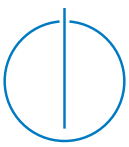
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Thesis type (Bachelor's Thesis in Informatics, Master's Thesis in Robotics, . . .)

Thesis title

Author



DEPARTMENT OF INFORMATICS

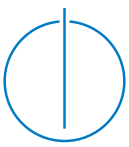
TECHNISCHE UNIVERSITÄT MÜNCHEN

Thesis type (Bachelor's Thesis in Informatics, Master's Thesis in Robotics, . . .)

Thesis title

Titel der Abschlussarbeit

Author:	Author
Supervisor:	Supervisor
Advisor:	Advisor
Submission Date:	Submission date



Abstract

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.

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 [1]. Since then, two improved versions were published by the original author. These architectures were called YOLOv2 [2] and YOLOv3 [3]. After that, other authors published the next Version YOLOv4 [4] 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 [5] 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 in [1], the network is designed to find objects and the corresponding boxes in a global manner meaning that the image is processed as 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 $Pr(Object) * IOU_{pred}^{truth}$. 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 the class probabilities and box confidence predictions are multiplied resulting in the class-specific confidence scores for each box: $Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth}$.

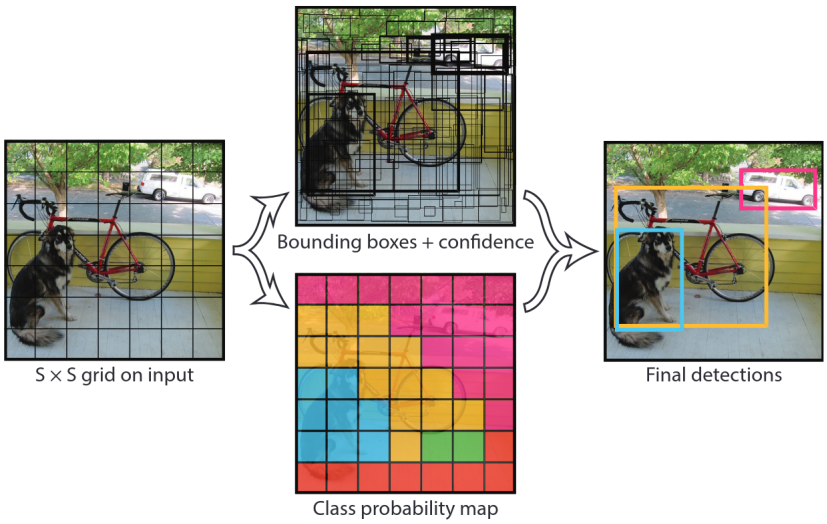


Figure 2.1: Description of idea of YOLO. Taken from [1]

All in all, a tensor of size $S \times S \times (B * 5 + C)$ results. Figure 2.1 depicts the idea of predicting bounding boxes with corresponding classes.

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 layer 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 plays an important role.

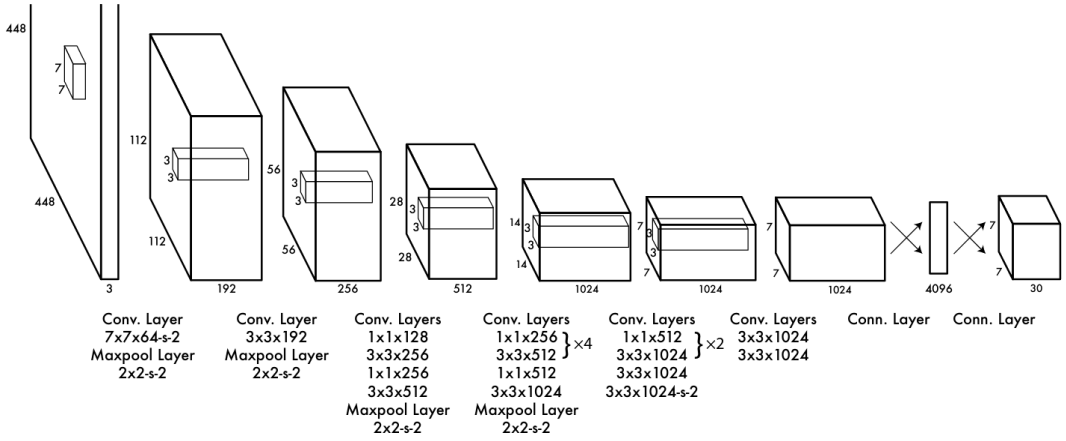


Figure 2.2: Description of network of YOLO. Taken from [1]

2.2 Improvements in latest versions

The first architecture was improved incrementally in different versions. The original authors were involved until version 3 ([1]–[3]). After that, other authors published improvements as YOLOv4 [4]. There is no publication to YOLOv5 yet, although there is already an implementation [6].

2.2.1 YOLOv2

With the first improvement which is published in [2], the problem of the localization errors of the boundaries of objects are 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 therefore introduce a new network called darknet-19 as the basis for YOLOv2. It has many advantages compared tp the previous one such as high accuracy and faster times.

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.

2.2.2 YOLOv3

The main difference to YOLOv2 is that in the new version presented in [3], 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 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.

2.2.3 YOLOv4

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

YOLOv4 [4] also consists of such parts. It is a One-Stage Detector and therefore only has dense prediction. As a backbone, CSPDarknet53 [7] is used. The neck consists of SPP (spatial pyramid pooling) [8] and PAN (path aggregation network for instance segmentation) [9]. Finally, the head (dense prediction) is the YOLOv3 network [3]. In

	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: Description of network of YOLOv3. Taken from [3]

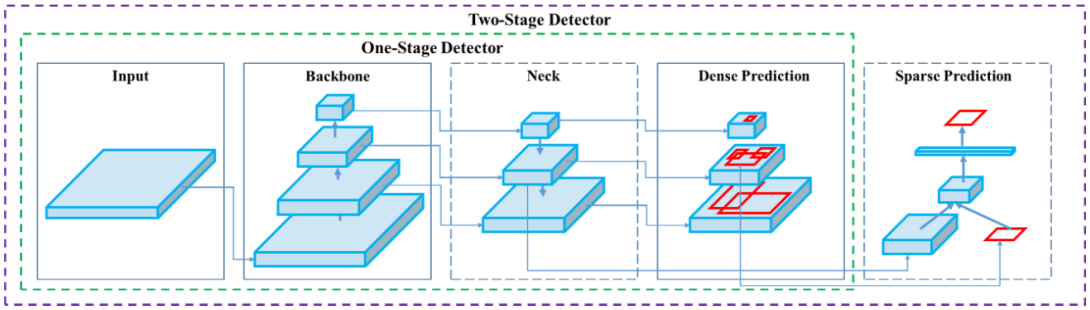


Figure 2.4: Description of network of YOLOv4. Taken from [4]

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.

2.2.4 YOLOv5

The current version YOLOv5 [6] 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.

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. 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.

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 described in the following, a nvidia gpu with 24267 MiB memory was used.

In the following, the given dataset, the training of the network and the evaluation of results 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 [10] 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 two different types. 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 can be seen in figure 3.1.

Figure 3.1: Image types

The left one 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 right example. 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 (right example). The numbers of images for each type are 9964 for the type of the right example and 123 for the left one. In practice, most images will be of bigger plants because the damage prediction is more important in those cases.

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 [11] which provides a graphical user interface to draw the rectangles around the objects.

As second source, Imagenet was used. About 3500 images with other plants and flowers were labeled automatically as only one plant other than a sugar beet. Also here, 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. Further experiments also showed that false positive detections of sugar beets were reduced drastically by including automatically labeled Imagenet pictures. As a third type of images, background images with completely different things on it such as persons, animals or food were also used. Imagenet provides a wide range of data for this use case. These images were just not labeled.

3.2 Training

3.3 Evaluation

4 Results

5 Conclusion and Outlook

Bibliography

[1] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. arXiv: 1506.02640.

[2] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016. arXiv: 1612.08242.

[3] ——, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. arXiv: 1804.02767.

[4] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *CoRR*, vol. abs/2004.10934, 2020. arXiv: 2004.10934.

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

[6] *Yolov5 github repository*, https://github.com/ultralytics/yolov5, Accessed: 2022-06-22. DOI: https://doi.org/10.5281/zenodo.3908559.

[7] C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, "Cspnet: A new backbone that can enhance learning capability of cnn," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2020.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015. DOI: 10.1109/TPAMI.2015.2389824.

[9] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2018.

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

[11] *Labelimg*, https://github.com/tzutalin/labelImg, Git code (2015).