=0.37.1=pyhd3eb1b0_0
- win_inet_pton=1.1.0=py39haa95532_0
```

```
- wincertstore=0.2=py39haa95532_2
- wrapt=1.14.1=py39h2bbff1b_0
- xz=5.2.6=h8cc25b3_0
- yaml=0.2.5=he774522_0
- yapf=0.31.0=pyhd3eb1b0_0
- zeromq=4.3.4=hd77b12b_0
- zlib=1.2.13=h8cc25b3_0
- zstd=1.5.2=h19a0ad4_0
- pip:
  - absl-py==1.3.0
  - altgraph==0.17.3
  - antlr4-python3-runtime==4.9.3
  - asttokens==2.1.0
  - astunparse==1.6.3
  - auto-py-to-exe==2.27.0
  - black==22.3.0
  - bottle==0.12.23
  - bottle-websocket==0.2.9
  - cachetools==5.2.0
  - click==8.1.3
  - cloudpickle==2.2.0
  - colorama==0.4.6
  - contourpy==1.0.6
  - cycler==0.11.0
  - cython==0.29.32
  - eel==0.14.0
  - executing==1.2.0
  - fairscale==0.4.12
  - filelock==3.8.0
  - flatbuffers==23.1.4
  - fonttools==4.38.0
  - future==0.18.2
  - fvcore==0.1.5.post20220512
  - gast==0.4.0
  - gevent==22.10.2
  - gevent-websocket==0.10.1
  - google-auth==2.14.1
  - google-auth-oauthlib==0.4.6
  - google-pasta==0.2.0
  - greenlet==2.0.1
  - grpcio==1.50.0
  - h5py==3.7.0
  - huggingface-hub==0.10.1
  - hydra-core==1.2.0
  - importlib-metadata==5.0.0
  - iopath==0.1.9
  - ipython==8.6.0
  - keras==2.11.0
```

```
  - kiwisolver==1.4.4
  - libclang==15.0.6.1
  - markdown==3.4.1
  - matplotlib==3.6.2
  - mypy-extensions==0.4.3
  - oauthlib==3.2.2
  - omegaconf==2.2.3
  - opencv-python==4.6.0.66
  - opt-einsum==3.3.0
  - pandas==1.5.1
  - pathspec==0.10.2
  - pefile==2022.5.30
  - platformdirs==2.5.4
  - portalocker==2.6.0
  - prompt-toolkit==3.0.32
  - protobuf==3.19.6
  - psutil==5.9.4
  - pure-eval==0.2.2
  - pyasn1==0.4.8
  - pyasn1-modules==0.2.8
  - pycocotools==2.0.6
  - pydot==1.4.2
  - pygments==2.13.0
  - pyinstaller==5.7.0
  - pyinstaller-hooks-contrib==2022.15
  - pytz==2022.6
  - pywin32==305
  - pyyaml==5.1
  - requests-oauthlib==1.3.1
  - rsa==4.9
  - scipy==1.9.3
  - seaborn==0.12.1
  - stack-data==0.6.1
  - tabulate==0.9.0
  - tensorboard==2.11.0
  - tensorboard-data-server==0.6.1
  - tensorboard-plugin-wit==1.8.1
  - tensorflow==2.11.0
  - tensorflow-estimator==2.11.0
  - tensorflow-intel==2.11.0
  - tensorflow-io-gcs-filesystem==0.29.0
  - termcolor==2.1.0
  - thop==0.1.1-2209072238
  - timm==0.6.11
  - tqdm==4.64.1
  - traitlets==5.5.0
  - werkzeug==2.2.2
  - whichcraft==0.6.1
```

```
    - yacs==0.1.8
    - zipp==3.10.0
    - zope-event==4.6
    - zope-interface==5.5.2
```

Mask-rcnn enviroment

```
name: mask-rcnn
channels:
  - defaults
dependencies:
  - ca-certificates=2022.10.11=haa95532_0
  - certifi=2022.9.24=py38haa95532_0
  - libffi=3.4.2=hd77b12b_6
  - openssl=1.1.1s=h2bbff1b_0
  - pip=22.3.1=py38haa95532_0
  - python=3.8.15=h6244533_2
  - setuptools=65.5.0=py38haa95532_0
  - sqlite=3.40.0=h2bbff1b_0
  - vc=14.2=h21ff451_1
  - vs2015_runtime=14.27.29016=h5e58377_2
  - wheel=0.37.1=pyhd3eb1b0_0
  - wincertstore=0.2=py38haa95532_2
  - pip:
    - absl-py==1.3.0
    - alabaster==0.7.12
    - anyio==3.6.2
    - argon2-cffi==21.3.0
    - argon2-cffi-bindings==21.2.0
    - arrow==1.2.3
    - astunparse==1.6.3
    - attrs==22.1.0
    - babel==2.11.0
    - backcall==0.2.0
    - beautifulsoup4==4.11.1
    - bleach==5.0.1
    - cachetools==5.2.0
    - cffi==1.15.1
    - charset-normalizer==2.1.1
    - colorama==0.4.6
    - comm==0.1.2
    - cycler==0.11.0
    - cython==0.29.32
    - debugpy==1.6.4
    - decorator==5.1.1
    - defusedxml==0.7.1
    - docutils==0.19
    - entrypoints==0.4
```

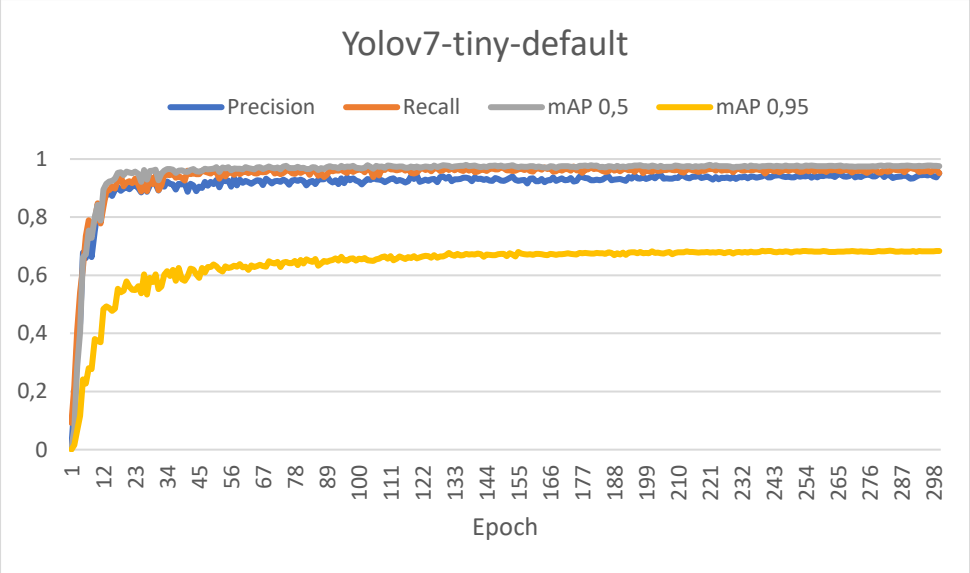
```
  - fastjsonschema==2.16.2
  - fonttools==4.38.0
  - fqdn==1.5.1
  - gast==0.3.3
  - google-auth==2.15.0
  - google-auth-oauthlib==0.4.6
  - google-pasta==0.2.0
  - grpcio==1.51.1
  - h5py==2.10.0
  - idna==3.4
  - imageio==2.22.4
  - imagesize==1.4.1
  - imgaug==0.4.0
  - importlib-metadata==5.1.0
  - importlib-resources==5.10.1
  - ipykernel==6.19.2
  - ipyparallel==8.4.1
  - ipython==7.34.0
  - ipython-genutils==0.2.0
  - ipywidgets==8.0.3
  - isoduration==20.11.0
  - jedi==0.18.2
  - jinja2==3.1.2
  - jsonpointer==2.3
  - jsonschema==4.17.3
  - jupyter-client==7.4.8
  - jupyter-core==5.1.0
  - jupyter-events==0.5.0
  - jupyter-server==2.0.1
  - jupyter-server-terminals==0.4.2
  - jupyterlab-pygments==0.2.2
  - jupyterlab-widgets==3.0.4
  - keras==2.4.3
  - keras-preprocessing==1.1.2
  - kiwisolver==1.4.4
  - markdown==3.4.1
  - markupsafe==2.1.1
  - matplotlib==3.5.3
  - matplotlib-inline==0.1.6
  - mistune==2.0.4
  - nbclassic==0.4.8
  - nbclient==0.7.2
  - nbconvert==7.2.6
  - nbformat==5.7.0
  - nest-asyncio==1.5.6
  - networkx==2.8.8
  - nose==1.3.7
  - notebook==6.5.2
```

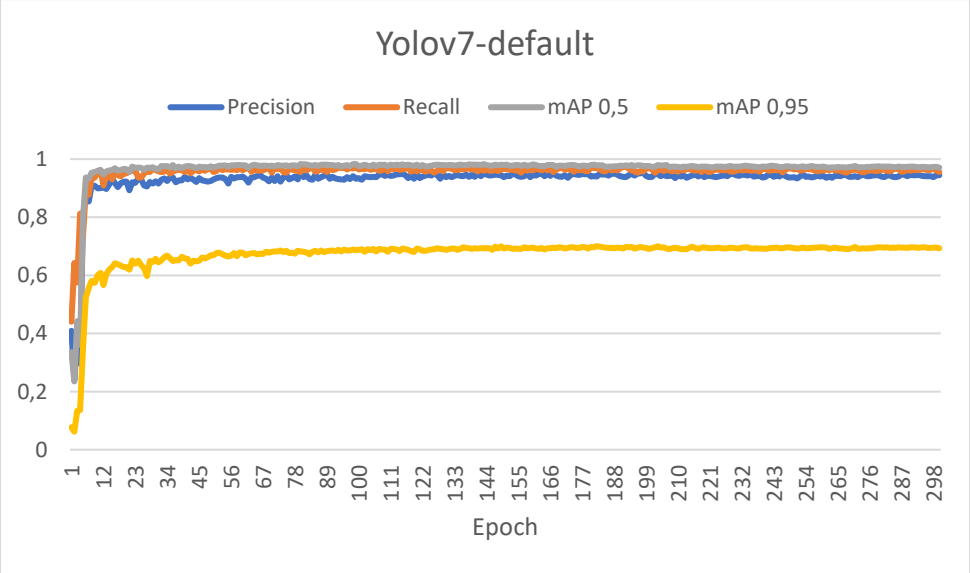
```
  - notebook-shim==0.2.2
  - numpy==1.18.5
  - oauthlib==3.2.2
  - opencv-python==4.6.0.66
  - opt-einsum==3.3.0
  - packaging==22.0
  - pandocfilters==1.5.0
  - parso==0.8.3
  - pickleshare==0.7.5
  - pillow==9.3.0
  - pkgutil-resolve-name==1.3.10
  - platformdirs==2.6.0
  - prometheus-client==0.15.0
  - prompt-toolkit==3.0.36
  - protobuf==3.20.3
  - psutil==5.9.4
  - pyasn1==0.4.8
  - pyasn1-modules==0.2.8
  - pycocotools-windows==2.0.0.2
  - pycparser==2.21
  - pygments==2.13.0
  - pyparsing==3.0.9
  - pyrsistent==0.19.2
  - python-dateutil==2.8.2
  - python-json-logger==2.0.4
  - pytz==2022.6
  - pywavelets==1.4.1
  - pywin32==305
  - pywinpty==2.0.9
  - pyyaml==6.0
  - pyzmq==24.0.1
  - qtconsole==5.4.0
  - qtpy==2.3.0
  - requests==2.28.1
  - requests-oauthlib==1.3.1
  - rfc3339-validator==0.1.4
  - rfc3986-validator==0.1.1
  - rsa==4.9
  - scikit-image==0.16.2
  - scipy==1.4.1
  - send2trash==1.8.0
  - shapely==1.8.5.post1
  - six==1.16.0
  - sniffio==1.3.0
  - snowballstemmer==2.2.0
  - soupsieve==2.3.2.post1
  - sphinx==5.3.0
  - sphinxcontrib-applehelp==1.0.2
```

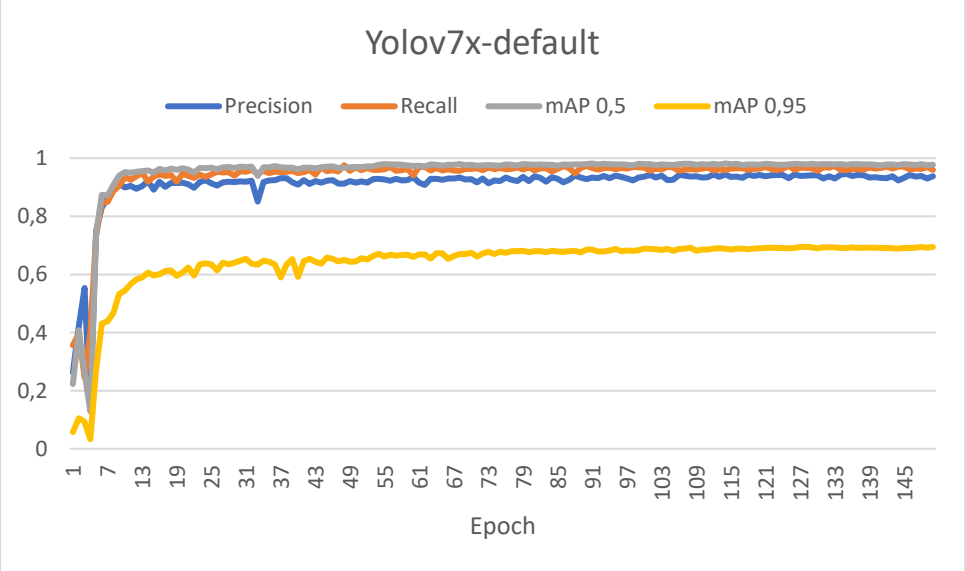
```
  - sphinxcontrib-devhelp==1.0.2
  - sphinxcontrib-htmlhelp==2.0.0
  - sphinxcontrib-jsmath==1.0.1
  - sphinxcontrib-qthelp==1.0.3
  - sphinxcontrib-serializinghtml==1.1.5
  - tensorboard==2.11.2
  - tensorboard-data-server==0.6.1
  - tensorboard-plugin-wit==1.8.1
  - tensorflow==2.3.0
  - tensorflow-estimator==2.3.0
  - termcolor==2.1.1
  - terminado==0.17.1
  - testpath==0.6.0
  - tifffile==2021.11.2
  - tinycss2==1.2.1
  - tornado==6.2
  - tqdm==4.64.1
  - traitlets==5.7.0
  - uri-template==1.2.0
  - urllib3==1.26.13
  - wcwidth==0.2.5
  - webcolors==1.12
  - webencodings==0.5.1
  - websocket-client==1.4.2
  - werkzeug==2.2.2
  - widgetsnbextension==4.0.4
  - wrapt==1.14.1
  - zipp==3.11.0
```

Appendix B: Training curves for every model

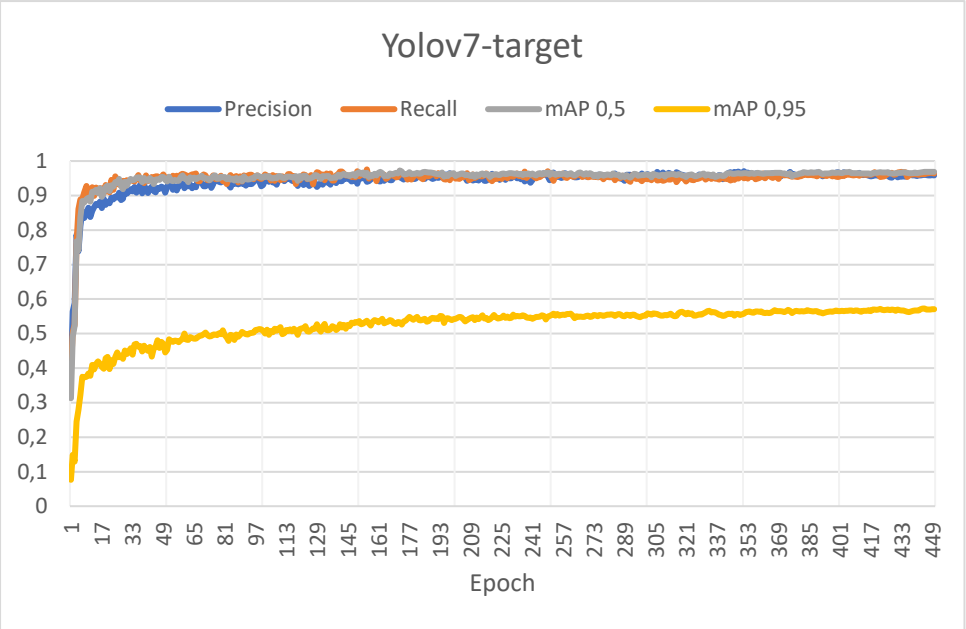
Default

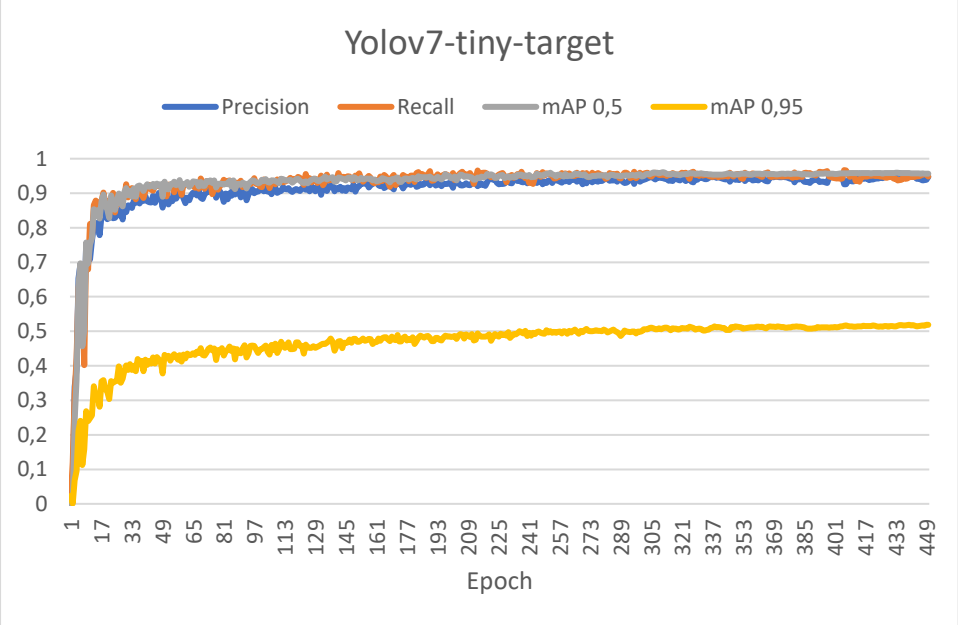

Yolov7-tiny-default
Precision
Recall
mAP 0,5
mAP 0,95
1
0,8
0,6
0,4
0,2
0
1 12 23 34 45 56 67 78 89 100 111 122 133 144 155 166 177 188 199 210 221 232 243 254 265 276 287 298
Epoch

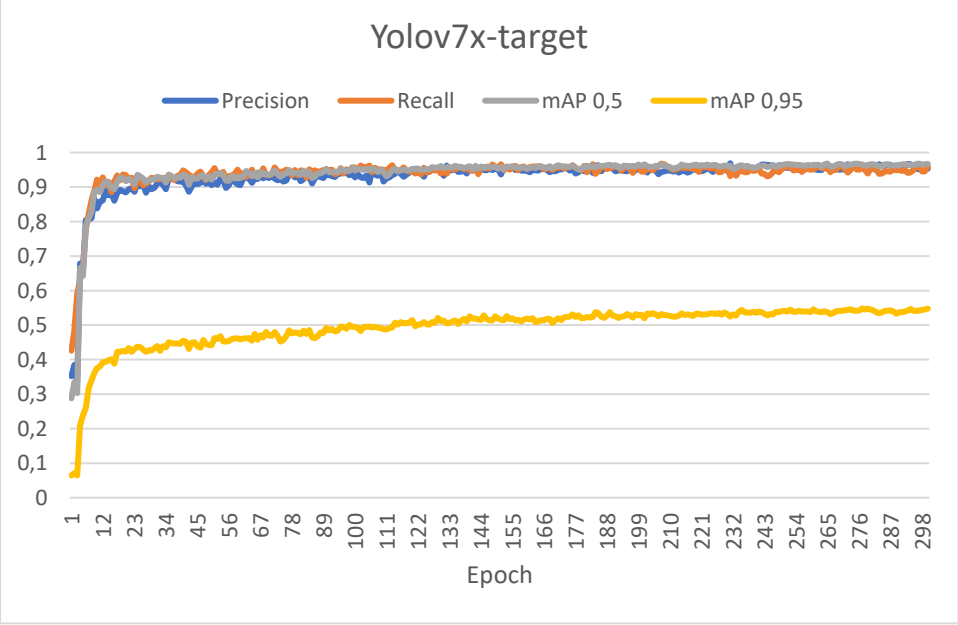

Yolov7-default
Precision
Recall
mAP 0,5
mAP 0,95
1
0,8
0,6
0,4
0,2
0
1 12 23 34 45 56 67 78 89 100 111 122 133 144 155 166 177 188 199 210 221 232 243 254 265 276 287 298
Epoch


Yolov7x-default
Precision
Recall
mAP 0,5
mAP 0,95
1
0,8
0,6
0,4
0,2
0
1 7 13 19 25 31 37 43 49 55 61 67 73 79 85 91 97 103 109 115 121 127 133 139 145
Epoch

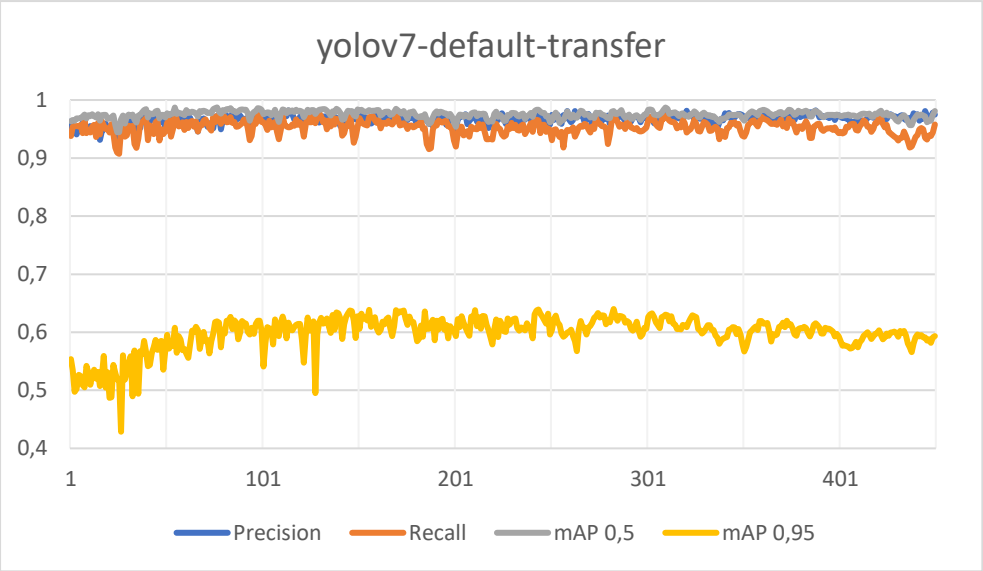
Target

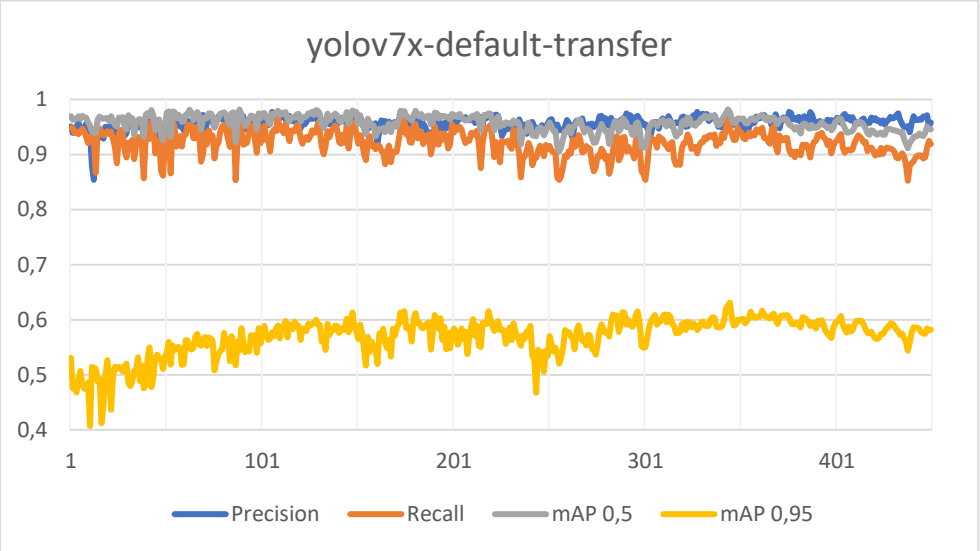

Yolov7-target
Precision
Recall
mAP 0,5
mAP 0,95
1
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
1 17 33 49 65 81 97 113 129 145 161 177 193 209 225 241 257 273 289 305 321 337 353 369 385 401 417 433 449
Epoch

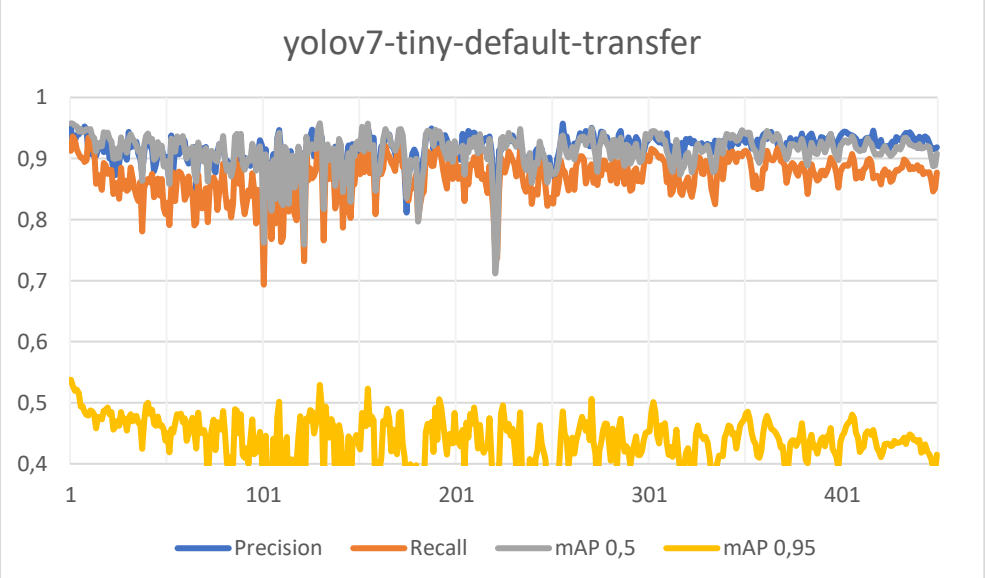

Yolov7-tiny-target
Precision
Recall
mAP 0,5
mAP 0,95
1
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
1 17 33 49 65 81 97 113 129 145 161 177 193 209 225 241 257 273 289 305 321 337 353 369 385 401 417 433 449
Epoch


Yolov7x-target
Precision
Recall
mAP 0,5
mAP 0,95
1
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
1 12 23 34 45 56 67 78 89 100 111 122 133 144 155 166 177 188 199 210 221 232 243 254 265 276 287 298
Epoch

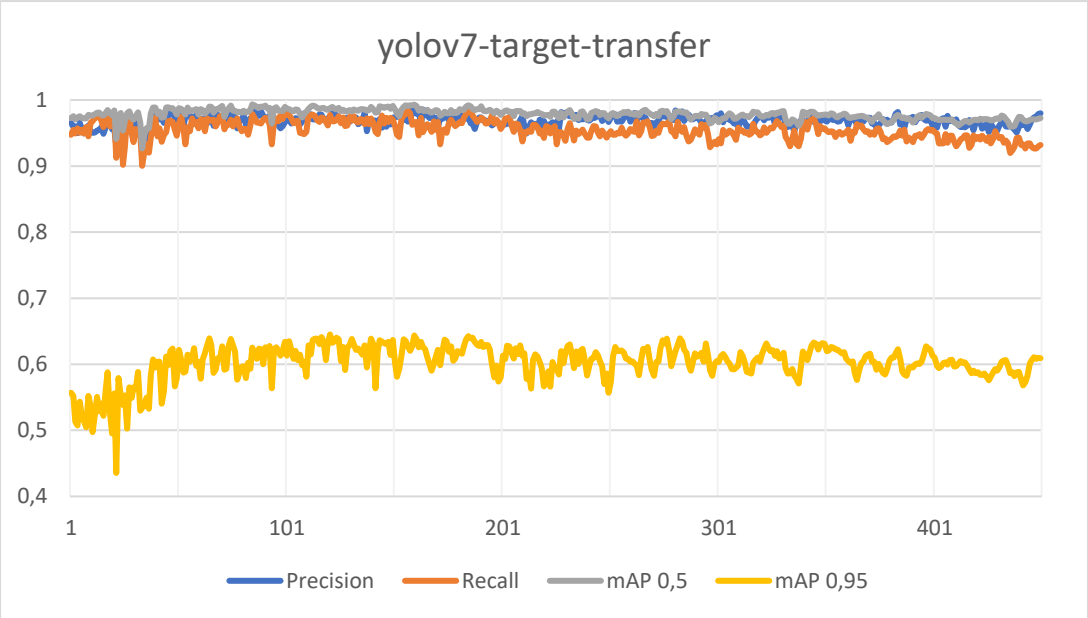
Default transfer

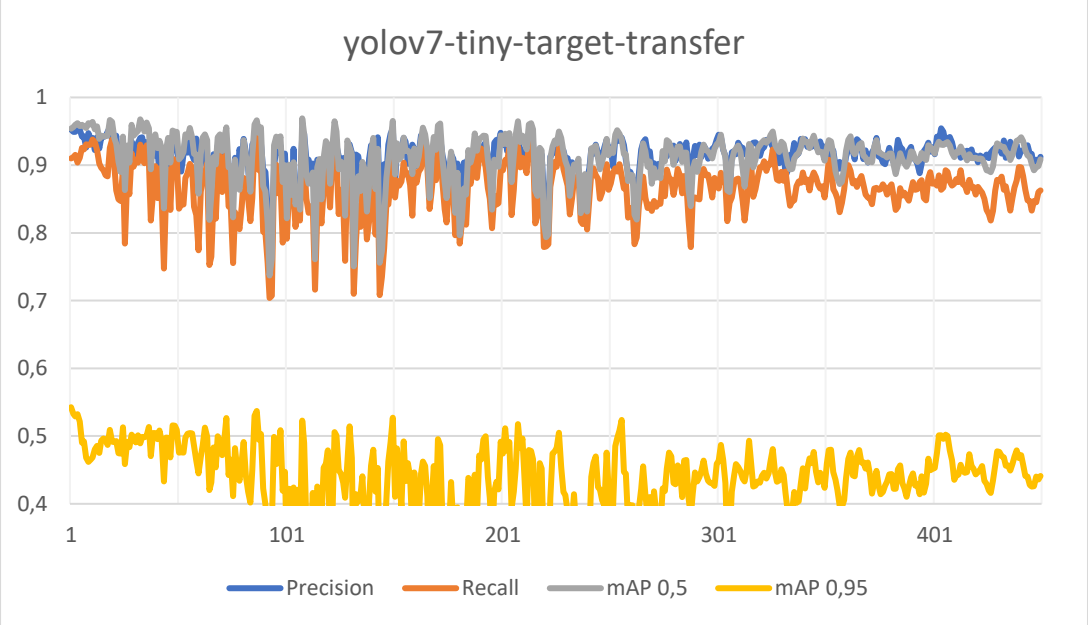


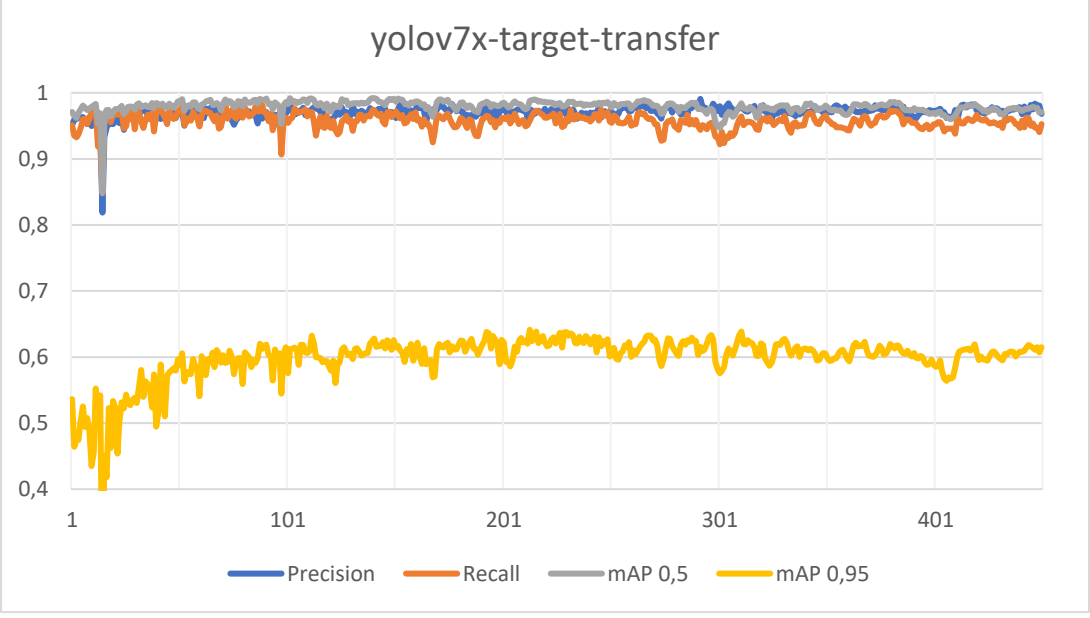




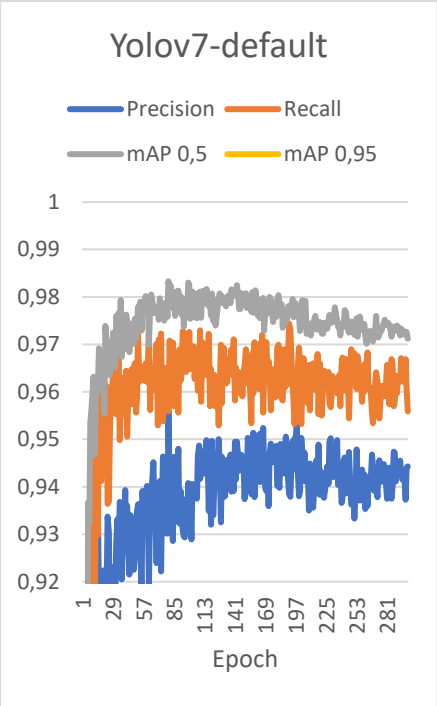
Target-transfer

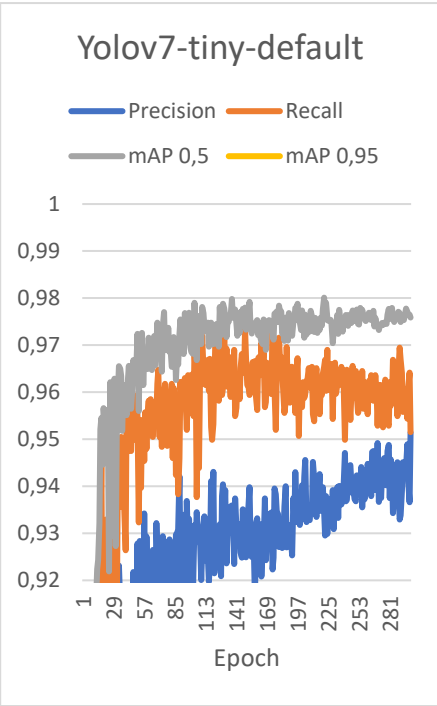


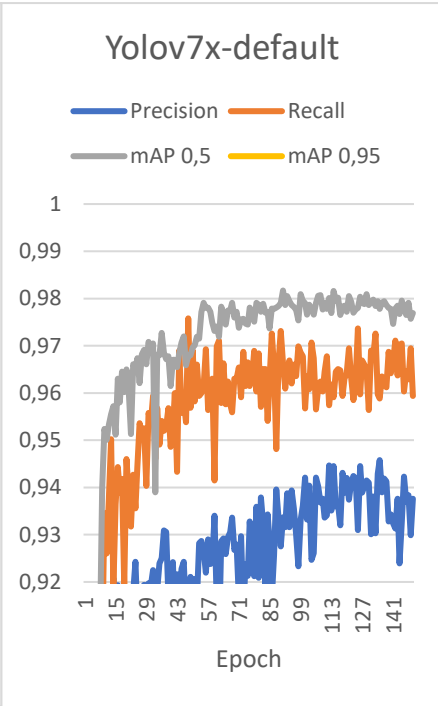




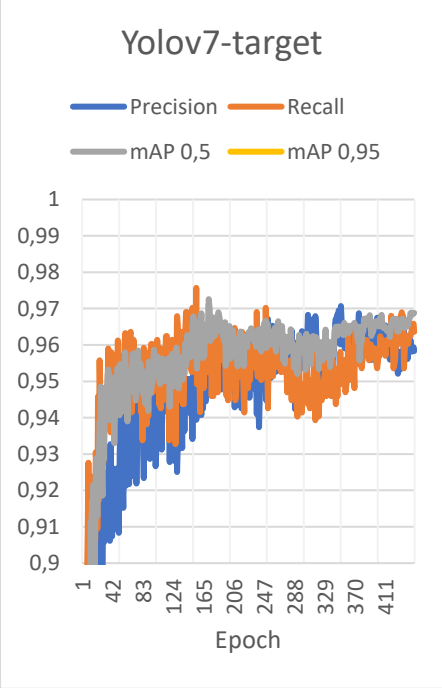
Default

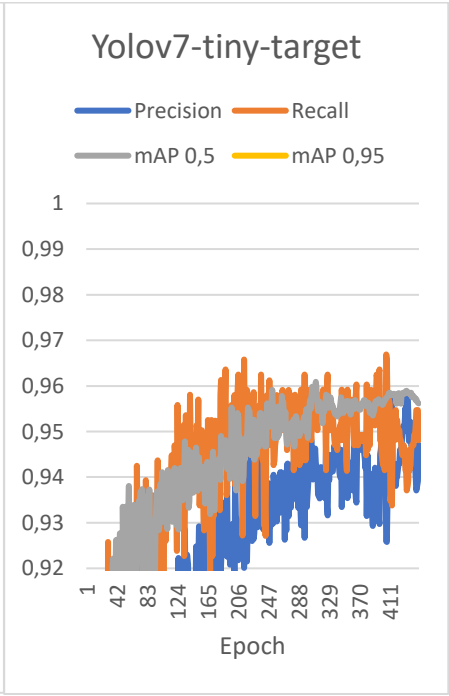


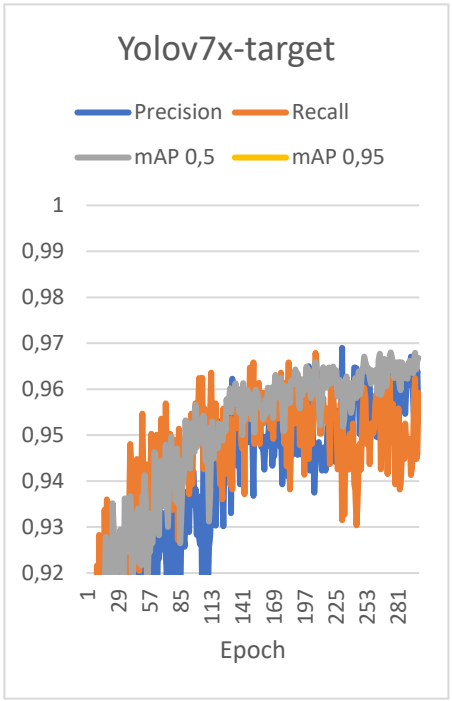




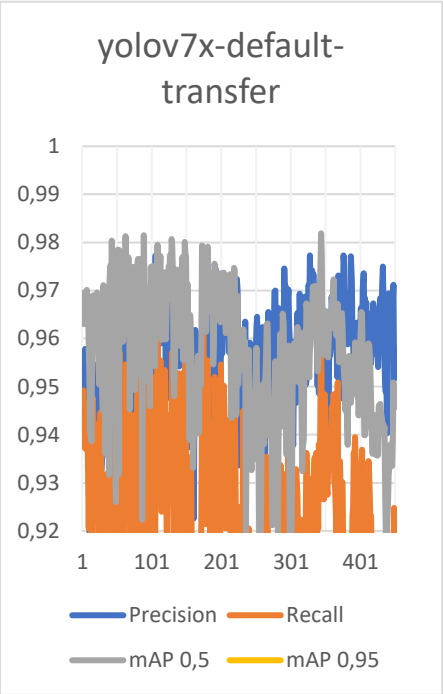
Target

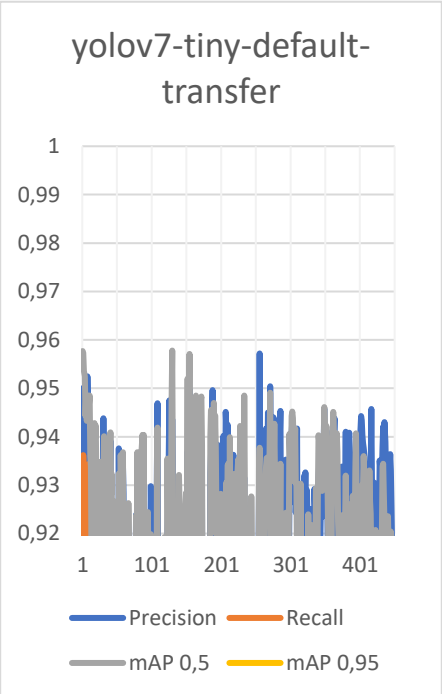


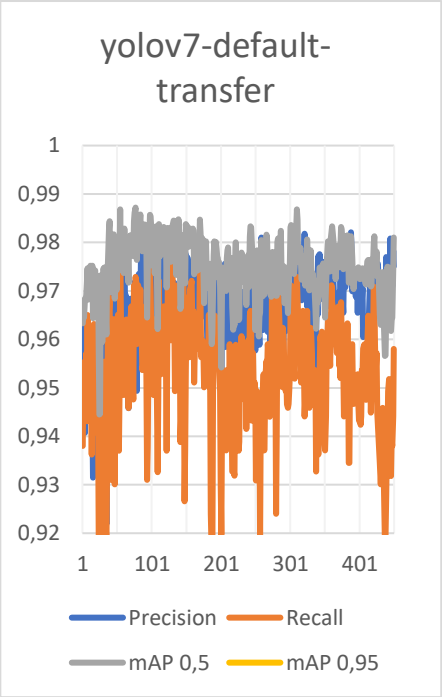




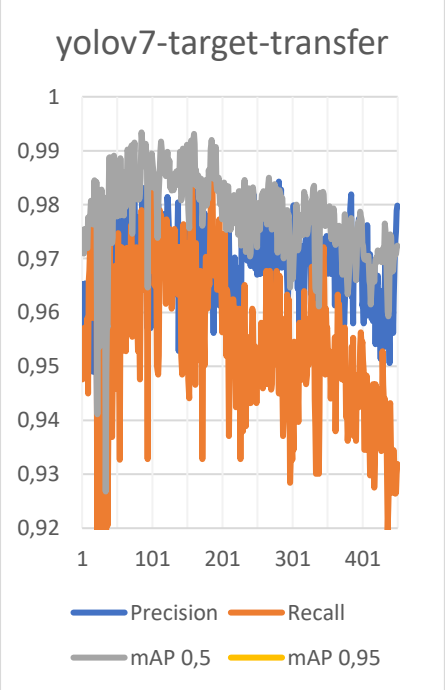
Default – transfer

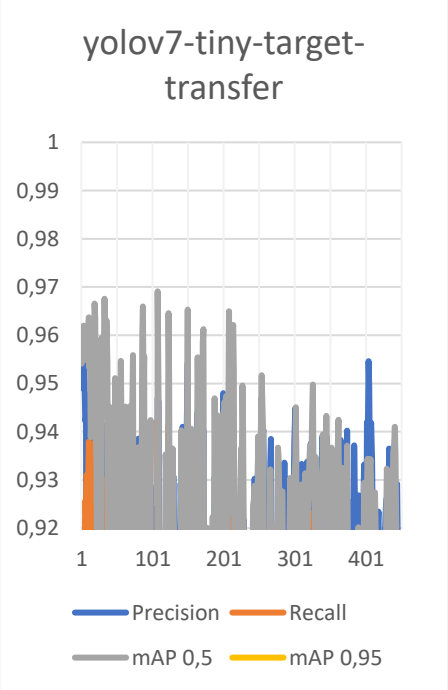


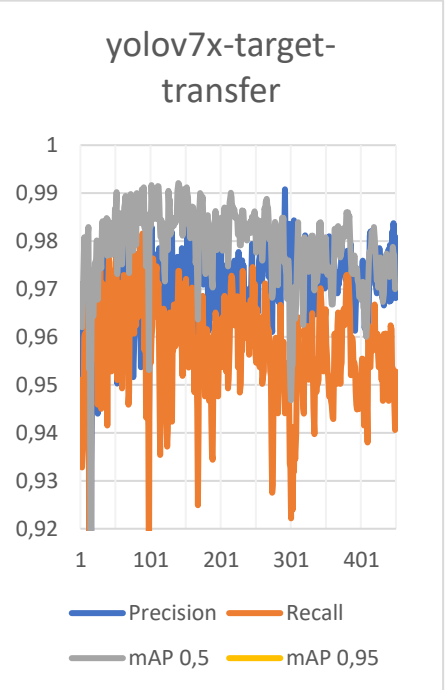




Target-transfer







Appendix C: Code for improving precision with stacks

```
def overlaps_withxyxy(box1, box2):
   # Extract the coordinates of the bounding boxes
  x1_1, y1_1, x2_1, y2_1 = box1
  x1_2, y1_2, x2_2, y2_2 = box2

  # Check if the bounding boxes overlap
  return  not (x1_1 > x2_2 or x2_1 < x1_2 or y1_1 > y2_2 or y2_1 < y1_2)


def find_overlapping_boxes(boxes, min_overlap):
  # Create a dictionary to store the overlaps for each bounding box
  overlaps = {}
  # Then, we iterate over all images and all bounding boxes in each image
  for image in boxes:
    for box in image:
      # Convert the bounding box coordinates to a tuple

      box_tuple = tuple(box)
      # Initialize the overlap count for this bounding box
      overlaps[box_tuple] = 0
      # For each bounding box, we check if it overlaps with any other box in
the other images
      for other_image in boxes:
        if other_image != image:
          for other_box in other_image:
            if overlaps_withxyxy(box, other_box):
              # If it does, we increment the overlap count for that box
              overlaps[box_tuple] += 1

  # Finally, we return all bounding boxes that have at least the minimum
number of overlaps
  return [box for box, overlap in overlaps.items() if overlap >= min_overlap]
```

In the main detect method

```
overlapping_boxes = find_overlapping_boxes(totalImg,110)
    for path, img, im0s, vid_cap in dataset:
        totalImg.append([])
        #im0s = torch.from_numpy(img).to(device)
        #im0s /= 255.0  # 0 - 255 to 0.0 - 1.0
        for box in overlapping_boxes:
            #box_list = list(box)
            x1, y1, x2, y2 = box
            #box_list[0] = box_list[0]*im0s.shape[0]
            #box_list[1] = box_list[1]*im0s.shape[1]
            #box_list[2] = box_list[2]*im0s.shape[1]
```

```
            #box_list[3]=  box_list[3]*im0s.shape[0]
            #print(box_list)
            #x,y,w,h = box_list

            #plot_one_box(box_list, im0s, label='stomata_overlap',
color=colors[int(cls)], line_thickness=1)
            cv2.rectangle(im0s, (x1, y1), (x2, y2), (0, 255, 0), 2)
        im0s = cv2.resize(im0s, (1330, 1150))
        cv2.imshow('Overlapping Boxes', im0s, )

        # To wait for a key press before closing the window, use the
cv2.waitKey function
        cv2.waitKey(0)
        return
```

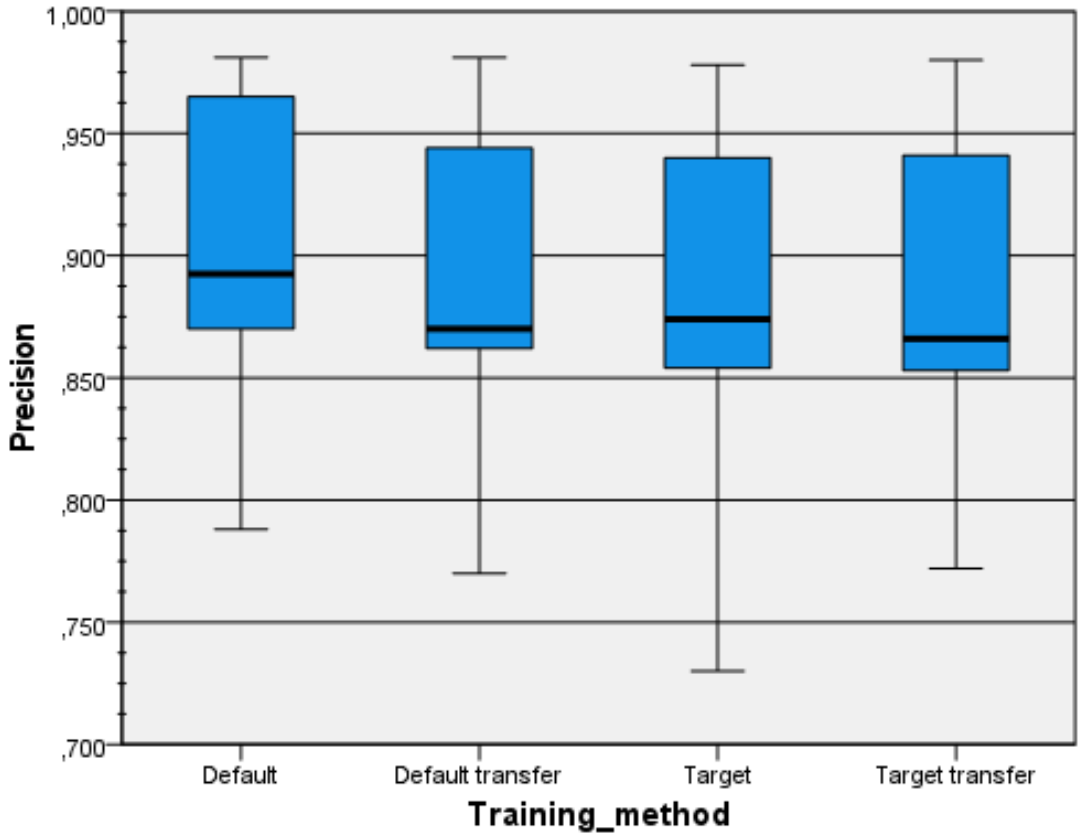
Appendix D: All test results

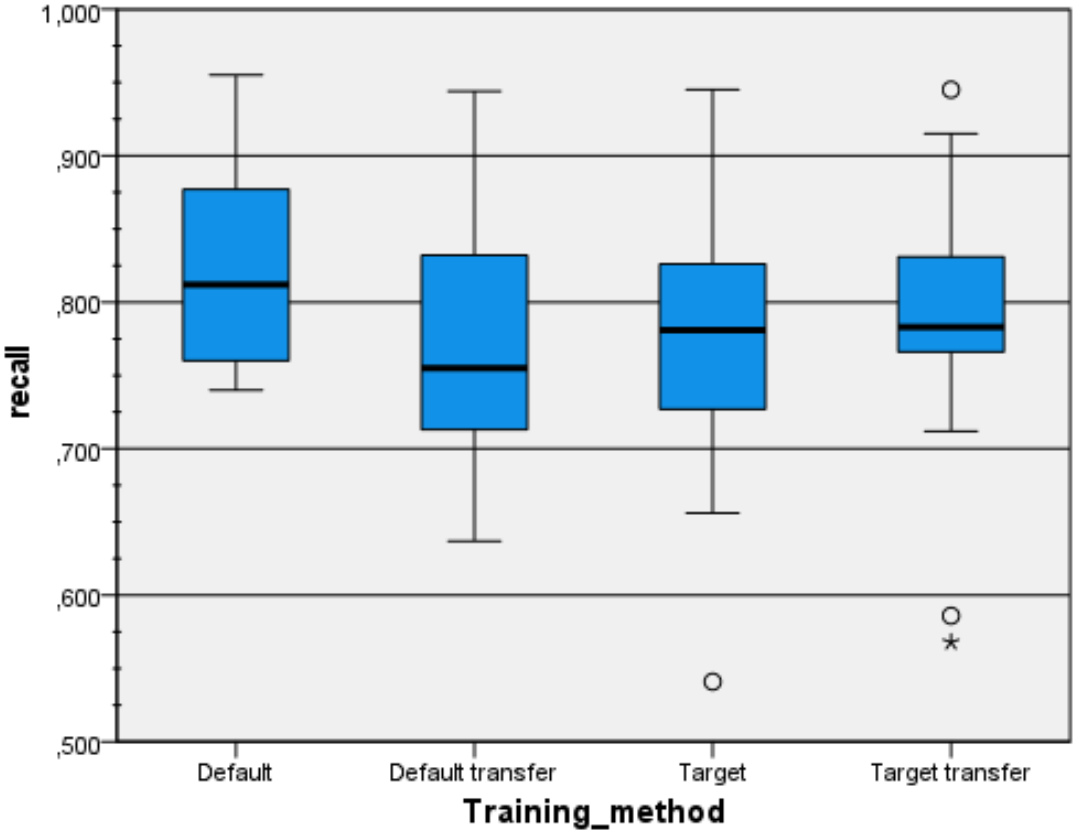
Model	Training_method	TestData	Precision	recall	F1	mAP0.5	mAP0.95
yolov7	default	All	0,883	0,799	0,84	0,858	0,399
yolov7-tiny	default	All	0,884	0,803	0,84	0,874	0,403
yolov7x	default	All	0,889	0,751	0,81	0,837	0,394
yolov7	target	All	0,868	0,713	0,78	0,791	0,361
yolov7-tiny	target	All	0,862	0,790	0,82	0,837	0,378
yolov7x	target	All	0,904	0,720	0,80	0,818	0,377
yolov7	default-transfer	All	0,875	0,727	0,79	0,801	0,375
yolov7-tiny	default-transfer	All	0,879	0,810	0,84	0,882	0,415
yolov7x	default-transfer	All	0,859	0,707	0,78	0,770	0,348
yolov7	target-transfer	All	0,864	0,775	0,82	0,822	0,366
yolov7-tiny	target-transfer	All	0,853	0,806	0,83	0,847	0,397
yolov7x	target-transfer	All	0,874	0,758	0,81	0,830	0,377
yolov7	default	no_sl	0,973	0,874	0,92	0,927	0,479
yolov7-tiny	default	no_sl	0,941	0,877	0,91	0,921	0,474
yolov7x	default	no_sl	0,965	0,911	0,94	0,947	0,501
yolov7	target	no_sl	0,944	0,818	0,88	0,893	0,463
yolov7-tiny	target	no_sl	0,924	0,858	0,89	0,900	0,467
yolov7x	target	no_sl	0,973	0,832	0,90	0,910	0,471
yolov7	default-transfer	no_sl	0,940	0,780	0,85	0,860	0,455
yolov7-tiny	default-transfer	no_sl	0,944	0,878	0,91	0,932	0,482
yolov7x	default-transfer	no_sl	0,907	0,766	0,83	0,848	0,437
yolov7	target-transfer	no_sl	0,962	0,769	0,85	0,834	0,439

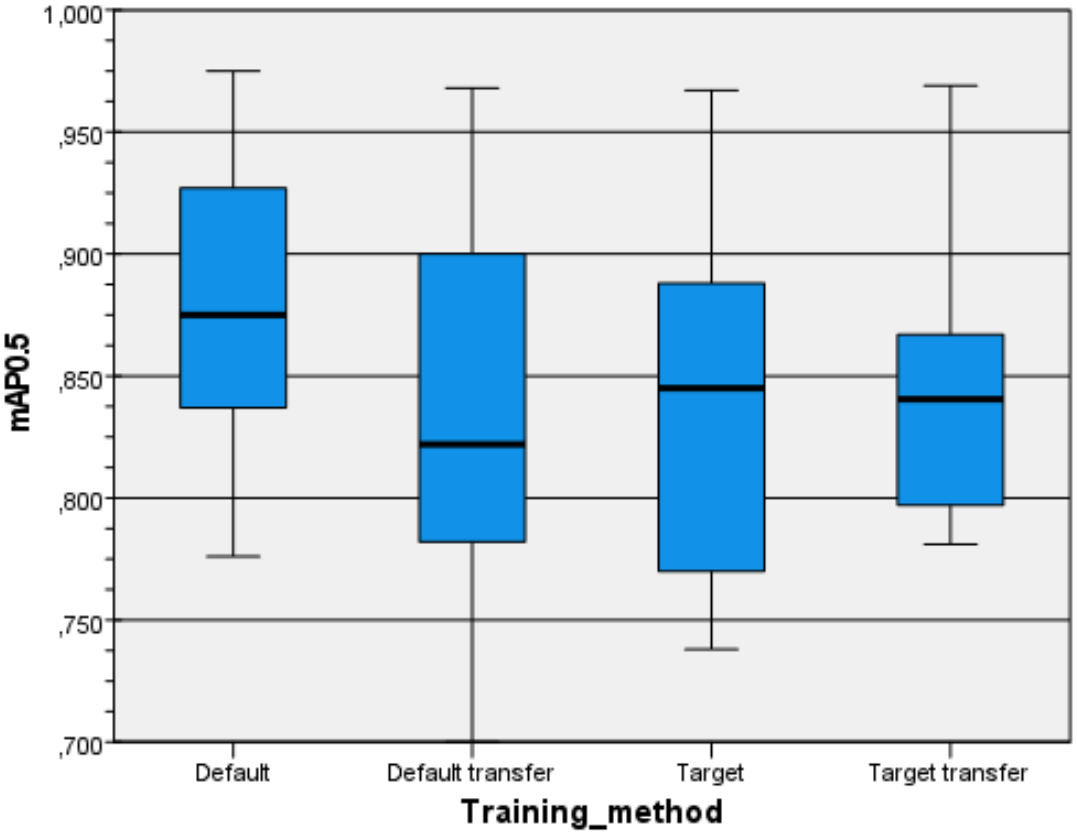
yolov7-tiny	target-transfer	no_sl	0,941	0,852	0,89	0,906	0,476
yolov7x	target-transfer	no_sl	0,907	0,766	0,83	0,848	0,437
yolov7	default	no_kom	0,876	0,798	0,84	0,859	0,392
yolov7-tiny	default	no_kom	0,870	0,825	0,85	0,885	0,401
yolov7x	default	no_kom	0,868	0,787	0,83	0,858	0,392
yolov7	default	only_ch	0,975	0,940	0,96	0,952	0,497
yolov7-tiny	default	only_ch	0,969	0,950	0,96	0,961	0,499
yolov7x	default	only_ch	0,981	0,955	0,97	0,975	0,523
yolov7	default	only_kom	0,905	0,821	0,86	0,876	0,453
yolov7-tiny	default	only_kom	0,896	0,748	0,82	0,833	0,425
yolov7x	default	only_kom	0,938	0,840	0,89	0,895	0,462
yolov7	default	only_sl	0,802	0,743	0,77	0,790	0,321
yolov7-tiny	default	only_sl	0,816	0,760	0,79	0,825	0,330
yolov7x	default	only_sl	0,788	0,740	0,76	0,776	0,298
yolov7	target	no_kom	0,870	0,736	0,80	0,804	0,357
yolov7-tiny	target	no_kom	0,864	0,804	0,83	0,858	0,380
yolov7x	target	no_kom	0,852	0,765	0,81	0,826	0,371
yolov7	target	only_ch	0,973	0,926	0,95	0,961	0,497
yolov7-tiny	target	only_ch	0,979	0,937	0,96	0,965	0,506
yolov7x	target	only_ch	0,981	0,944	0,96	0,968	0,501
yolov7	target	only_kom	0,870	0,646	0,74	0,748	0,404
yolov7-tiny	target	only_kom	0,862	0,712	0,78	0,771	0,390
yolov7x	target	only_kom	0,920	0,690	0,79	0,799	0,428
yolov7	target	only_sl	0,805	0,637	0,71	0,698	0,264
yolov7-tiny	target	only_sl	0,796	0,745	0,77	0,782	0,294
yolov7x	target	only_sl	0,770	0,713	0,74	0,737	0,286
yolov7	default-transfer	no_kom	0,854	0,807	0,83	0,842	0,385
yolov7-tiny	default-transfer	no_kom	0,873	0,826	0,85	0,888	0,409
yolov7x	default-transfer	no_kom	0,875	0,782	0,83	0,824	0,375
yolov7	default-transfer	only_ch	0,968	0,909	0,94	0,946	0,506
yolov7-tiny	default-transfer	only_ch	0,978	0,945	0,96	0,967	0,498
yolov7x	default-transfer	only_ch	0,955	0,892	0,92	0,930	0,480
yolov7	default-transfer	only_kom	0,865	0,656	0,75	0,738	0,381
yolov7-tiny	default-transfer	only_kom	0,857	0,792	0,82	0,854	0,457
yolov7x	default-transfer	only_kom	0,730	0,541	0,62	0,580	0,237

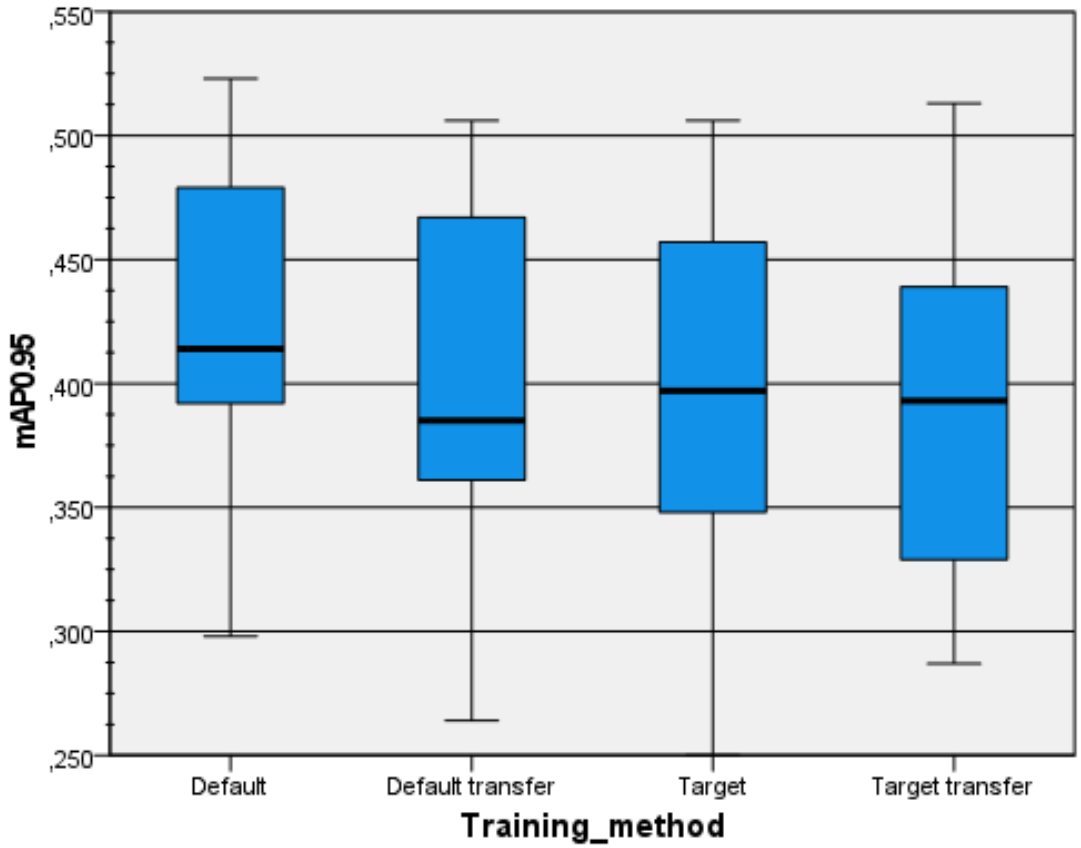
yolov7	default-transfer	only_sl	0,795	0,752	0,77	0,769	0,302
yolov7-tiny	default-transfer	only_sl	0,818	0,761	0,79	0,822	0,345
yolov7x	default-transfer	only_sl	0,837	0,719	0,77	0,747	0,303
yolov7	target-transfer	no_kom	0,856	0,831	0,84	0,866	0,381
yolov7-tiny	target-transfer	no_kom	0,875	0,808	0,84	0,867	0,398
yolov7x	target-transfer	no_kom	0,868	0,817	0,84	0,863	0,389
yolov7	target-transfer	only_ch	0,980	0,915	0,95	0,950	0,513
yolov7-tiny	target-transfer	only_ch	0,976	0,945	0,96	0,969	0,511
yolov7x	target-transfer	only_ch	0,952	0,904	0,93	0,937	0,498
yolov7	target-transfer	only_kom	0,855	0,568	0,68	0,623	0,309
yolov7-tiny	target-transfer	only_kom	0,860	0,712	0,78	0,781	0,412
yolov7x	target-transfer	only_kom	0,772	0,586	0,67	0,669	0,329
yolov7	target-transfer	only_sl	0,796	0,787	0,79	0,797	0,287
yolov7-tiny	target-transfer	only_sl	0,778	0,778	0,78	0,790	0,324
yolov7x	target-transfer	only_sl	0,821	0,779	0,80	0,811	0,317

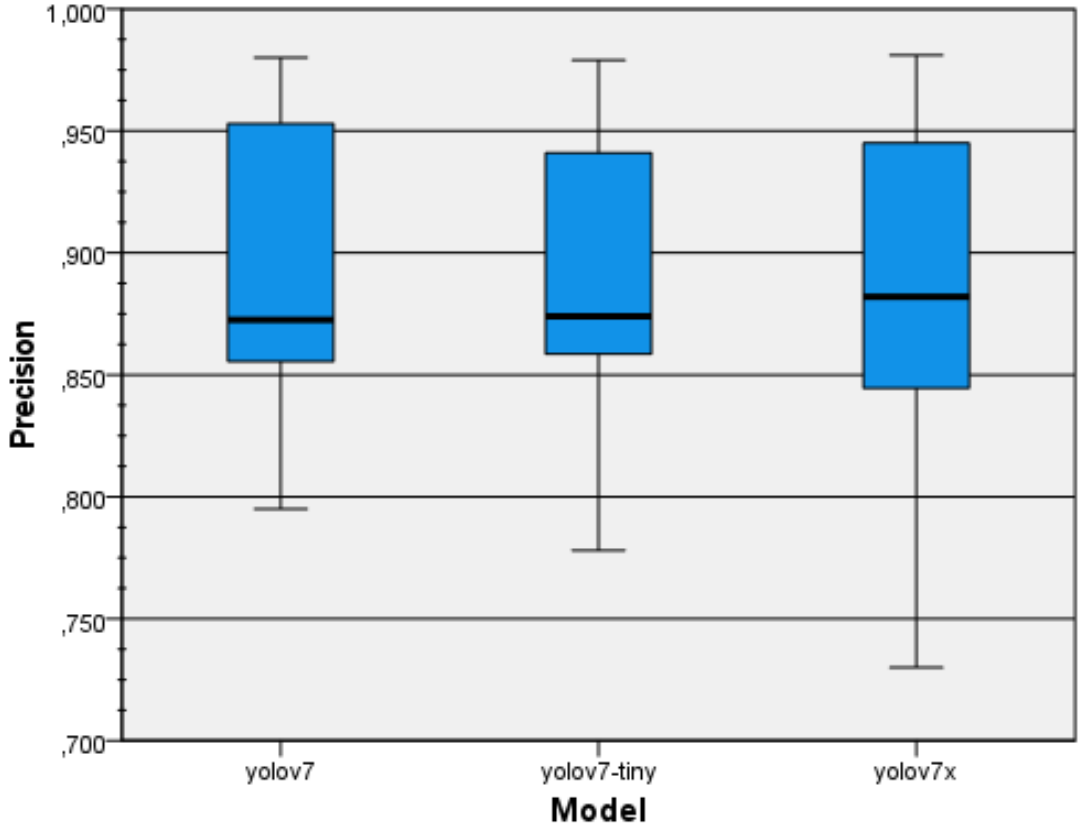
Appendix E: Test boxplots

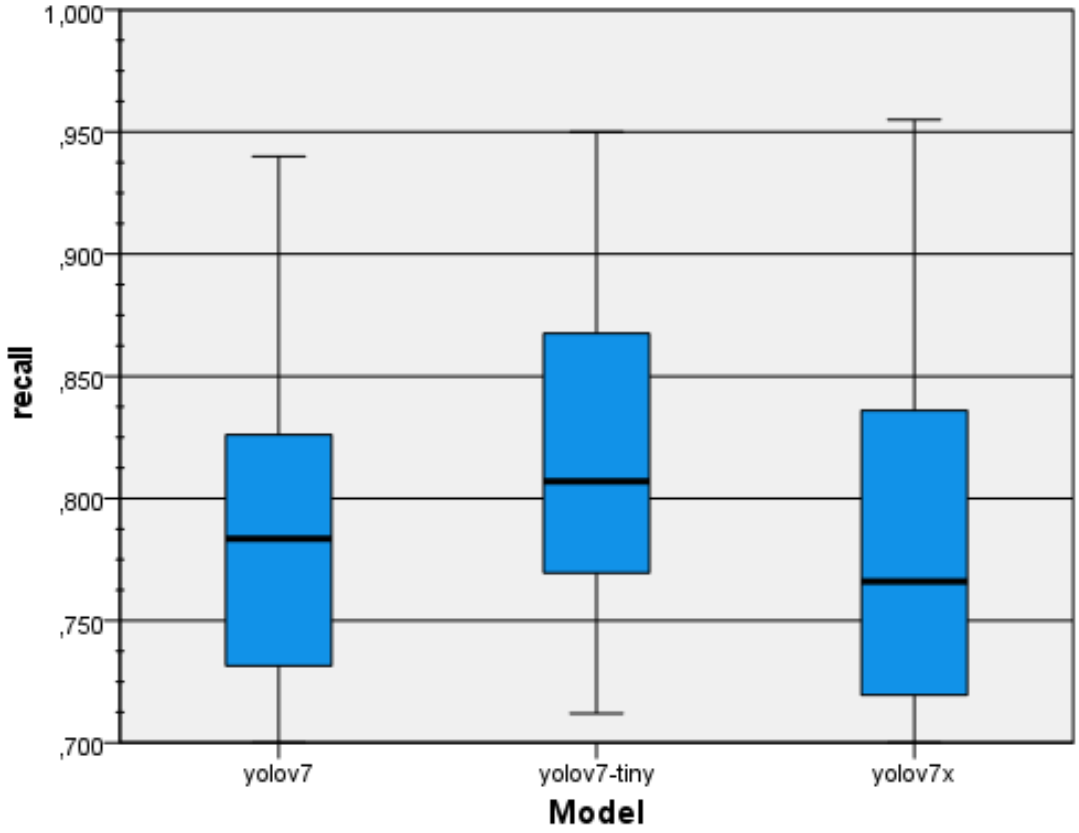


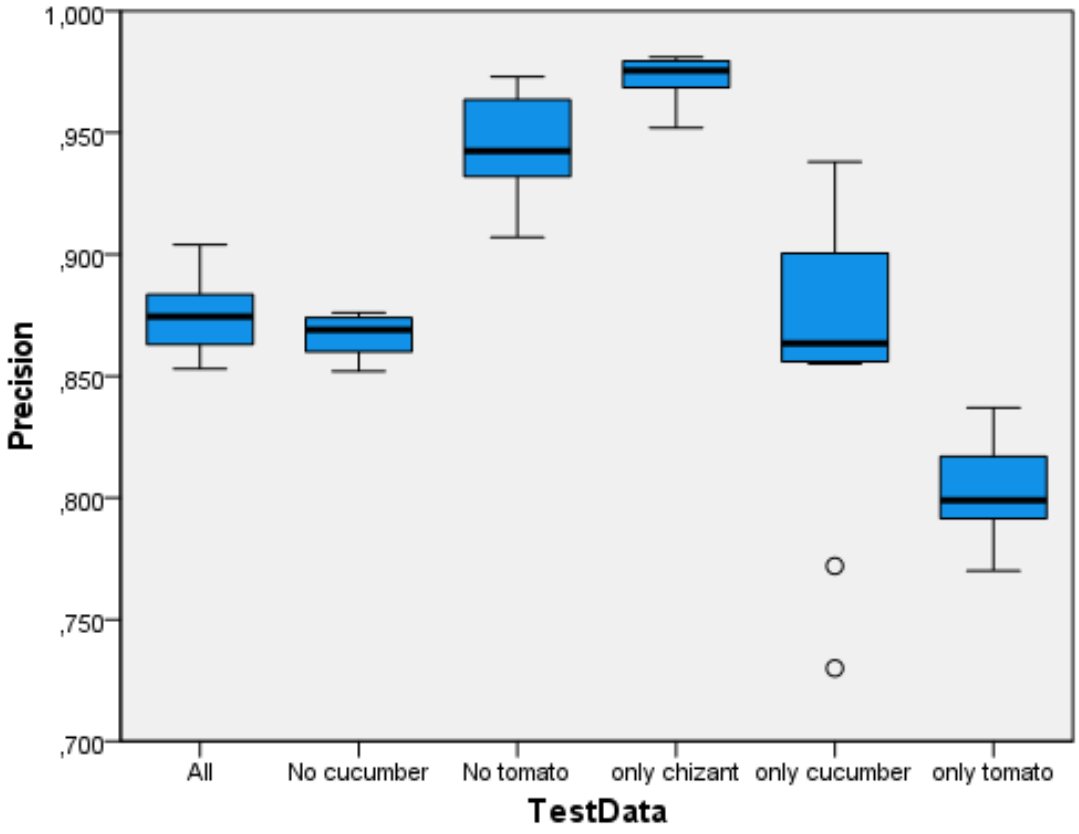

1,000
,900
,800
,700
,600
,500
recall
Default
Default transfer
Target
Target transfer
Training_method

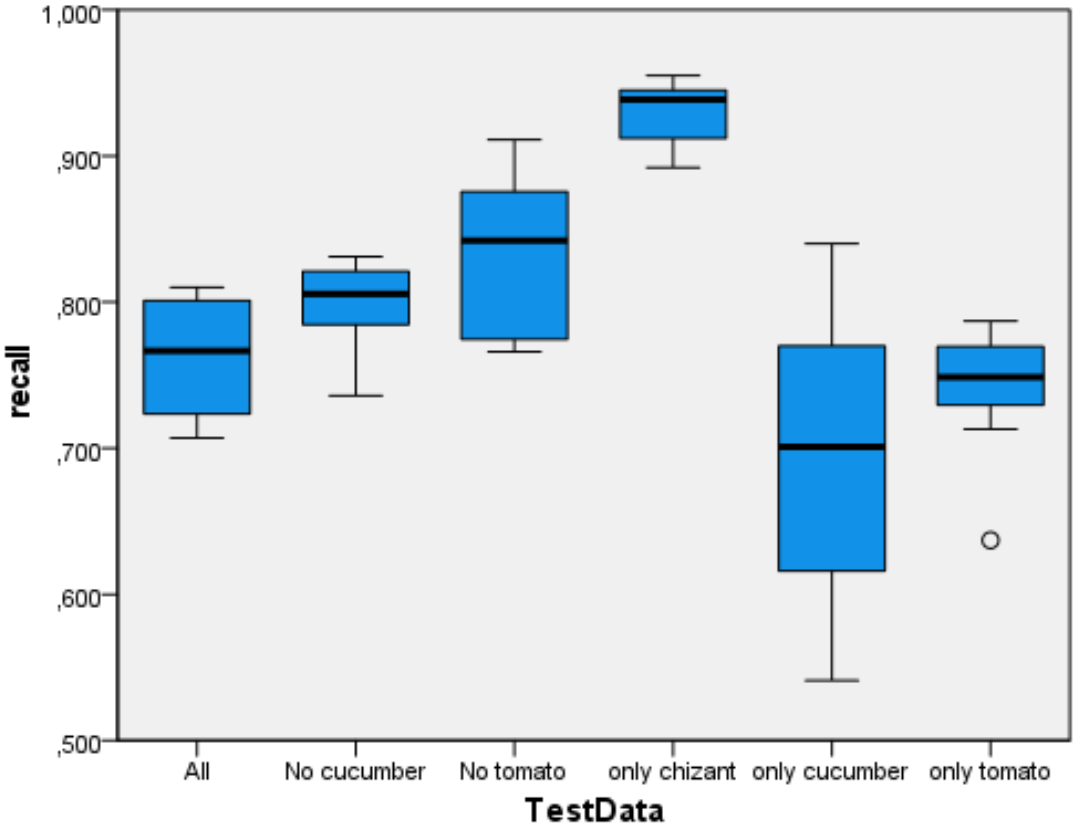

1,000
,950
,900
,850
,800
,750
,700
mAP0.5
Default
Default transfer
Target
Target transfer
Training_method

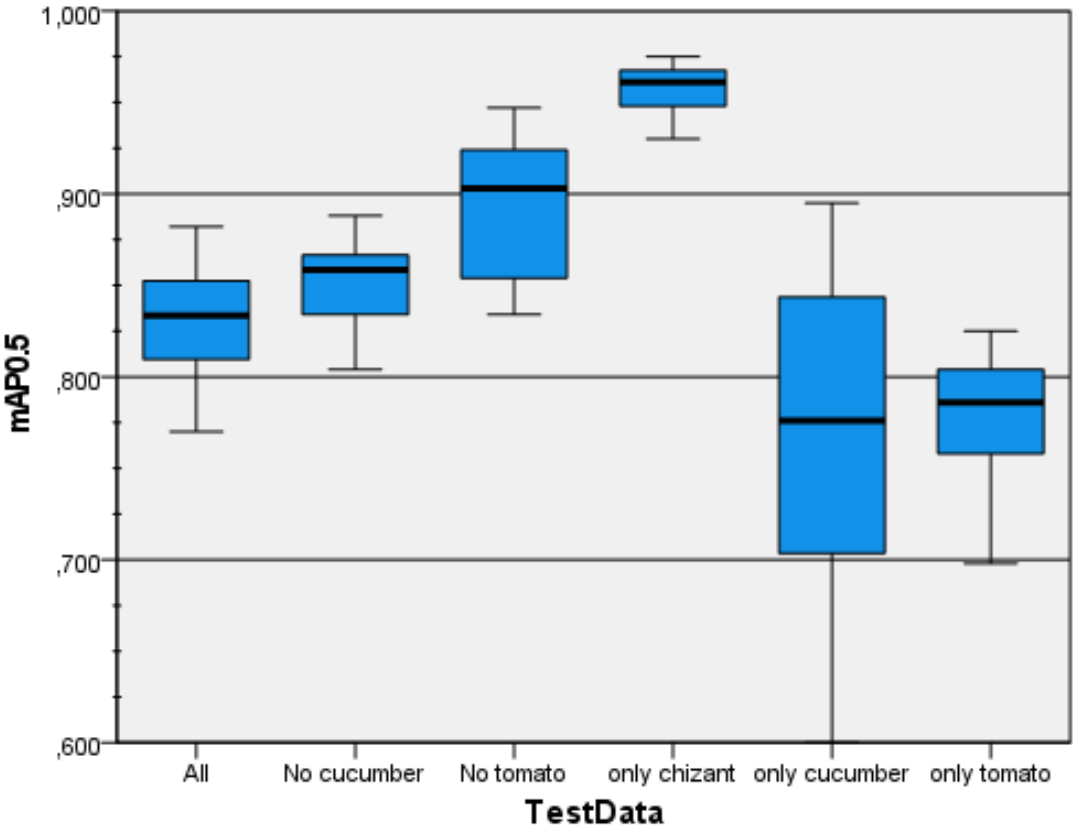

,550
,500
,450
,400
,350
,300
,250
mAP0.95
Default
Default transfer
Target
Target transfer
Training_method

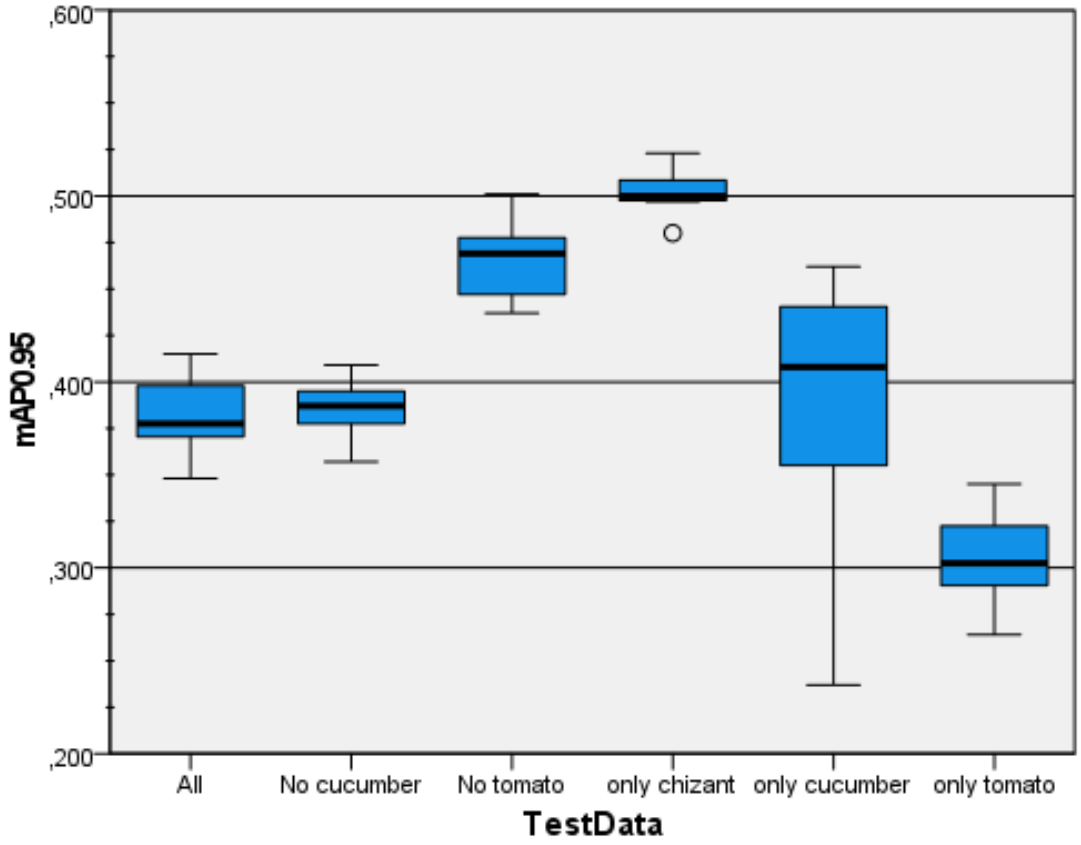

1,000
,950
,900
,850
,800
,750
,700
Precision
yolov7
yolov7-tiny
yolov7x
Model

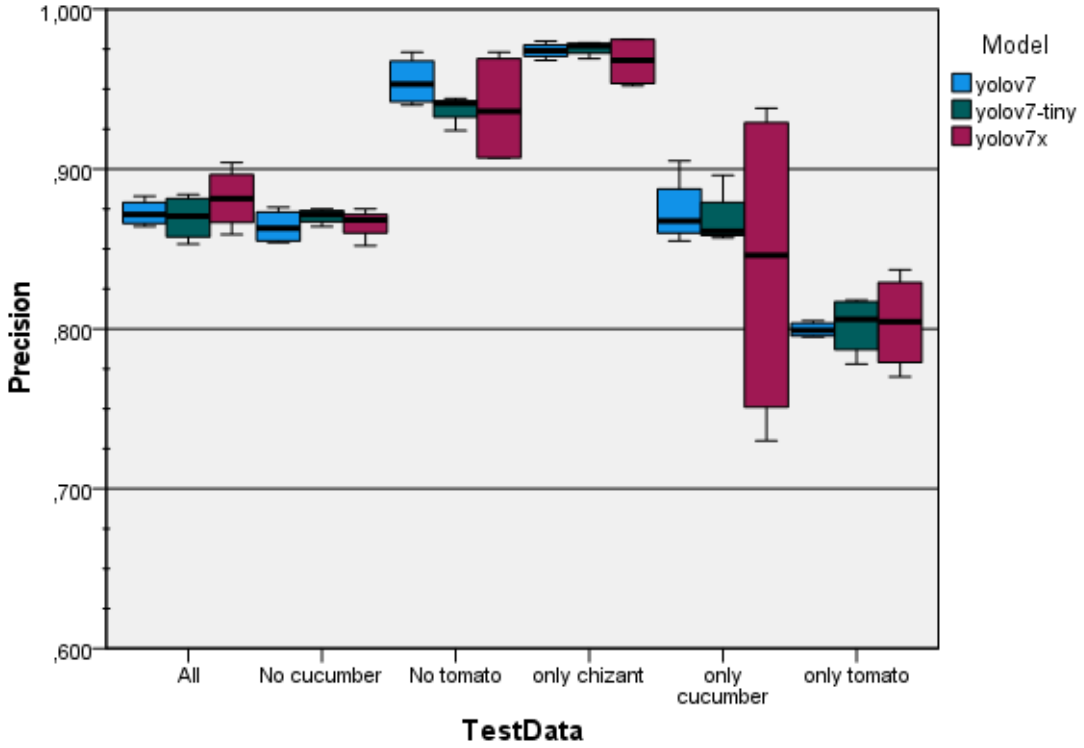

1,000
,950
,900
,850
,800
,750
,700
recall
yolov7
yolov7-tiny
yolov7x
Model

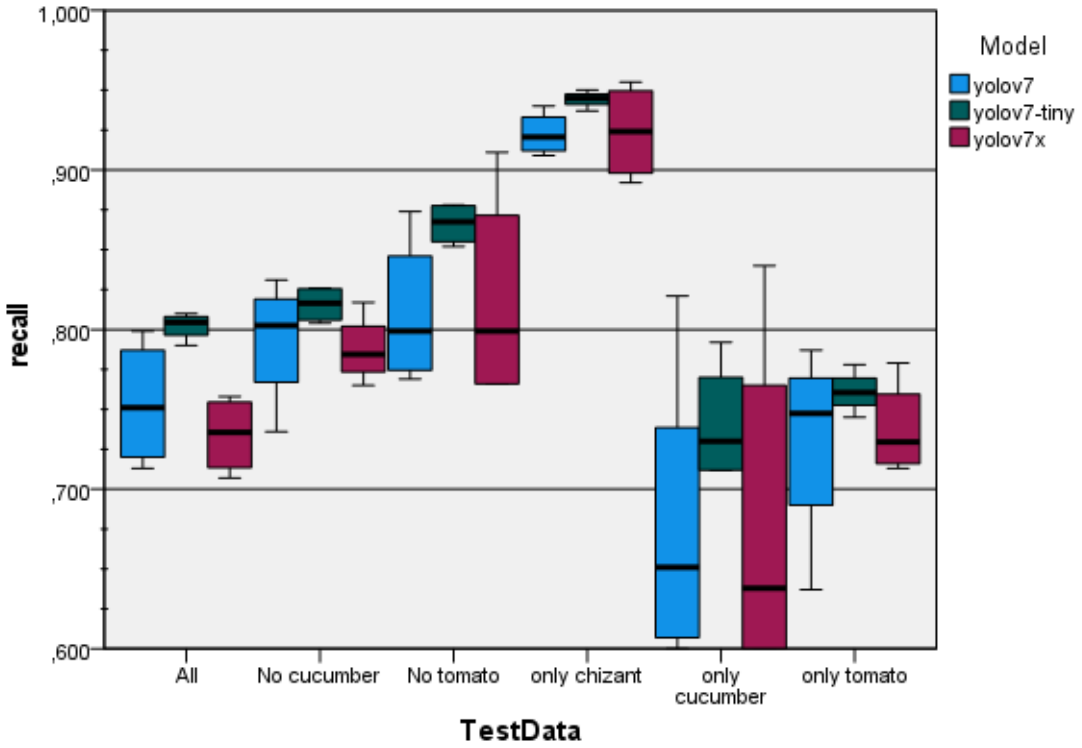

1,000
,950
,900
,850
,800
,750
,700
Precision
All
No cucumber
No tomato
only chizant
only cucumber
only tomato
TestData

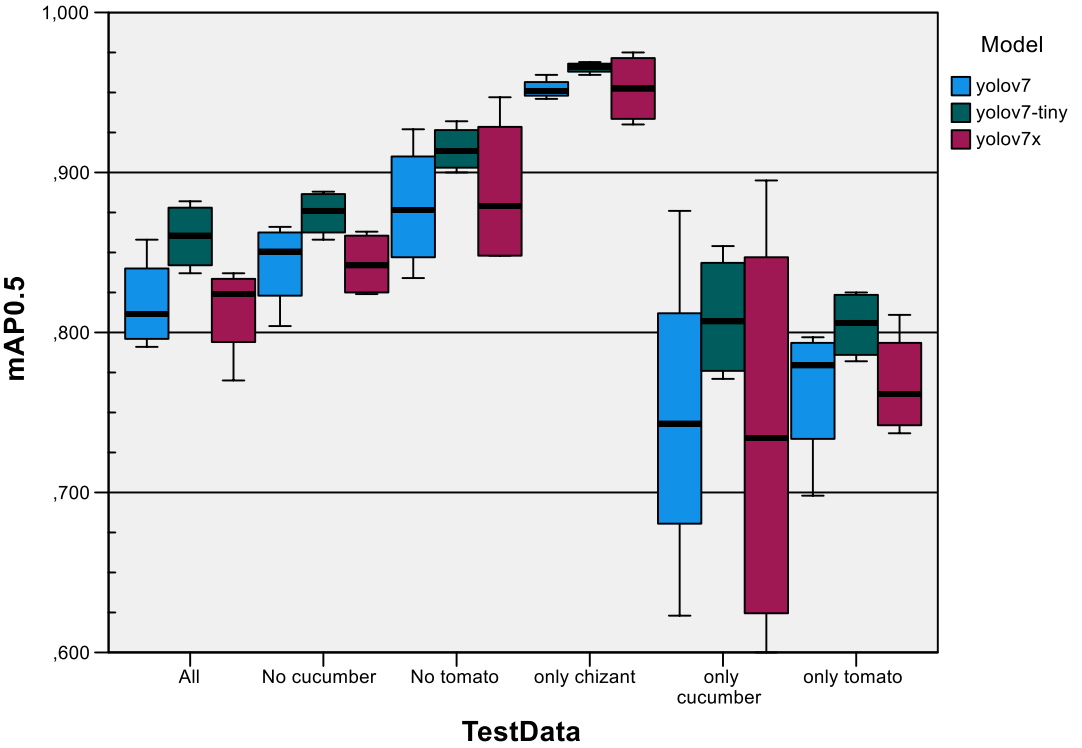

1,000
,900
,800
,700
,600
,500
recall
All
No cucumber
No tomato
only chizant
only cucumber
only tomato
TestData

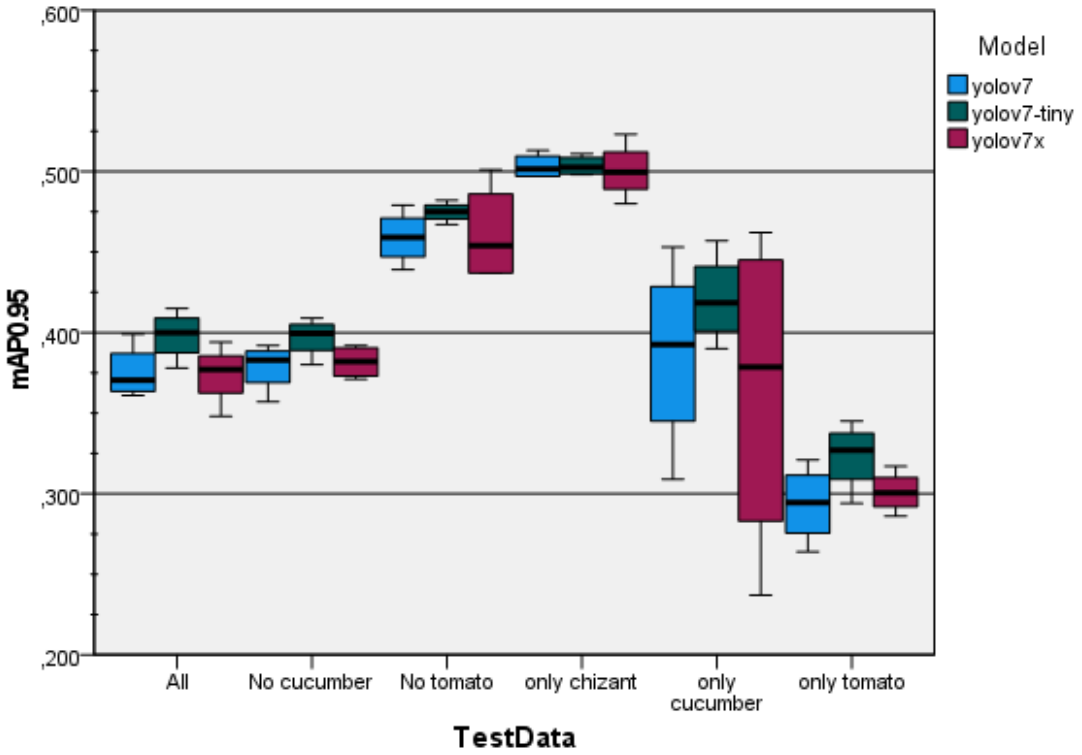

1,000
,900
,800
,700
,600
mAP0.5
All
No cucumber
No tomato
only chizant
only cucumber
only tomato
TestData


,600
,500
,400
,300
,200
mAP0.95
All
No cucumber
No tomato
only chizant
only cucumber
only tomato
TestData


1,000
,900
,800
,700
,600
Precision
All
No cucumber
No tomato
only chizant
only cucumber
only tomato
TestData
Model
yolov7
yolov7-tiny
yolov7x


1,000
,900
,800
,700
,600
recall
All
No cucumber
No tomato
only chizant
only cucumber
only tomato
TestData
Model
yolov7
yolov7-tiny
yolov7x


1,000
,900
,800
,700
,600
mAP0.5
All
No cucumber
No tomato
only chizant
only cucumber
only tomato
TestData
Model
yolov7
yolov7-tiny
yolov7x


,600
,500
,400
,300
,200
mAP0.95
All
No cucumber
No tomato
only chizant
only cucumber
only tomato
TestData
Model
yolov7
yolov7-tiny
yolov7x