DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

IDP Report

Preprocessing of Sugar Beet Images for Hail Damage Analysis using Object Detection

Author:	Maximilian Michallik
Supervisor:	Prof. Dr. Senthold Asseng
Advisor:	M.Sc. Malte von Bloh
Submission Date:	Submission date

Abstract

Machine learning algorithms are gaining more and more importance in many application fields. Also in agriculture, artificial intelligence can be used to improve certain workflows by making them more efficient.

In this report, a preprocessing algorithm for sugar beet plant detection is presented. It is part of a machine learning pipeline to predict the hail damage of plants in sugar beet fields. The goal of this preprocessing step is to standardize the images taken with the mobile application to improve the results of damage prediction. Available data now has different angles and heights of the camera. By detecting the sugar beets, the images should be modified in a way that only one plant is contained in the center of the image.

The used model is YOLO, a state of the art object detection algorithm. Different aspects of training are compared and results of detecting sugar beet plants are presented. Experiments showed that pretrained models with higher data augmentation achieve the best accuracy. Because of the small difference in predictions of different sized architectures, we decided to focus on a smaller one for live prediction in the mobile application for predicting the damage of sugar beets.

Contents

1 Introduction

Machine Learning and Computer Vision has gained much importance in the last years. It can be used in many application fields, such as smart plant monitoring. In this context, it can be used to simplify certain workflows. One example is the hail damage detection of sugar beets.

The idea is to develop a system or application which automatically predicts the damage of plants. This has several advantages, for example less time has to be spent analyzing the fields and possible more accurate results can be achieved.

More concrete, a mobile application was developed, which allows to take images of sugar beet plants. The fotos taken should then be preprocessed and sent to a backend server which analyzes the images and predicts the damage.

In the context of this report, the preprocessing step of such a mobile application is presented. The general idea of the preprocessing step in this application is to standardize the image format of the fotos taken. This means that factors like the number of sugar beet plants, the angle in which the foto was taken and many more things are not often optimal for the model that is predicting the damage. In general, the images that we use for training are taken from directly above the plant in a 90 degrees angle. In the best case, the image only contains one plant which is directly in the center. To be therefore consistent with training images, the new pictures of the plants should ideally be in the same format with similar settings. Therefore, the preprocessing step helps to get better results by standardization of the images.

Object detection has gained much more importance in the last years. As an example, **object_detection_history** present that the number of publications about "object detection" or "detecting objects" has increased from approximately 200 in 2005 to about 100 in 2017 and even nearly 1200 in 2018. They also present some state of the art algorithms for this task. Deep convolutional neural networks play an important role for them as feature extractor for the so called backbone part of a network. They call it the "engine" of a detector as a very important part. The authors name different concrete examples as state of the art object detection algorithms. Three of them are discussed in the following. There are different tradeoffs that influence the design of the network. Huang, Rathod, C. Sun, et al. 2017 presents a discussion about different feature extractors for object detection algorithms.

Faster R-CNN (region-based convolutional neural networks) presented by **faster_rcnn** is an improvement of Fast R-CNN (R. Girshick 2015). The authors describe that the problem of other detectors is the region proposal of objects meaning that the location of bounding boxes is not fast enough by now. Therefore, they introduce a region proposal network which automatically locates objects.

As another concrete example, J. Dai, Y. Li, He, and J. Sun 2016 introduces R-FCN (Region-based fully convolutional networks). They call it an improvement of Fast and Faster R-CNN and improve the times of locating an object in an image. Almost all computation is made on the whole image by so called position-sensitive score maps.

A next example is single shot multibox detector (SSD) which is presented by W. Liu, Anguelov, Erhan, et al. 2015. For this method, location proposals are not even necessary anymore. It is a single deep neural network which predicts scores and parameters of default bounding boxes of objects in an image at inference time.

As fourth example, **yolov1** introduces the first version of the so called YOLO network. It is used in this work and a more concrete description and evolution is provided in chapter 2.

These algorithms are used in many different application fields. This is also the case for agriculture. For example Han and J. Li 2022 applied an improved fifth version of the YOLO network to detect the heads of wheat plants. They use the data set provided by David, Serouart, Smith, et al. 2021, to learn their network. This application and availability of the images with their labels shows the importance of this field.

Another example of object detection in agriculture is presented by Kragh, Jørgensen, and Pedersen 2015. They use this technique to classify terrain using 3D lidar data.

bale_detection also presented one application of object detection in agriculture. Especially, they focus on one big problem of real world scenarios which is the lack of labeled data. They present a pipeline which consists of one object detection algorithm trained on a small amount of images, transfer learning and an optimized detector with the help of synthesized data.

Similar to our application case, also detected damage of sugar beet plants. The difference to our goal was to predict mechanical damages caused by harvesting the plants. The camera used was located directly inside the harvester machine. The author compared different architectures including YOLOv4 (Bochkovskiy, C.-Y. Wang, and Liao 2020), region-based fully convolutional network (R-FCN) and faster regions with convolutional neural network features (Faster R-CNN). The results of their work show that YOLOv4 predicted the sugar beets with better accuracy and also faster than the other two methods.

All in all, there is a wide range of application cases of object detection in agriculture.

For a broader overview, refer to **applications_review**. The survey provides state of the art algorithms and techniques for object detection and tracking algorithms especially for unmanned aerial vehicles.

2 Theoretical Background

There are different types of machine learning algorithms which have various application cases. In this case, an algorithm which performs object detection is used. In the following, the open source architecture YOLO ("you only look once") is presented. First, the general principle of object detection is introduced, followed by the description of the architecture of deep YOLO networks. The principle of object detection is to combine Classification with detecting the boundaries of class instances which is exactly what needs to be done in the preprocessing step of sugar beet plants. By detecting the object boundaries of those plants, the image can be cropped to the right size which results in a standardized image format for the following regression task predicting the value of the damage (between 0 and 1).

The state of the art architecture for object detection is YOLO which was first introduced by **yolov1**. Since then, two improved versions were published by the original author. These architectures were called YOLOv2 (**yolov2**) and YOLOv3 (**yolov3**). After that, other authors published the next version YOLOv4 (Bochkovskiy, C.-Y. Wang, and Liao 2020) and also a newest version YOLOv5 can be found.

The advantage of YOLO-networks is that they are much faster than other object detection algorithms like for example R-CNNs (R. B. Girshick, Donahue, Darrell, and Malik 2013) which let the model run on a very high number of regions in the image. The idea of YOLO is that the network runs once for the complete image, predicting the bounding boxes and corresponding classes at once. It finds regions in the image which are assigned predicted bounding boxes and probabilities. In the following, the general idea of YOLO is introduced, followed by the incremental improvements in the following versions.

2.1 General Architecture of YOLO

As proposed in the first introduction to YOLO by **yolov1**, the network is designed to find objects and the corresponding boxes in a global manner meaning that the image is processed as a whole once. Therefore, a $S \times S$ grid divides the image into smaller parts. Each of those cells is responsible to predict B bounding boxes and corresponding confidences. This confidence is defined as $Pr(Object) * IOU_{pred}^{truth}$ with IOU the interface over union of the true box and the predicted one respectively. The

bounding boxes consist of five predictions each, x, y, w, h and the confidence. The first four numbers represent the bounding box with the coordinates of the center (x, y) and the width and height (both relative to the whole image). Additionally, the cell predicts $Pr(Class_i|Object)$, which are the conditional class probabilities. Only one set of class probabilities are predicted per cell. All in all, the class probabilities and box confidence predictions are multiplied resulting in the class-specific confidence scores for each box: $Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth}$. The result is a tensor of size $S \times S \times (B * 5 + C)$. Figure 2.1 depicts the idea of predicting bounding boxes with corresponding classes.

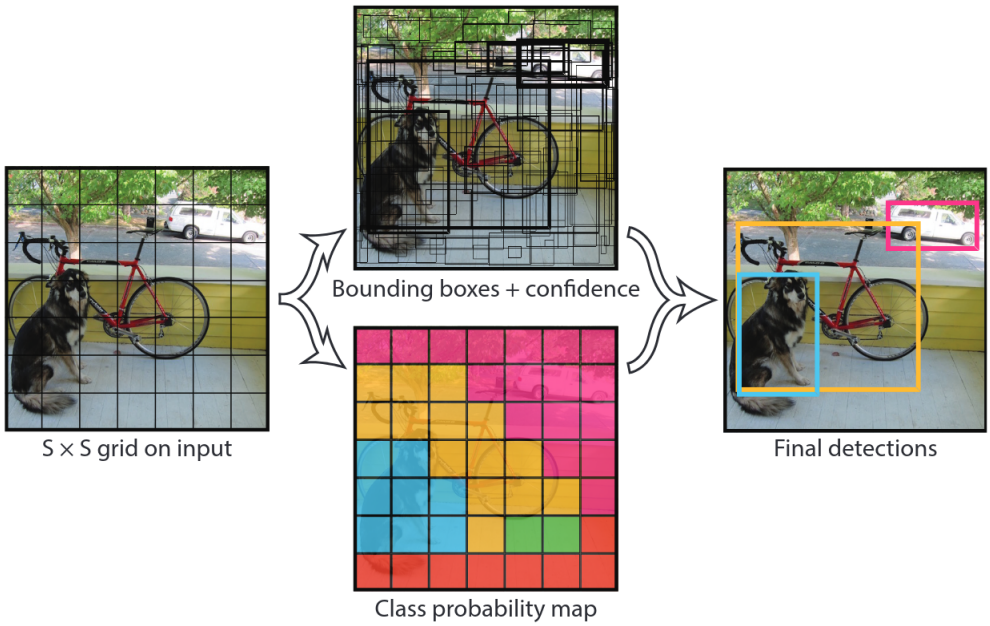


Figure 2.1: The model divides the image into $S \times S$ grids. For each of them, B bounding boxes and C class probabilities are predicted. The overall result is a tensor of size $S \times S \times (B * 5 + C)$. Taken from **yolov1**.

To achieve this, a convolutional neural network is used in combination with fully connected layers. The first part extracts features while the latter part predicts the probabilities and coordinates of the bounding boxes. It consists of 24 convolutional layers and 2 fully connected ones. The concrete network can be seen in figure 2.2.

All in all, the first network has comparable accuracies as other object detection algorithms such as R-CNN or Fast R-CNN. The most important advantage of this architecture is the real time capability. Especially for this preprocessing task, the time

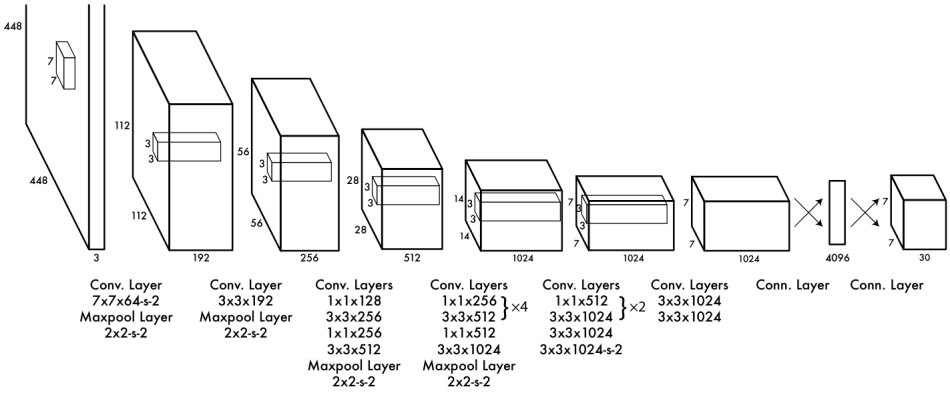


Figure 2.2: The network consists of 24 convolutional and two fully connected layers. In this case, $S = 7, B = 2$ and $C = 20$, so a tensor of size $7 \times 7 \times 30$ results. Taken from **yolov1**.

plays an important role. The authors also present a different version of this network focusing on faster inference times. It is called Fast YOLO and only has 9 convolutional layers (instead of 24). All other parameters and layers are the same.

2.2 Incremental Improvements

The first architecture was improved incrementally in different versions. The original authors were involved until version 3 (**yolov1**; **yolov2**; **yolov3**). After that, other authors published improvements as YOLOv4 Bochkovskiy, C.-Y. Wang, and Liao 2020. There is no publication to YOLOv5 yet, although there is already an implementation by Jocher 2022a and a detailed description of the network, pretrained models and is even contained in PyTorch hub Jocher 2022b.

YOLOv2 With the first improvement which is published in **yolov2** (called YOLO9000: Better, Faster, Stringer), the problem of the localization errors of the boundaries of objects is addressed. The authors call it a better, faster, and stronger version of the YOLO network because of the following improvements.

Depending on the authors, it is made better regarding the recall and localization of bounding boxes. The added batch normalization helps in regularization of the model. The dropout from the model can be left out without overfitting. It is also made better in terms of the image resolution. First, it is trained with the full resolution of 448×448 for 10 epochs. Afterwards, the network is fine tuned on detection. Additionally, the

last layers which were originally fully connected ones, are replaced by anchor boxes for the prediction of the bounding boxes. A disadvantage of doing this is that the box dimensions are hand picked. By running k-means clustering on the training set bounding boxes, it can be avoided and good priors can be found automatically. A second disadvantage of using anchor boxes is the instability of the model due to the coordinates of the centers of the boxes. These are calculated depending on the predicted box boundaries. Instead, it is better to predict the location coordinates relative to the cell's location resulting in values between 0 and 1. Another change is that the feature map is made finer so that the predictions get more accurate. One last change to make the network better is that multi-scale training. Due to the use of convolutional and pooling layers, the input resolution of images can easily be changed. This is used in the training to vary this resolution every few epochs. It makes it all in all more robust and not depending on the input resolution.

Additionally to making the network better, it is also made faster. The authors present this new network as darknet-19. It has many advantages compared to the previous one such as high accuracy and faster times. The network consists of 19 convolutional layers and 5 maxpooling layers.

To make the network stronger, hierarchical classification is used. Therefore, the principle of subclassing is used. If a concrete dog such as Norfolk terrier is detected, it is also detected as terrier and dog. Additionally, dataset combination with WordTree is used where multiple datasets are combined. With all this, joint classification and detection is possible meaning that classification and object detection datasets can be used to efficiently train the network.

All in all, this second version of YOLO already brings improvements to the network which is further made better in the next versions.

YOLOv3 The main difference to YOLOv2 is that in the new version presented by **yolov3** (YOLOv3: An Incremental Improvement), a new network architecture for feature extraction is used. It is called Darknet-53 and the structure can be seen in figure 2.3.

It is based on the architecture from YOLOv2 and Darknet-19 and has 53 convolutional layers. It is a deeper network, although the speeds are still much better than other comparable object detection models.

The results of the authors show that the accuracy is comparable with ResNet-101 and ResNet-152, Top-5 is even best compared to the others. Additionally, the detectable frames per seconds are much higher than the other models.

The authors also present idea that did not improve the network. One thing is anchor

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 2.3: Precise description of Darknet-53 network which consists of 53 convolutional layers. Taken from **yolov3**.

box x,y offset predictions which predicts the coordinates as a multiple of the box width or height using linear activation. Additionally, linear x,y predictions instead of logistic which directly predicts the coordinate offset using linear instead of logistic activation decreased the model accuracy. Also the use of focal loss which adds an additional factor to the standard cross entropy loss did not improve the model. Lastly, a dual IOU threshold and truth assignment which is comparable to the use in the Faster R-CNN network was tried out. Essentially, two thresholds is defined which partitions the interval $[0, 1]$ into three parts. The highest part is a positive example, the middle part is ignored and the lowest part is a negative example.

YOLOv4 Currently, many object detectors are built up in a similar way. The rough structure can be seen in figure 2.4.

YOLOv4 Bochkovskiy, C.-Y. Wang, and Liao 2020 also consists of such parts. It is a One-Stage Detector and therefore only has dense prediction. As a backbone, CSPDarknet53 (**Wang_2020_CVPR_Workshops**) is used. The neck consists of SPP (spatial pyramid pooling) (He, Zhang, Ren, and J. Sun 2015) and PAN (path aggregation network for instance segmentation) (S. Liu, Qi, Qin, et al. 2018). Finally, the head (dense

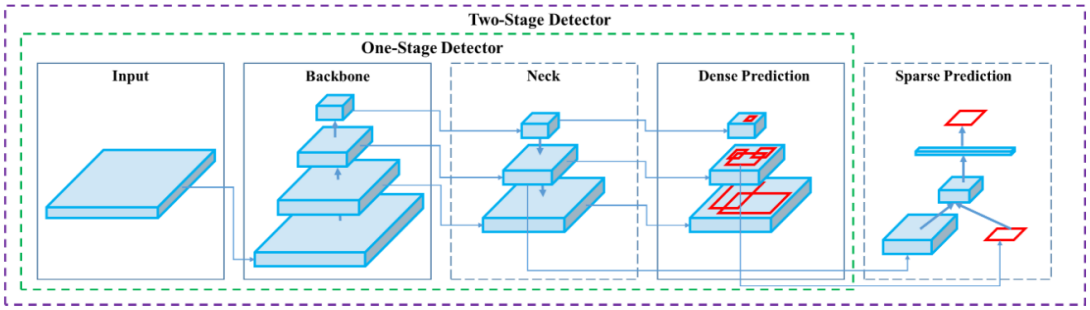


Figure 2.4: The different parts of a detector. Input is given to the Backbone which forwards it to the Neck. The last part is the Dense Prediction. A Two-Stage-Detector differs from a One-Stage-Detector as it has an additional Sparse Prediction at the end. Taken from Bochkovskiy, C.-Y. Wang, and Liao 2020.

prediction) is the YOLOv3 network by **yolov3**. In addition, it uses some techniques such as data augmentation to improve accuracy for backbone and detector.

All in all, the presented architecture with the additional techniques is superior to other state of the art object detection algorithms in both speed and accuracy.

2.3 Current Architecture YOLOv5

The current version YOLOv5 (Jocher 2022a) again has further improvements compared to YOLOv4. The structure is similar to the previous one, also with the same backbone (CSPDarknet 53) and head (YOLOv3). One first difference is that instead of SPP, a faster version SPPF is used which is more than twice as fast. Together with PAN, this builds up the neck part.

Different Network Sizes There are five different network sizes available. They differ in the number of layers and parameters and are called YOLOv5n (nano, smallest), YOLOv5s (small), YOLOv5m (medium), YOLOv5l (large) and YOLOv5x (extra large). The differences can be seen in table 2.1.

The values are measurements made by the authors and are available on the release notes (version 6.1) (Jocher 2022c).

Additional Features As data augmentation, different techniques are used. For example mosaic concatenates several images to one big image. The copy paste technique copies objects from images to other pictures. Additionally, random affine transformations are applied to the images, such as rotations, scaling, translation and shear.

Size	Speed (ms)	Parameter (million)	mAP^{val} (0.5)
nano	45	1.9	45.7
small	98	7.2	56.8
medium	224	21.2	64.1
large	430	46.5	67.3
extra large	766	86.7	68.9

Table 2.1: Comparison of the different network sizes regarding inference speed, number of parameters and mean average precision with IOU threshold of 0.5. Values taken from Jocher 2022a.

Similar to mosaic, MixUp combines different images but with the difference that the pictures are directly laid behind each other. Additional settings like hue, saturation and value are randomly adjusted. Together with random horizontal flips, the variety of the training set is increased which leads to better object detection accuracy. More details on data augmentation can be found in the Chapter Results.

3 Methodology

To achieve the goal of detecting sugar beet plants, the presented YOLOv5 is used. The exact goal is to develop a preprocessing algorithm which standardizes images of sugar beets for a damage prediction regression task. As described, the images should be standardized in a way that each picture contains one sugar beet plant photographed in a 90 angle to the ground from above. To train the network on the dataset, a nvidia gpu with 24267 MiB memory was used.

In the following, the given dataset, the training of the network, the evaluation of the models and mobile detection will be presented.

3.1 Dataset

There are essentially two sources for images of sugar beets and other plants available. One part consists of images taken in sugar beet fields and the other one is Imagenet (Deng, Dong, Socher, et al. 2009) where a large number of different images with classes is available.

The number of images of sugar beet plants is 10087. This set can be categorized in multiple different types called (1), (2), (3) and (4). The first type is an image with many small sugar beets. The other one is an image with older and bigger plants where the concrete object boundaries are hard to see. One example for each class (1) and (2) can be seen in figure 3.1.

The first one (3.1a) is taken from further away and contains multiple smaller plants. The object boundaries can be seen very clearly. This is not the case for the second image type which is depicted in figure 3.1b. It contains bigger plants and the boundaries can not be seen clearly because of the neighboring plants. It is not even easy for the human eye to make find the leafs of a distinct plant. The majority of the whole dataset is of the second type. An example of the third image type can be seen in figure 3.1c. It is a screenshot of a drone video and contains multiple larger plants. Also in this case, the object boundaries can not be seen very clearly because of the overlapping leafs. The last type can be seen in figure 3.1d. This is an example of the images provided by the partner. In general, these pictures are not taken in a standardized way. For example in this case, the angle of the camera is not 90 and multiple plants can be seen from the side. All in all, table **??** shows an overview over the number of images of each type.

(a) Image type (1). Shows multiple small plants and the boundaries can be seen perfectly.

(b) Image type (2). Shows one larger plant, boundaries are overlapping with neighboring plants.

(c) Image type (3). Shows many larger plants. The boundaries of single sugar beets can hardly be seen.

(d) Image type (4). Shows plants from a different angle than 90.

Figure 3.1: Example images of types (1), (2), (3) and (4).

The row with pictures from Imagenet describes the images taken without any labeling. They do not contain any objects of type sugar beet. Figure 3.2 visualizes the distribution.

The numbers of images for each type are 9461 for the first type and 123 for the second one. They are all provided by TUM. In practice, most images will be of bigger plants because the damage prediction is more important in those cases. 244 images are of

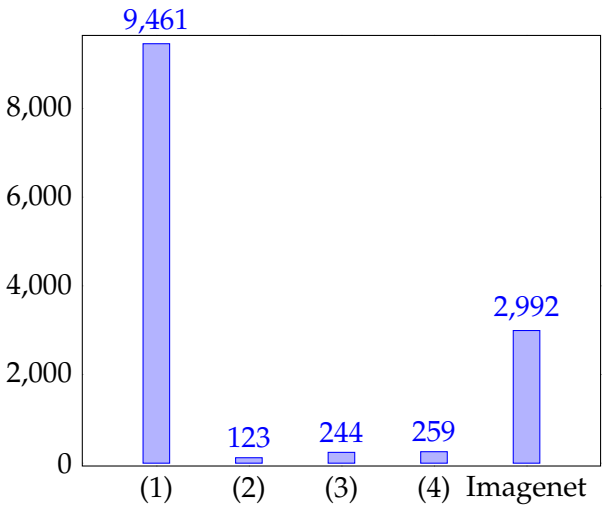


Figure 3.2: Overview over the data set with the available images. (1) and (2) are given by TUM. Images of type (3) are screenshots from drone videos and (4) from partner.

type (3) and they are all screenshots from drone videos. Partner provided 259 usable images and they are of type (4). Additionally, Imagenet was used. 2992 images with other plants, flowers and persons were used. These images were not labeled at all. An exact labeling by hands could have been possible but due to time constraints and the fact that only the boundaries of sugar beets have to be detected exactly, this was not done. Possible other solutions would have been to include multiple other classes like concrete plant types or "person" but this is not needed in this application.

To train the network, the pictures have to be labeled. For each image file, one text-file has to be made which contains the information of the image. For each object, one line with five numbers has to be added. The first one is the class. The second and third one are the coordinates of the center of the bounding box (x- and y-coordinate). These have to be relative to the width and height of the image. The third and fourth ones are the length in x- and y-direction. They also have to be relative to the image's height width. First experiments showed that the labeling has to be done manually to obtain reasonable results. It could be seen that only labeling the images with multiple small plants and annotating the other ones automatically with `0 0.5 0.5 1 1` is not sufficient to detect the exact boundaries of bigger plants. Such a labeling exactly means that the center of the bounding box is in the middle and the boundaries are exactly the same as the image boundaries. In most cases, the whole image was detected as a whole

sugar beet plant, which was definitely not the goal. The manual labeling was done with the tool labelImg by *LabelImg* n.d. which provides a graphical user interface to draw the rectangles around the objects. An example of such a labeling of a bigger plant can be seen in figure 3.3.

Figure 3.3: Labeling images with labelImg which provides a graphical user interface to draw bounding boxes into the pictures.

As you can see in this figure, the plant in the center of the image is labeled as sugar beet. The rectangle of the object has to be as exact as possible to obtain best results in predictions. Although especially in the case of big plants, this is not always easy because neighboring sugar beets are overlapping and the plant boundaries can not even be seen very clearly with the human eye. Nevertheless, all available images were labeled manually to get best possible results. An overview of the complete data set can be seen in figure 3.2.

All in all, the most available images contain about one larger plant from above which can also be seen in figure 3.2. The biggest variety of image types can be seen from the Partner. These were taken in very different angles and contain various damages ranging from nearly no to high destruction. The drone screenshots are similar to the TUM data set but are taken from a bit more far above. The drones captured most of the time more than one large plant at a time resulting in overlapping sugar beets again. This overview in the figure 3.2 shows that much data was gathered and labeled. Compared to the use case where the plants should be detected, the available data is very suitable for the application as it has many different cases ranging from many plants on one images to only containing one sugar beet. Also the damage classes are distributed meaning that images with very different degree of damage are available.

3.2 Training

For the training of the YOLOv5 model, different parameters and other settings have to be specified. One first thing is the decision between different model sizes. As described in the description of YOLOv5, there are different possibilities. The ones we are concentrating on are the small, medium, and large one. For each of them, a pretrained model on the COCO data set by Lin, Maire, Belongie, et al. 2014 with 300 epochs and default settings are available (Jocher 2022a). This data set contains 328 thousand images with 2.5 million instances of 91 object classes. Alternatively, the raw model without pretraining can be used. In a configuration file, many different hyperparameters can be adjusted such as box, class and obj loss gain. Also the IoU (intersection over union) training threshold can be set. This value determines the fraction of the area of the intersection of two bounding boxes over the union of them. The highest achievable value is 1 which means that the predicted box is exactly the same as the labeled box. Many other parameters for image augmentation are also available. One example therefore is hue, saturation, and value which are then just changed in the image. Also degrees of rotation, translation, scaling, shearing and the perspective can be adjusted. More complex ones are the probability of flipping the image upside down or left-right, respectively. Additionally, mosaic can be chosen. This means that many different images with their labels are concatenated to form a larger image. Also the probability for mixup and copy paste can be chosen independently. Examples for the last three augmentations can be seen in figure 3.4.

Depicted in the top row, mosaic just puts different labeled images together in a random way to form one larger training image. In the middle row, mixup augmentation is depicted. This one is more complex because it copies multiple labeled image overlapping into a larger one. The bottom row depicts the copy paste augmentation which copies single instances of classes to other labeled training images. All in all, these different augmentation strategies help to improve the model's robustness because the variety of input images is increased.

Further settings that have to be made to train the models are defining the train, validation, and testing split. Additionally, the number of epochs can be defined. As optimizer, SGD (stochastic gradient descent), Adam, and AdamW can be chosen. The concrete training scenarios and their settings will be described in the results.

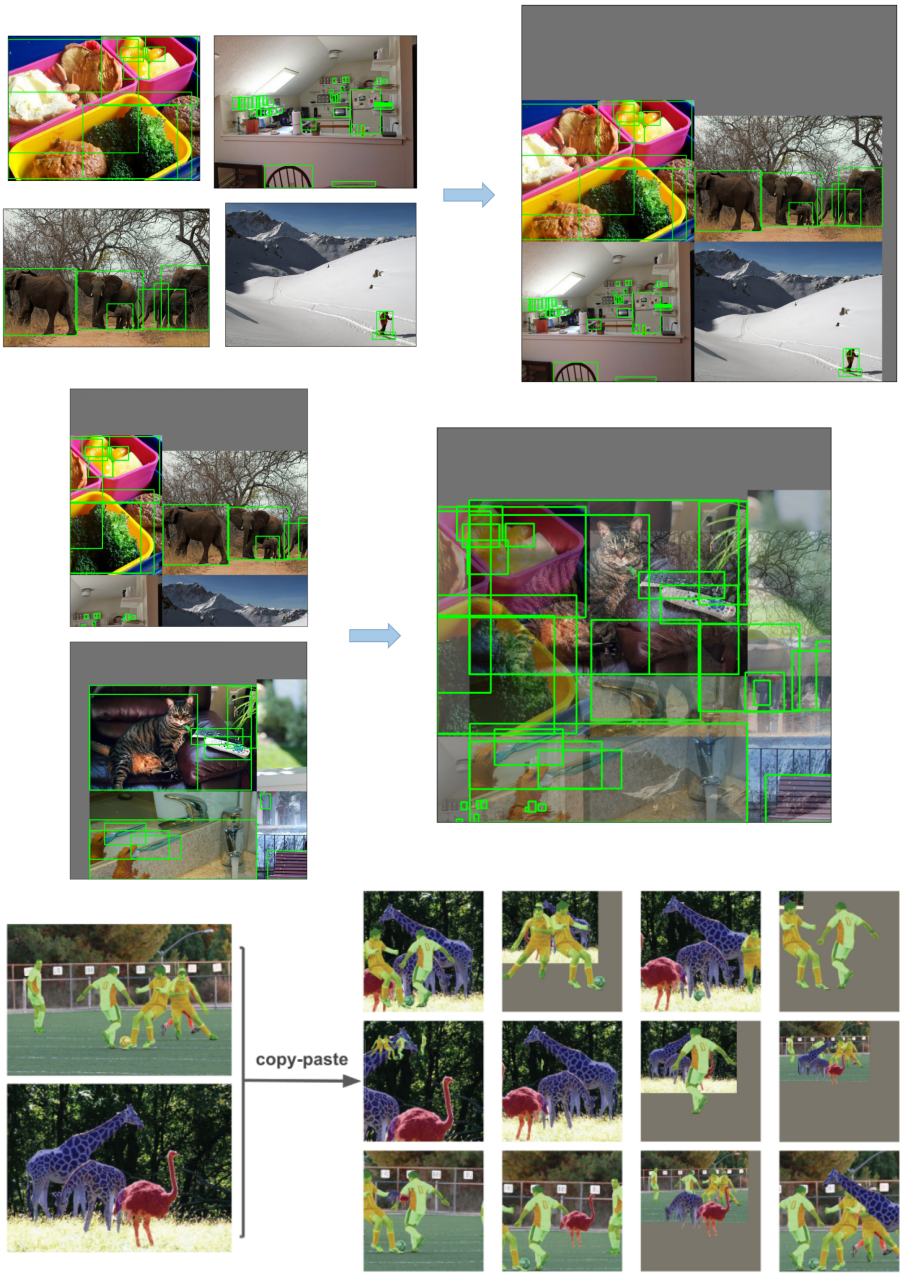


Figure 3.4: Data augmentation types. These techniques can be used to combine different images as new input for training. The first row shows mosaic which just concatenates them. Middle row is mixup laying several images behind each other. Last row is copy-paste which duplicates different objects into other pictures. Taken from Jocher 2022a

3.3 Inference and Evaluation

For the detection of objects, multiple different inputs are possible. Either the images of whole folders, single pictures, or the live stream of the webcam can be processed. In each way, the image is directly labeled with the bounding box of the detected class. Additionally, it is possible to write the result to a text file in the same format as the label for the training images.

To evaluate the model's accuracies, the test set is used. Metrics like the precision and the recall can be directly measured with the framework by Jocher 2022a. Precision and Recall are calculated with

$$P = \frac{T_p}{T_p + F_p}, \; R = \frac{T_p}{T_p + F_n} \tag{3.1}$$

with T_p true positive, F_p false positive, and F_n false negative predictions.

For the three model sizes, different inference times can be observed. For the small one, the prediction takes about 100ms, the medium one takes about 224ms and the slowest is the large one with about 450ms. All times are taken on a normal CPU.

For a live application detecting the objects, the large model is definitely too slow. It takes too much time to detect objects and the live stream updates not fast enough. Even with the medium model, the window is not fluent. Using the small model, a live application detecting the sugar beets can be done. The inference times are not too high to still give a fluent live image.

These inference times leave room for two versions. The first one is just processing the image taken from the field on the server where also the regression model is implemented. The second one is using the model directly in the app so that the user can also see and maybe interact with the bounding boxes of the predicted sugar beets.

As a metric for evaluating how the network performed, the area under curve (AUC) of the P-R-curve is used. This measures the mean average precision of the model which is in our case (only one class sugar beet) the average precision. For other metrics that can be used in object detection, refer to the survey by **metrics_survey**.

3.4 Mobile Detection

For the detection directly in the app, Pytorch mobile **pytorch_mobile** can be used. For an overview of machine learning in mobile applications, refer to X. Dai, Spasić, Chapman, and Meyer 2020.

For the application, the pt file of the model can be used to detect objects directly in the application. This would have the advantage that the user can directly see whether a sugar beet is detected and adjust his angle or distance to the plant. Additionally possible would be to let the user adjust the bounding box if he thinks that the prediction is not precise enough.

4 Results

In this chapter, different approaches for training the YOLO model are presented. First, the results of training a simple large model will show a baseline model which is then improved by different settings. Afterwards, the results of training the small and medium model are shown. Finally, the three different model sizes are compared.

4.1 Large Network

The first experiments were made with only a small part of the data set being manually labeled. At first, the images of the TUM data that contain approximately one big sugar beet per image were labeled automatically with `0 0.5 0.5 1 1`. This means that the whole picture is labeled as containing exactly one sugar beet. We assumed that these images were already in the perfect format for our use case and the automatic labeling would be sufficient. The images containing multiple small plants are labeled manually and exactly. The model was trained with default hyperparameters and settings. This means that the number of epochs is 300, box loss gain of 0.05, class loss gain of 0.5, object loss gain of 1.0 and IoU threshold for training of 0.2. The data augmentation values can be found in table 4.1.

Hue	Sat	Val	Deg	Trans	Scale	Shear	Persp	UD	LR	Mos	Mix	CP
0.015	0.7	0.4	0.0	0.1	0.5	0.0	0.0	0.0	0.5	1.0	0.0	0.0

Table 4.1: Values for data augmentation. Hue, Saturation and Value are given in a fraction. Deg is degree of the rotation, Trans the translation (fraction), scale and shear (also in degree). UD and LR are the probabilities of the image to be flipped up-down (UD) or left-right (LR). Mosaic, mixup and copy paste are also given as probabilities.

All in all, small data augmentation is used. 85% of the data was used as training set and 15% as validation set. The different image types were divided equally into each set (training and validation).

For the training and validation set, the results were accurate. With a real number of

9110 sugar beets and a detected number of plants of 8162, this results in an accuracy of 89, 6%. Although for new, unseen images, the accuracy is very low. Here, the number of labeled sugar beets is 575 and only 68 are detected. An overall accuracy of 11, 8% results. In this experiment, only the number of detected and real sugar beets are observed, not the location or size of the bounding boxes. However, the results show that this model is not sufficient for detecting the boundaries of sugar beets. Two examples of detected objects can be seen in figure 4.1.

Figure 4.1: Examples of whole images being detected as one sugar beet. That was one of the problems that had to be faced.

You can see that in both cases, the whole image is detected as one plant. In the left example, the angle of the recording camera is very good with 90, in the right example which is taken from the Imagenet folder containing sugar beet images, the angle is not perfect. However, also here the whole images is labeled as sugar beet. the class probabilities are 0.95 in the left case and 0.89 in the right image.

Another problem of this model is the high false positive rate in case of other plants. In another test, 1271 images of all kinds of plants of Imagenet are tested. 922 labels of sugar beets were detected which means that about 70% are detected false positive. Two examples can be seen in figure 4.2.

You can see that also random plants are detected as sugar beets which should not be the case.

The following figure 4.3 shows the testing result of different training strategies and setting to mitigate this problem.

Testing the model with a test set consisting of the heterogeneous Partner data and 400 TUM images which contain multiple small plants and also larger ones, yields the following result which can be seen in figure 4.3a.

Figure 4.2: Examples of other plants also detected as sugar beets. That was another problem that had to be solved.

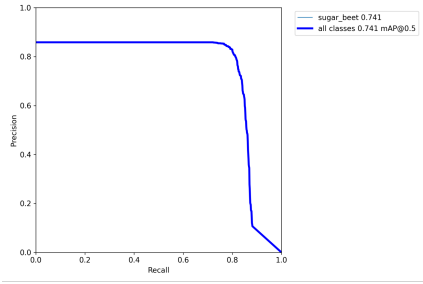
A perfect model would have an area under curve of 1.0. This one has 0.741 which still has room for improvement.

All in all, we encounter two main problems. The first one is that the bounding boxes of detected plants are just very inaccurate. Most of the time, the whole image is labeled as one sugar beet. The second problem is that the model is not robust. It also detects other plants as sugar beets and it can not distinguish between different kinds of plants.

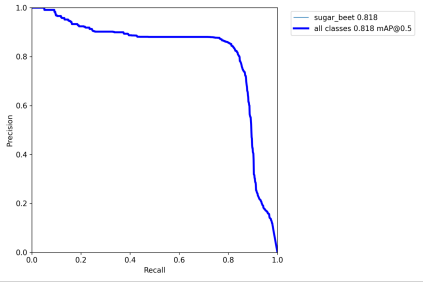
The first problem can be solved by labeling the data more accurate. By this, the model learns the exact boundaries of sugar beet plants and the predicted results get better. For the second problem, two different solutions exist. One is to add another class called other plant which is essentially everything else than sugar beets. The problem of this is that for example cars or persons are also detected as other plant which is also not intended. The second possible solution of this problem is to add so called background images. These are pictures which are not labeled with any object. Possible images therefore are other plants, persons or cars. At first, we added the so called other_plant class to get a better distinction between sugar beets and any different type of plant.

Although the predicted results are not too good by now, the model has already seen the structures of sugar beets. This can be used as pretrained model. In the following, four different training strategies are presented. The goal is to compare training the model from scratch and using the pretrained version. Additionally, the impact of including background images is investigated. The values of different hyperparameters for higher data augmentation can be seen in table 4.2.

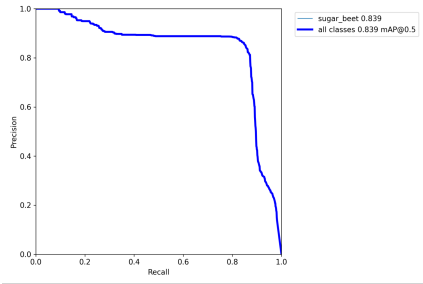
The first experiment is done from scratch without pretraining and lower data aug-



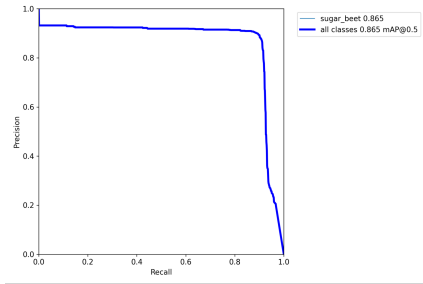
(a) P-R-curve for the first experiment. With mainly automatically labeled images, an average precision of only 0.741 can be achieved.



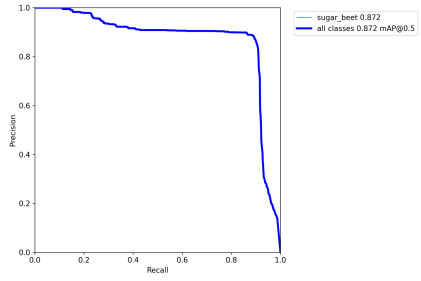
(b) P-R-curve for the second experiment. Now, the AUC has increased to 0.818 by adding manually labeled images.



(c) P-R-curve of third experiment. Now, higher data augmentation is used, leading to higher AUC of 0.839.



(d) P-R-curve of fourth experiment. Now, the training was done with two classes (also other_plant). The resulting AUC is 0.865.



(e) P-R-curve of last experiment. With only one class and higher data augmentation, an average precision of 0.872 can be achieved.

Figure 4.3: Comparison of testing P-R-curves of the different training experiments.

mentation. The dataset used is labeled completely by hand. The results of the tests can be seen in figure 4.3b.

With an AUC (area under curve) of 0.818, it has already better results compared to the first experiment. The reason is the more exact labeling.

Hue	Sat	Val	Deg	Trans	Scale	Shear	Persp	UD	LR	Mos	Mix	CP
0.015	0.7	0.4	0.0	0.1	0.9	0.0	0.0	0.0	0.5	1.0	0.1	0.1

Table 4.2: Values for higher data augmentation. Meanings of the variables are the same as in 4.1. Now, higher values are used compared to the first one.

In the next training, again no pretrained model was used. Instead, the parameters were adjusted to use higher data augmentation. The results of the testing can be seen in figure 4.3c.

You can see that an improvement can be achieved with an AUC of 0.839 (compare 0.818 from before). We can conclude from this that data augmentation has quiet a high impact on the accuracy.

In the next experiment, the pretrained model from before is used. The data augmentation is again the higher one and here, the images of other plants from Imagenet are labeled with other_plant. Testing results can be seen in figure 4.3d.

With an AUC of 0.865, the prediction accuracy again has improved.

In the next experiment, the difference to the previous one is that the images from Imagenet are not labeled at all. As already mentioned, the problem of the inexact labeling of the Imagenet pictures can sometimes lead to unintended predictions. The testing results of this experiment can be seen in figure 4.3e.

This model has the best AUC with 0.872.

The accuracy measured in AUC of the P-R-curve, improved with the different training settings. Figure 4.4 visualizes this again. There, the average precisions of each model are compared.

It can be observed that the best improvement is made with the exact labeled data. In general, the precision gets better with different training settings and techniques, but the difference between them decreases. This might be a saturation where no big improvements can be made anymore.

Now with these improvements, the new predictions are more accurate and also, the problem of the second class other_plant is no longer existent. Examples for current predictions can be seen in figure 4.5.

These predictions were made in one of the drone videos. You can see that now, the bounding boxes are just around one plant each. This is the behavior that was expected of the object detection algorithm. Previously, the whole image or one row of plants

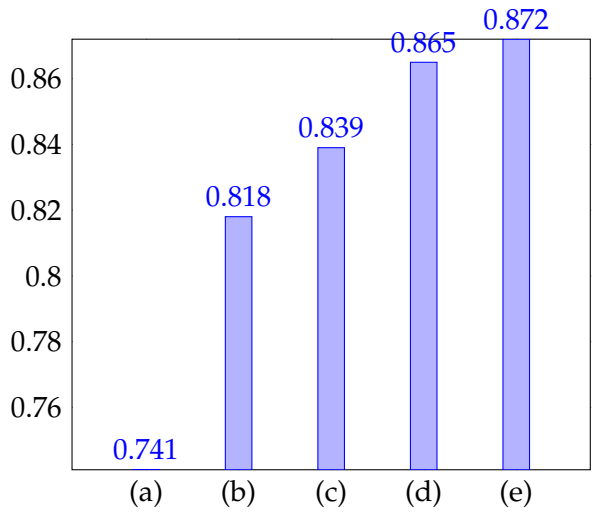


Figure 4.4: Direct comparison of the average precision of the different training experiments. At first, the AUC increases fast. A saturation can be seen at the last experiments.

were labeled as sugar beets. Other problems like detecting the image as other_plant if no sugar beet was detected is also improved.

4.2 Small and Medium Network

As already mentioned, the inference time of the large model don't really make it possible to apply it in the application in-place. This is the reason why also the small and medium sized model is trained. Already some behavior could be seen in experiments with the large model such as the improvements of including exactly labeled images, background data with no labels and using higher data augmentation. The results of the small and medium sized models are presented in the following.

First experiments are made with the models which are pretrained using the COCO dataset. Data augmentation was low and also all other hyperparameters were chosen to be the default ones as this was just a first experiment to compare the accuracy of the two types of models. Figure 4.6 depicts the testing results.

Both R-P-curves look quite similar. However, the left one (small model) has a bit smaller precision in regions of small recall. In contrast, with higher recall, the small model has higher precision. All in all, both models have similar AUC with 0.806 (small) and 0.805 (medium). This small difference and the fact that the small model has much

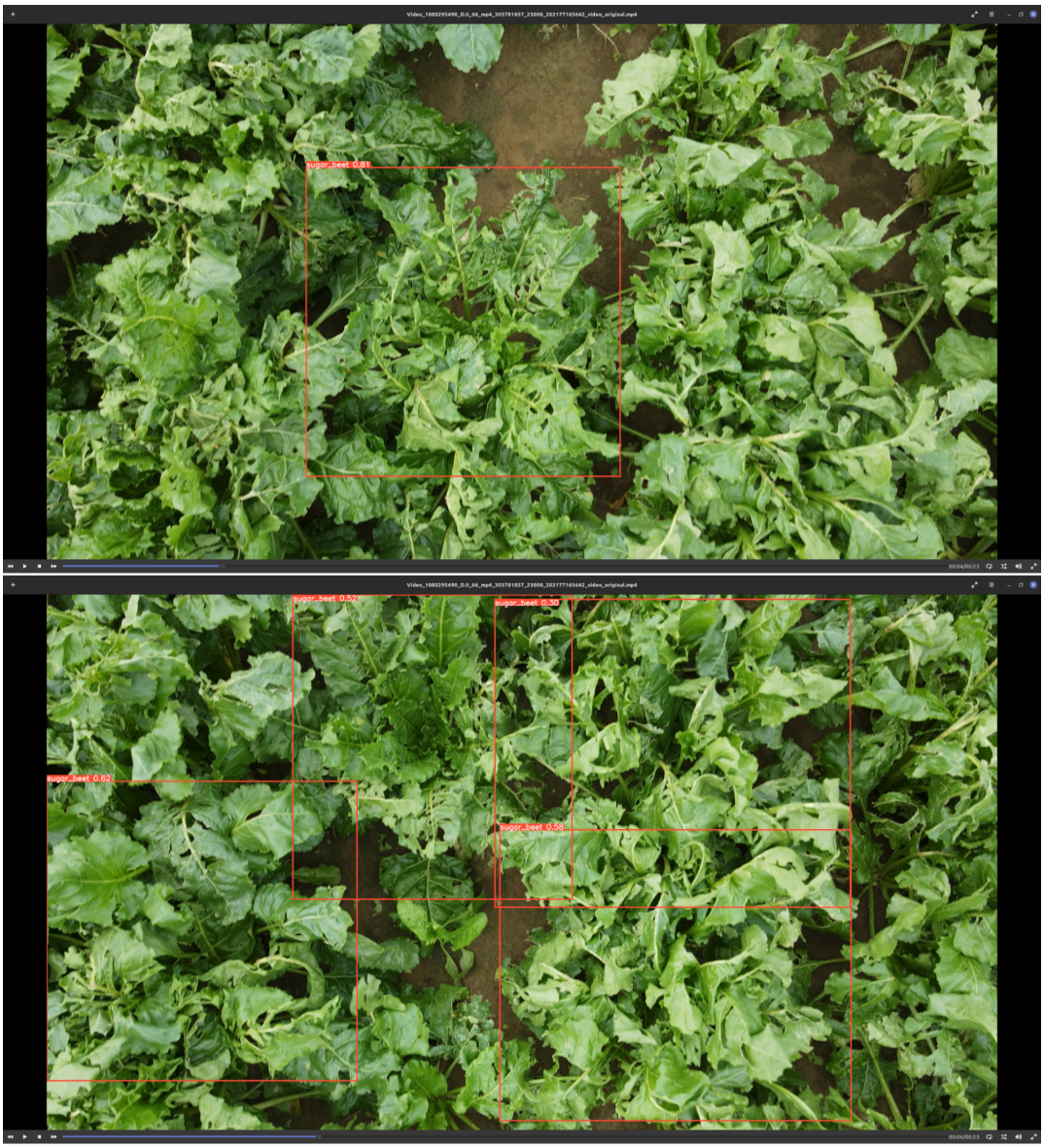


Figure 4.5: Predictions of sugar beets in screenshots of drone videos. Now, single plants are detected with high accuracy of bounding boxes.

faster inference times lead to the decision to not further train the medium model and focus on the small one.

With the knowledge of the large model to include background images and use higher

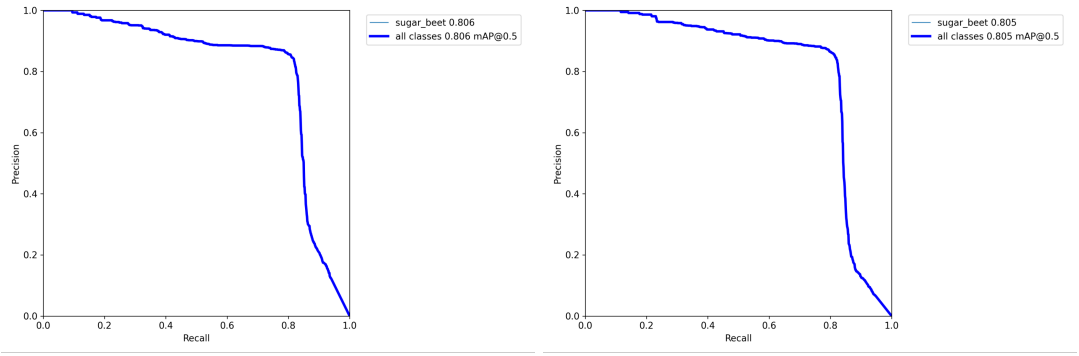


Figure 4.6: Comparison of small and medium sized model. Left one (small model) has AUC of 0.806 and right one (medium) AUC of 0.805.

data augmentation, the setting of the small model can also be adjusted accordingly. With these hyperparameters and a pretrained model on the sugar beet images labeled as whole picture as one plant, the following results of figure 4.7 can be observed.

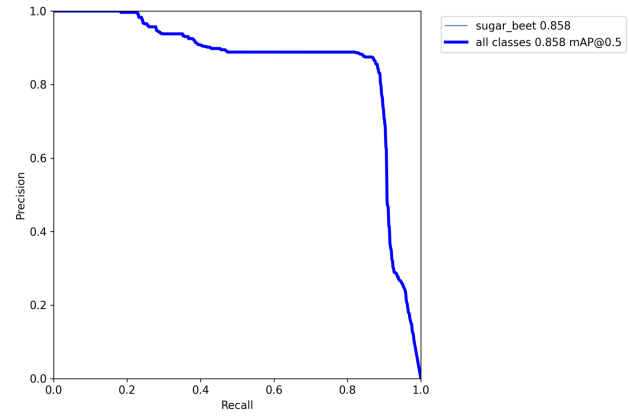


Figure 4.7: Result of the better trained small model. Now, the average precision is 0.858.

It can be seen that the accuracy has further improved compared to the first experiments. Now, with small recall (nearly up to 0.3), a precision of about 100% can be achieved. The curve now drops much faster, but later. The overall AUC is increased by approximately 6% and lays at 0.858.

4.3 Comparison and Discussion

In this section, a comparison is made and a general discussion about the models introduced and the object detection in general is presented.

Now the first aspect is the type of model. YOLO is in general very efficient due to the architecture and the idea of processing the image once (ÿou only look once¨). However, in the real time application of detecting the plants in the field, big differences between the large and small architectures can be observed. While the large model takes nearly half a second per image, the small one can predict the labels of a videostream nearly 10 times per second. This makes it very hard to apply the model in-place in the mobile application as it would not be very user-friendly.

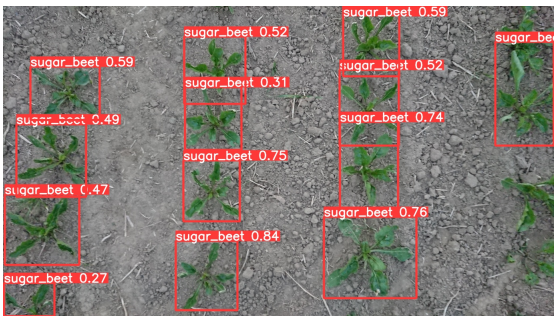
One next aspect is the accuracy of predicted labels. While the large model has an AUC of 87.2%, the small model has an AUC of 85.8%. As expected, the larger models predicts the bounding boxes of sugar beets in fields more accurately. But the difference to the small model is very low. We also have to keep in mind that about 10000 images of sugar beets were labeled manually. Of course, this was done as exactly as possible, but in many cases, the borders of sugar beets were not clearly visible at first or even second glance leading to small errors in the labeling. Especially in images of larger sugar beet plants, the bounding box labeling by hand is very hard because of the overlapping leafs. Especially with this background, the small difference of accuracy between large and small model is not too important.

All in all, this leads to the decision of implementing the small model directly in the application. It has two advantages. The first is that the user can directly see if a plant is detected and immediately adjust the camera position and angle. Another possibility is the implementation of adjusting the bounding box so that a different part of the image is cropped if the user thinks that the prediction is not correct enough. This leads to more flexibility of the application.

The following images show results of detected sugar beets. They can be seen in figure 4.8 and some properties of the model can be followed. The first two examples (figure 4.8a and 4.8b) show easier cases. Image 4.8a is already very similar to the standardized format. The camera angle is 90 and the plant is centered. The second one 4.8b depicts small plants where the boundaries do not overlap and the soil has a completely different color than the plants which makes the detection easier. Example 4.8c shows an edge case. It shows two half plants at the top and right border of the image. They have high damage and are not centered on the image. However, one sugar beet plant is detected at the top of the picture because the structure of the plant was learned. In this case, the live application in the app helps the user to find the right position of the camera. He would have to move it to the top to capture the whole

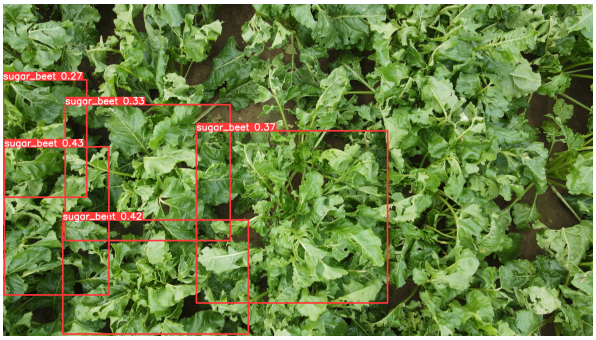
plant. In figures 4.8d and 4.8e, many larger plants are depicted. In the first one (4.8d), 5 plants are detected with low confidence because it is very hard to find the borders because of the overlaps. However also in this case, the user of the app can use this information to hold the camera closer to the field. In the last image 4.8e, two plants are detected with higher confidence. Especially the upper one can be detected very well because the boundaries can be seen much better than all other sugar beets in this picture. Additionally, the source of the leafs can be seen here which makes it easier to center the bounding box on the plant. This is a tendency that can be observed for the whole model. If this is the case, the bounding box is much better than in cases of leafs not being structured.

(a) Example of an image with only one plant. The image is already similar to the standardized format.



(b) Example of many small plants. The model is able to predict the boundaries very well because of nearly no overlap between the sugar beets.

(c) Example of an image of plants with high damage. The picture was not taken optimally (angle, not centered on one plant). The model detects one plant at the top of the image.



(d) Example of many larger plants. The model detects 5 sugar beet plants with relatively low confidence because the boundaries can not be seen very easily.

(e) Example of many larger plants. The model detects 2 sugar beet plants. The upper one with higher confidence because the boundaries are clearer there.

Figure 4.8: Results of the object detection algorithm. Different scenarios are tested.

5 Conclusion and Outlook

All in all, in this project a preprocessing algorithm was built as part of a complete machine learning pipeline to predict the hail damage of sugar beets. With this step, the images taken in the mobile application are cropped in a standardized way to improve the damage prediction value. By only training the images with sugar beets in an 90 angle with the plants centered, the general accuracy of the damage prediction can be improved. Also, with the live predictions in the mobile application, the user can directly interact with the bounding boxes and see whether his camera position and angle is already good or if he still needs to adjust it.

By now, this pipeline can only handle sugar beet plants. Of course, this can be further improved by adding other plants if needed. The only requirement is much data of a new plant to add it. For example, a new class of this plant can be added to the detection algorithm to also differentiate the type of plant. Based on this decision, the image could e.g. be sent to a different model trained on the specific plant.

Other ideas to improve the models would be to use newer versions of YOLO such as YOLOv6/ YOLOv7. This is an emerging field with very fast development of new feature and more robust models. One such thing is pose detection which could be an interesting new feature. Instead of just predicting bounding boxes, also the concrete structure of for example the leafs would be learned and predicted in new images.

Bibliography

blx@hook@bibinit

Bochkovskiy, A., C.-Y. Wang, and H.-Y. M. Liao (2020). "YOLOv4: Optimal Speed and Accuracy of Object Detection." In: *CoRR* abs/2004.10934. arXiv: 2004.10934.

Dai, J., Y. Li, K. He, and J. Sun (2016). "R-FCN: Object Detection via Region-based Fully Convolutional Networks." In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett. Vol. 29. Curran Associates, Inc.

Dai, X., I. Spasić, S. Chapman, and B. Meyer (2020). "The State of the Art in Implementing Machine Learning for Mobile Apps: A Survey." In: *2020 SoutheastCon*, pp. 1–8. DOI: 10.1109/SoutheastCon44009.2020.9249652.

David, E., M. Serouart, D. Smith, S. Madec, K. Velumani, S. Liu, X. Wang, F. Pinto, S. Shafiee, I. S. Tahir, et al. (2021). "Global wheat head detection 2021: an improved dataset for benchmarking wheat head detection methods." In: *Plant Phenomics* 2021.

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009). "Imagenet: A large-scale hierarchical image database." In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, pp. 248–255.

Girshick, R. (Dec. 2015). "Fast R-CNN." In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Girshick, R. B., J. Donahue, T. Darrell, and J. Malik (2013). "Rich feature hierarchies for accurate object detection and semantic segmentation." In: *CoRR* abs/1311.2524. arXiv: 1311.2524.

Han, F. and J. Li (2022). "Wheat Heads Detection via Yolov5 with Weighted Coordinate Attention." In: *2022 7th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)*, pp. 300–306. DOI: 10.1109/ICCCBDA55098.2022.9778925.

He, K., X. Zhang, S. Ren, and J. Sun (2015). "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.9, pp. 1904–1916. DOI: 10.1109/TPAMI.2015.2389824.

Huang, J., V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy (July 2017). "Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Jocher, G. (2022a). *YOLOv5 Github repository*. `https://github.com/ultralytics/yolov5`. Accessed: 2022-06-22. DOI: `https://doi.org/10.5281/zenodo.3908559`.

— (2022b). *YOLOv5 in PyTorch Hub*. Accessed: 2022-08-02. DOI: `10.5281/zenodo.6222936`.

— (2022c). *YOLOv5 release version 6.1*. `https://github.com/ultralytics/yolov5/releases`. Accessed: 2022-08-02.

Kragh, M., R. N. Jørgensen, and H. Pedersen (2015). "Object Detection and Terrain Classification in Agricultural Fields Using 3D Lidar Data." In: *Computer Vision Systems*. Ed. by L. Nalpantidis, V. Krüger, J.-O. Eklundh, and A. Gasteratos. Cham: Springer International Publishing, pp. 188–197. ISBN: 978-3-319-20904-3.

LabelImg (n.d.). `https://github.com/tzutalin/labelImg`. Git code (2015).

Lin, T.-Y., M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick (2014). "Microsoft COCO: Common Objects in Context." In: *CoRR* abs/1405.0312. arXiv: `1405.0312`.

Liu, S., L. Qi, H. Qin, J. Shi, and J. Jia (June 2018). "Path Aggregation Network for Instance Segmentation." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Liu, W., D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, and A. C. Berg (2015). "SSD: Single Shot MultiBox Detector." In: *CoRR* abs/1512.02325. arXiv: `1512.02325`.