Deep Learning - EnBeeMo

Sissy Friedrich, Maximilian Sandner, Markus Schmidt, Michael Sieber
Munich University of Applied Sciences
Munich, Germany
{sissy.friedrich, sandner, schmid78, sieber0}@hm.edu

***Abstract*—A big and widely discussed agricultural topic is the preservation of the honey bee, as the need for environmental protection has been increasingly gaining in importance. Beekeepers want to analyze the behavior and change of their bee colonies. However, it is detrimental if such analysis is done directly with interventions in the bees' natural habitat. Therefore, we propose a deep learning method for detecting bees in images allowing further research to be conducted on behavior and quantitative analysis. The results prove the recognition of bees in an image sequence and thus contributing to the goal of non-interfering research.**

I. Introduction

The debate about the death of the honey bee has recently become topical again. In 2010, the University in Hohenheim started a german bee monitoring project [1] to understand the periodically high winter losses of honey bee colonies.

The beekeepers' state of knowledge is only subjectively recorded information about the bee population and correlations to environmental factors, such as temperature, humidity, and pesticides. Their vision is to be able to objectively relate the bee population to other environmental factors in order to derive appropriate measures. Knowing the change in colony strength is important for beekeepers to be able to draw conclusions about certain location factors and events that may influence the development of the respective bee colony.

The goal of the EnBeeMo (Environmental Bee Monitoring) project is the detection, tracking, and counting of bees without interfering with the bees' natural habitat.

Based on the need to determine the colony strength, the first step emerges the research question: "Can we develop a pareto-optimal algorithm for the detection of bees?". In this regard, the contribution of this work includes (i) the comparison of available algorithms dealing with object detection in general, (ii) the implementation of a neural network classifier with optimized hyper-parameters to detect bees, and (iii) the evaluation based on metrics like accuracy, recall, precision, and the comparison of the different models.

The paper is organized as follows. Section II gives an overview over related work on the topic concerning object detection algorithms. A short introduction of the development system and environment is given in Section III. The processing of the given data is the topic of Section IV. Section V introduces the neural network with its architecture, hyper-parameters, and training phase. The approach is evaluated in Section VI, before a conclusion is drawn in Section VII.

II. Related Work

Some research has already been applied in the area of detecting and counting bees. Respondek and Westwanska [2] present an approach for segmentation and counting bees in colored images. They use a U-Net CNN for counting objects of interest (OOI).

Tiwari [3] implements a deep learning algorithm for the analysis of bee traffic in real time. If a motion is detected in an image sequence, the corresponding image is sent to the neural network, which classifies the input according to the classes "bee" or "no bee".

The state of the art on object detection in images is represented in [4]. Jiao et al. give an overview of object detection algorithms in deep learning, such as YOLO, SSD, ImageNet, and further implementations. The most common algorithms are briefly introduced by procedural analysis and main detector types.

In this work, a Faster R-CNN (Regions with Convolutional Neural Network features) is used for the bee detection, introduced in [5]. The Faster R-CNN is an improved version of Fast R-CNN [6], which is a progression of R-CNN [7]. This object detection system contains two main modules: A Feature Network combined with a Region Proposal Network (RPN), that proposes regions, and a Fast R-CNN detector. A pre-trained CNN is used as a Feature Network, representing a powerful backbone, to extract a convolutional feature map. The RPN takes this feature extractor as input and outputs a set of boxes as object proposals by checking whether the corresponding anchor boxes actually contain objects (Regions of Interests, ROIs). Anchor boxes are reference bounding boxes of different sizes and ratios that are placed over the entire image, using selected pixels as anchors. An example is depicted in Fig. 1, showing 9 boxes for one anchor. The next step, called ROI Pooling, extracts the features of the ROI and feeds them into the final R-CNN module, which contains a box regression layer and a box classification layer. The backbone can be extended by using a Feature Pyramid Network (FPN) [8], which gives an advantage when detecting small objects by extracting ROI features from different levels of the pyramid.

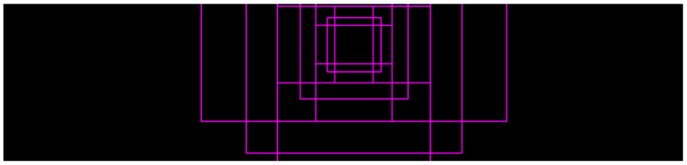


Fig. 1 Anchor boxes with different sizes and shapes.

III. System and Environment

This project is implemented on AWS Sagemaker with the kernel PyTorch 1.6 GPU in Python 3.6. The data set is loaded from a S3 bucket with boto3 and trained on an instance with a 16 GB GPU (ml.g4dn.xlarge).

IV. The Data Set

An infrared camera records videos of the bees entering and leaving a hive. The predefined data set consists of 2000 images. Fig. 2 shows an example image containing 7 bees. The ground-truth labels are created by AWS, representing bounding boxes enclosing the bees. Each image has a size of 1613×371 with 3 channels.

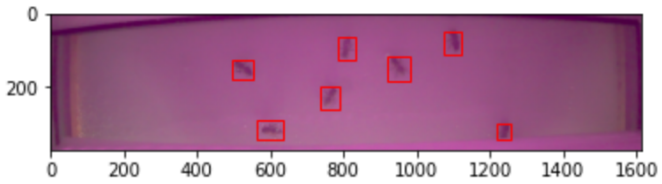


Fig. 2 Bounding boxes around the bees as a ground-truth data.

The data set contains one blank image without any bee. This image is removed from the data set to avoid problems with the neural networks, since they require at least one label in every image during the training phase.

A. Data Preparation

The images consist of a infrared data, in which bees appear black. The data is reduced to gray scale, as depicted in Fig. 3, since color information reveals no additional details. The images are also normalized to a range of $[0, 1]$ when processed by Faster R-CNN by taking the standard deviation of std $= 0.5$ and mean $= 0.5$.

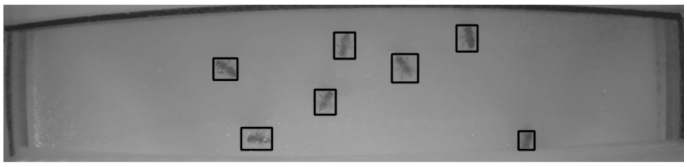


Fig. 3 Data reduced to gray scale.

B. Data Augmentation

For achieving better results, the images are augmented using Albumentations [9]. Fig. 4 shows two examples of augmentations carried out on Fig. 3: (a) vertical flip, and (b) horizontal flip.

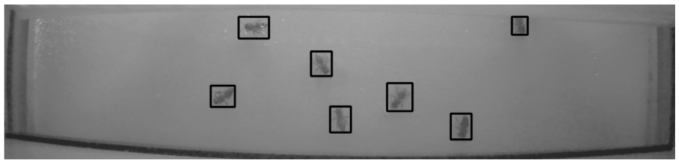
This increases the amount of the training data and enforces generalization. We only use a subset of augmentations, since the available AWS-instance types do not provide enough memory to handle more augmented data.

V. The Proposed Detection Algorithm

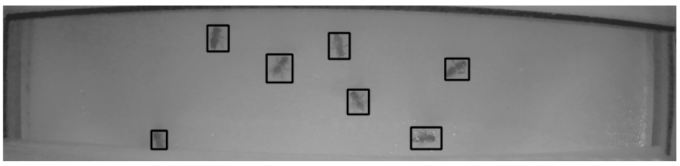
In this section, the proposed detection algorithm is explained in detail.

A. Dataloader

The Dataloader is specific for PyTorch models and wraps a torch tensor around each object to be fed into the model. Therefore, some functions of the DataSet have to be overwritten, e.g.: The `__init__()` function initializes the connection to the resources in the S3 bucket. The received image names are stored in a list, the manifest file with the annotations of the data (information about the bounding boxes) is loaded and the annotations are stored separately in a dictionary with the image name as a key. The `__getitem__(self, idx)` function receives an index *idx* and returns a tuple consisting of the image as a torch tensor and the target as a dictionary, which contains the coordinates of all bounding boxes, the corresponding label, and the image_id.



(a) Vertical flip.



(b) Horizontal flip.

Fig. 4 Applied augmentations using Albumentations [9].

B. Architecture of the Detection Algorithm

The first component of a Faster R-CNN, as described in Sec. II, is a Feature Network. Since the focus is more on the optimization of the hyper-parameters rather than building a complete network from scratch including all components, the provided networks from PyTorch are used. To build the architecture, we use the GitHub repository of the open-source library Torchvision [10], which is part of the PyTorch framework. Different backbones are examined to determine their impact on the evaluated metrics, as discussed in the following.

1) VGG16: The first approach uses a VGG16 backbone [11]. The default VGG16 network expects an image with 3 channels. Since the proposed approach uses gray scaled images, it is necessary to alter the first layer. The output of the VGG16 is downsized from 1000 to 512 feature maps to fit the RPN input size. Therefore, the last two layers are removed from the network.

2) ResNet: The second approach uses a ResNet backbone, introduced by He et al. [12]. Different ResNet architectures are tested, such as ResNet50, ResNet101, and ResNet152, to determine whether a bigger architecture of the Feature Network results in a higher accuracy. As well as the previously mentioned VGG16, the layers need to be adapted to fit the gray scale input and the consecutive RPN provided by the Faster R-CNN. Therefore, the first layer is altered and the last two layers are removed to achieve an output of 512 feature maps.

3) Pre-trained Faster R-CNN: As a third approach, a pre-trained Faster R-CNN from [10] without adaption is used to compare the output to the other Faster R-CNNs with individualized backbones. This model uses an additional FPN in the backbone after the ResNet50.

C. Training the Faster R-CNN

The models are trained on 75% of the available images, which are in total 1500 images without augmentation, as explained in Section IV-B. To compare the performance, the models are at first trained on the original data set. Training is carried out with a batch size of 1 and different numbers of epochs. As optimizer, SGD and Adam are used and compared to each other. For hyper-parameter tuning, different values of learning rate, weight decay, and momentum are tested. As a starting point, the learning rate $\alpha = 0.005$, the weight decay factor 0.0005, and the momentum 0.9 is used by the SGD optimizer. These hyper-parameters are inspired by [13].

VI. Evaluation

In this section, the compared architectures are evaluated concerning precision, recall, and accuracy. First, the evaluation procedure is explained, before the models are compared.

A. Metric-based Evaluation

This section describes the evaluation of a single picture.

1) Generating IOU Matrix: At first, a matrix $m \times n$ is generated, where m is the number of ground-truth boxes and n the number of predicted boxes. For each ground-truth box, the corresponding area (A_{gt}) is calculated and the corner coordinates are extracted and compared to each predicted bounding box with the area (A_{pred}) and the corresponding corner coordinates. Thus, the Intersection of Union (IOU) is computed between each combination of ground-truth and predicted boxes and denoted accordingly in the matrix. Eq. 1 shows how the IOU is calculated in a mathematical way w.r.t. the intersection area ($A_{\text{gt} \cap \text{pred}}$).

$$\text{IOU} = \frac{A_{\text{gt} \cap \text{pred}}}{A_{gt} + A_{\text{pred}} - A_{\text{gt} \cap \text{pred}}} \tag{1}$$

2) Mapping between ground-truth and prediction: Using the IOU matrix, the ground-truth boxes are mapped to predicted boxes. Mapping means that the indices of the mapped ground-truth box and predicted box are stored together with the IOU. The column and row of the corresponding boxes in the IOU matrix are then set to 0 to prevent multiple mapping. In the first step, all IOUs below a specific threshold are disregarded to avoid incorrect mapping in the case of a minimal overlap between two boxes. Afterwards, all ground-truth boxes with an unambiguous mapping to a predicted box are mapped. If there are no such mappings left, the box-combination with the maximum IOU is selected and mapped. Mapping one pair of boxes can enable other ground-truth boxes to have only one matching, which again is unambiguous. This procedure results in a list of mappings between ground-truth boxes and predicted boxes with the corresponding IOUs.

3) Metrics calculation: Finally, the common metrics, accuracy (acc), precision (prec), and recall (rec), can be calculated with the following equations:

$$\begin{aligned} \text{acc} = 1 - \text{errorrate} &= \frac{TP + TN}{TP + TN + FN + FP} \\ \text{prec} &= \frac{TP}{TP + FP} \\ \text{rec} &= \frac{TP}{TP + FN} \end{aligned} \tag{2}$$

For the true positives (TP), the length of the mapping list can be used, since this contains the correct predicted boxes. The true negatives (TN) can be set to 0, since the model detects only bees and not the explicit background. The false positives (FP) are all boxes the model predicted but where no corresponding ground-truth boxes were mapped. Thus, it is calculated as the difference between the count of predicted boxes and count of mappings. The false negatives (FN) are all ground-truth boxes, where no corresponding prediction box exists. This can be calculated as the difference between the count of ground-truth boxes and the count of mappings.

B. Evaluation of the Models

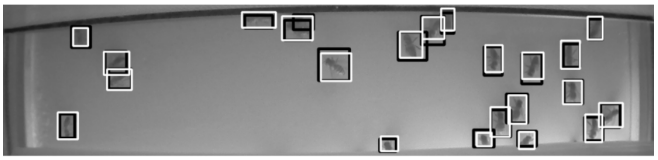
For the evaluation of the different models, the average for each metric in Eq. 2 is calculated over all test images. Every combination of backbone and different hyper-parameter combinations were trained and tested. Table I shows the evaluated metrics and the time required for predicting one image for the best hyper-parameter combination of each backbone.

backbone	acc	prec	rec	t[ms]
VGG16	0.1886	0.2013	0.7815	**57**
ResNet50	0.6488	0.7070	0.8710	66
ResNet101	0.4663	0.5202	0.7696	100
ResNet152	0.4117	0.4592	0.7618	130
Pre-trained	**0.8101**	**0.8502**	**0.9353**	99

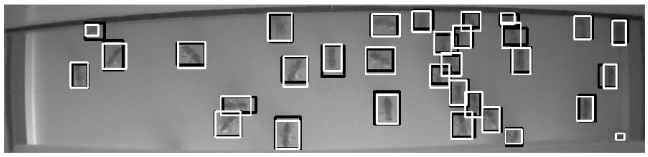
TABLE I Evaluated metrics for the best hyper-parameters of each backbone and the measured time required for detecting bees in one image.

The Faster R-CNN with the VGG16 backbone is trained for 20 epochs with SGD optimizer with weight decay 0.0005, momentum 0.9, and learning rate $\alpha = 0.005$. It achieves the lowest accuracy, precision, and recall, which is assumed to be due to the small architecture of the backbone, but has the fastest time for detecting bees in one image, as seen in Table I. The Faster R-CNN with the ResNet50 backbone has a SGD optimizer with weight decay 0.0005, momentum of 0.9, and learning rate $\alpha = 0.005$ and was trained for 40 epochs. Using a ResNet101 as a backbone, the best result is achieved with SGD optimizer, weight decay of 0.0005, learning rate $\alpha = 0.002$, and 20 epochs. The ResNet152 reaches the best results with weight decay 0.0005, momentum 0.9, learning rate $\alpha = 0.002$, and 20 epochs. It is noticeable that the time for detecting bees in one image increases proportionally to the size of the backbone. The pre-trained Faster R-CNN has a SGD optimizer with weight decay 0.0002, momentum 0.9, and learning rate $\alpha = 0.005$. The model was trained for 20 epochs to fit to the bee data set and has the best results concerning the calculated metrics, which is probably due to the FPN and the pre-trained backbone.

Fig. 5 shows two gray scale test images with bounding boxes, where the black boxes represent the ground-truth and the white boxes the predicted boxes. It is noticeable that image (b), representing the predictions of the pre-trained Faster R-CNN, is very precise and detects every bee in the image. Image (a), representing the predictions of the Faster R-CNN with ResNet50 as a backbone, depicts partially multiple boxes for some bees. This model thus predicts more FP, resulting in a lower precision and accuracy.



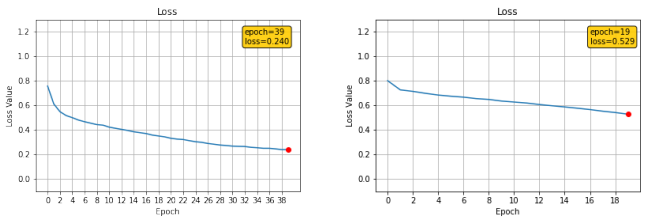
(a) Output of Faster R-CNN with ResNet50 backbone.



(b) Output of pre-trained Faster R-CNN.

Fig. 5 Output of the models on a test image. White boxes represent ground-truth, black boxes represent predicted boxes.

Fig. 6 shows the loss functions during training of the two best models, the Faster R-CNN with a ResNet50 backbone and the pre-trained Faster R-CNN. Even though the pre-trained Faster R-CNN has a higher loss function during training, it achieves a better accuracy in testing.



(a) Faster R-CNN, ResNet50 backbone. **(b)** Pre-trained Faster R-CNN.

Fig. 6 Loss function during training the models. The red point annotates the minimum of the loss function.

C. Problem with Ground-truth Data

The evaluated metrics, especially the precision, are not always satisfactory, due to partially incomplete ground-truth data. Fig. 7 shows an example image. It is noticeable that the ground-truth data covers only 19 bees out of more than 30 bees in the image. This results in a higher rate of FP predictions and thus influences directly the metrics precision and accuracy. In addition to the influence on the evaluation metrics, the ground-truth data also has a direct influence on the learning behavior of the models.

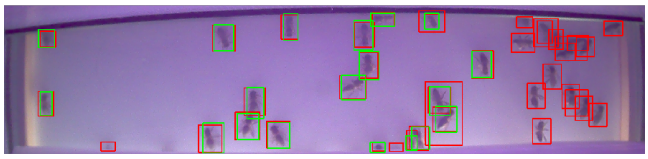


Fig. 7 The detected boxes (red) and the ground-truth data (green).

VII. Discussion & Outlook

The proposed models show that in general, it is possible to detect bees in an image. The accuracy, precision, and recall depend on the backbone and the hyper-parameters. Further optimization of the hyper-parameters and more augmentation of the images, such as inserting blur or changing the brightness and contrast, could improve the models. However, this was not possible within the scope of the course with the infrastructure provided with AWS, as only small instances with limited resources were enabled. Since augmentation increases the size of the training data set, this causes the memory to overload and the kernel to abort if the instances are too small. At first we tried to implement the whole Faster R-CNN from scratch, which lead to an enormous effort but better understanding of the architecture. Anand et al. [14] prove that deep learning experts are better than deep learning beginners concerning their intuition in using hyper-parameters or pre-processing methods. Since this course is an introduction into the bright field of deep learning, we consider ourselfes as beginners and lack of a good intuition concerning the right hyper-parameters for now.

Work Distribution

Teamwork: The team members met virtually, discussed the assignment, programmed together, and edited the summary.

References

[1] E. Genersch, W. von der Ohe, H. Kaatz, A. Schroeder, C. Otten, R. Büchler, S. Berg, W. Ritter, W. Mühlen, S. Gisder, M. Meixner, G. Liebig, and P. Rosenkranz, "The german bee monitoring project: a long term study to understand periodically high winter losses of honey bee colonies," *Apidologie*, vol. 41, no. 3, pp. 332–352, May 2010. [Online]. Available: https://doi.org/10.1051/apido/2010014

[2] J. Respondek and W. Westwańska, "Counting instances of objects specified by vague locations using neural networks on example of honey bees," 09 2019, pp. 87–90.

[3] A. Tiwari, "A deep learning approach to recognizing bees in video analysis of bee traffic," Utah State University All Graduate Theses and Dissertations. 7076., 2018.

[4] L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, and R. Qu, "A survey of deep learning-based object detection," *IEEE Access*, vol. 7, pp. 128 837–128 868, 2019.

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, 06 2015.

[6] R. Girshick, "Fast r-cnn," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448.

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 11 2013.

[8] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 936–944.

[9] E. K. V. I. I. A. Buslaev, A. Parinov and A. A. Kalinin, "Albumentations: fast and flexible image augmentations," *ArXiv e-prints*, 2018.

[10] Pytorch, "torchvision." [Online]. Available: https://github.com/pytorch/vision/tree/master/torchvision/models/detection

[11] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," in *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, 2015, pp. 730–734.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[13] P. Pesti, "Pytorch starter - fasterrcnn train," May 2020. [Online]. Available: https://www.kaggle.com/pestipeti/pytorch-starter-fasterrcnn-train

[14] K. Anand, Z. Wang, M. Loog, and J. V. Gemert, "Black magic in deep learning: How human skill impacts network training," *ArXiv*, vol. abs/2008.05981, 2020.