Freie Universität Berlin
Department of Mathematics and Computer Science
Team BioRoboticsLab

Bachelor's Thesis in Computer Science

Detection and localization of bees located in honeycomb cells using convolutional neural networks

Sophie Zabel

Enrollment number: 4989633
s.zabel@fu-berlin.de
Supervisor: Prof. Dr. Tim Landgraf

31 October 2019

Abstract

Honey bees are of particular interest to research in computer assisted behavioral analysis because of their complex social behavior and the large number of individuals in a colony. An important part for such studies is the detection and localization of bees during their activities in the hive. This thesis specifically focuses on bees located in honeycomb cells.
A ground truth dataset was created by manually marking bees located inside cells in images of whole honeycombs. Detail images of the marked position and random, unmarked positions were respectively extracted as positive and negative examples. A convolutional neural network was designed and trained on this dataset to classify detail images on whether they show bees in cells or not. The quality of the training was evaluated and heatmaps were created to visualize prediction strength in the original images. Local maxima of the heatmaps were determined as the predicted positions of bees in honeycomb cells. Several modifications were made during the work to increase the precision of the network predictions.
Testing the classifier used on the test sets made it possible to achieve a detection rate of 1 while testing on detail images and a detection rate of about 0.6 while testing on images of an entire honeycomb.

Contents

1 Introduction

Honey bees fascinate with their complex and differentiated behavior, the organization of the colony through division of labor, and their extreme and notable exchange of behavior in different environmental conditions. Their behavior in the hive has been extensively studied because of its suitability for observation and manipulation of their environment.[10, 26]
One of the most significant behavior patterns is their organization of the hive into distinct castes.[28] According to Johnson, there are two classes of division of labor, division of labor «between queens and workers and amongst the workers»[10]. Among the worker bees there are four castes: the cell cleaners, the nurses, the middle-aged bees and the foragers. The belonging to a caste mainly depends on the age of a honey bee and the associated amount of juvenile hormone III.[28] Foragers are the only bees that are working outside the hive while the other castes do not leave it. Instead, they clean the cells (mainly done by cell cleaners), feed the brood (only done by nurses), store pollen in the cells and are responsible for the building and maintenance of the nest, which includes building and repairing cells (only done by middle-aged bees).[10, 25]
Their task repertoire grows with time and some tasks become less important. This means that by observing the colony one can only assign a caste to each bee if its full behavior is known.
There have been manually performed studies of bee behavior using video recordings. Naug made a one-hour video recording of 1000 tagged bees for a meticulous analysis.[15] Baracchi and Cini recorded 211 bees by taking a high-resolution photo every minute for 10 hours to track their position. According to Baracchi and Cini, the time span of one minute is enough for a bee to walk around the comb.[13] Therefore, it is useful to increase the number of images per minute to get more precise results. In addition, tracking them for at least a whole day is essential to study the behavior of bees as accurately as possible. This results in a huge amount of data. Apart from this, bee colonies have a considerable size which makes it difficult to observe them manually. It would be necessary to watch the recordings as often as there are bees in the colony in order to track each bee individually.
In conclusion, manual long-term monitoring with a high time precision of huge numbers of bees is prohibitively time-consuming.[18]
The requirements seem ideal for a computer system if it is possible to teach it the evaluation.
One approach for the study of bees using computer systems has been developed by the BioroboticsLab[1] at Freie Universität Berlin and allows to individually track every bee of a small colony over its lifetime.[9] The BeesBook setup contains a modified one-frame honey bee observation hive standing inside an aluminium scaffold that holds infrared LED cluster lamps to illuminate the hive, and six cameras (see Figure 1). Two low-resolution cameras are used for real-time waggle dance detection. Four high-resolution cameras, two on each side, observe the surface of their respective combside.[17]
Unique tags were attached to the bees' thoraxes to identify them individually. The tags were circular and curved in order to not hinder the bees' freedom of movement and work in the cells.[19]
Two convolutional neural networks were used to localize and decode the tags: the localizer is a small neural network used to localize the tags in the image and return their position. The decoder is a large neural network used to decode the tag IDs and calculate the bees' orientation.[9]
However, this approach is not complete, since the tags are not visible when a bee is located in a honeycomb cell. To study bee behavior as precisely as possible, the behavior in the

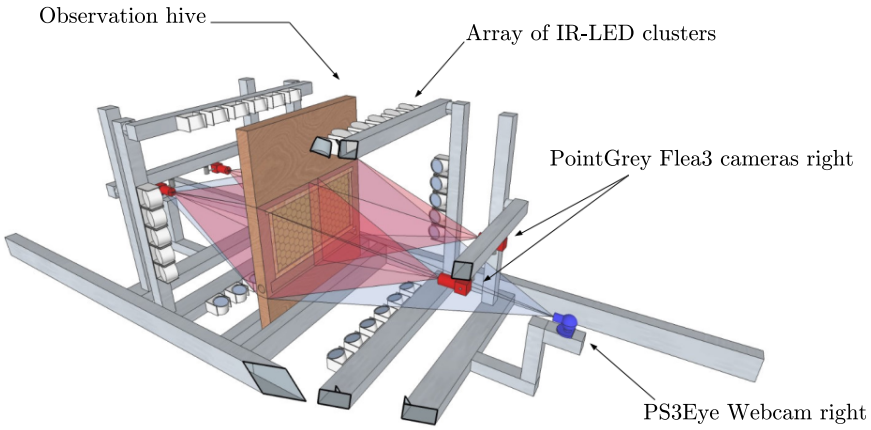


Figure 1: The construction of the BeesBook observation hive.
Two high-resolution cameras on each side are used to locate and identify the bees. They capture 12 megapixel grayscale images with a resolution of 4000 × 3000 pixels. These cameras cannot detect behaviors like the high-frequency waggle dance because of their low frame rate (6 fps in the 2018 recording set). Therefore, an additional camera was installed on each side observing the full comb at 320 × 240 pixel resolution, but at a higher frame rate (120 fps). Figure adapted from Wario et al. and Wild et al.[17, 9]

cells should be analyzable as well. Apart from that, it is very time-consuming to mark all the bees of a colony, especially considering that for future recordings new bees have to be marked each time.[1]
Bozek et al. tried to track bees without tags. They used 720 frames in total to create their ground truth dataset. 360 frames were used of 30fps recordings and 360 of 70fps recordings. The bees in the images were represented by a bee symbol to mark their position and orientation. A round symbol was used to mark the abdomen of the bees that were located in a cell. The network used was the U-Net segmentation architecture, a fully convolutional network for biomedical image segmentation.[22] The functionality of U-Net was expanded to «take advantage of regularities in the image time series patterns».[20]
However, working with recordings with a high spatial and high temporal resolution could get very expensive, also in terms of the network since the datasets to be used are huge.
The cheapest method would be to try to detect bees located in honeycomb cells and use this method to improve the already existing system of the BeesBook project, but it could also be used for future markerless tracking of bees outside the cell.
In addition to the behavior outside the cells, the behavior of the bees within a honeycomb could be analyzed. It provides information, for example, about whether a bee cleans a cell or deposits pollen.
To this end, two ground truth datasets were created in this work. These datasets were used to train a classifier, which was implemented as a neural network with a structure derived from the BeesBook localizer.

[1] In 2018, 2775 bees were marked.[18]

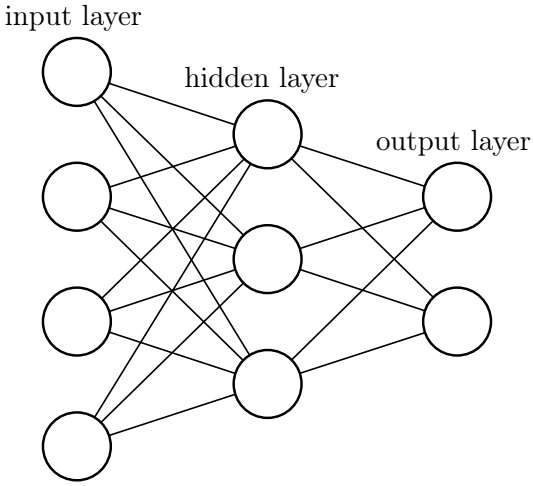


Figure 2: Example neural network consisting of an input layer, one hidden layer, and an output layer. The layers are fully connected, as each node in the hidden and output layers receives input from all nodes of the previous layer. An individual weight is associated with each link between two nodes.

2 Theory

2.1 What is a neural network?

A neural network is a set of algorithms, inspired by the work of the human brain and is used to recognize patterns.[11]
It is made of neurons which determine how the values of multiple inputs activate an output value that can either be an output of the neural network or, as part of a hidden layer, a new input for the next set of neurons. Each input is multiplied by a configurable weight and all weighed inputs are added together. The sum is passed through an activation function to avoid unbounded outputs.[27]
In principle, a neural network can have any number of layers, nodes in each layer, and output nodes. Each node can receive inputs from any node in the previous layer. If each node of a layer receives the inputs of each node in the previous layer, this layer is called fully connected, as shown in Figure 2. In general, each weight can be chosen individually. There are special cases that are less strongly linked, or that place further restrictions on the weight coefficients. In image processing, convolutional layers as in Figure 3 are particularly common. These layers only link a few spatially grouped input nodes to each output node. All output nodes share the same set of weights on their inputs. This corresponds to a convolution of the input layer with a filter kernel corresponding to the weights.

2.2 How to train a neural network

A neural network can be trained by feeding it input data with known corresponding outputs, the so called ground truth, and systematically changing its weight parameters to minimize the deviation between predicted outputs and ground truth.
To this end, the network is applied to the training data in its current state in which the weights could initially have arbitrary values. This step is called feed forward.
The quality of the resulting output is assessed with a loss function, which maps the deviation between predicted output and ground truth to a numerical value. Training is the process of minimizing that loss function.

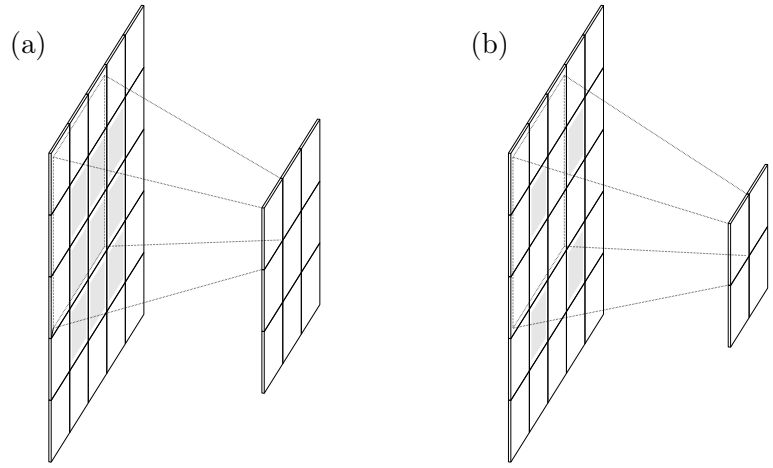


Figure 3: In a convolutional layer each output node only receives input from a small rectangular region of the previous layer, called its receptive field. It has the same size and the same set of weight coefficients for each output node. The shared weights comprise the rectangular filter kernel that is moved over the input image, optionally with a stride. Kernel size and stride size define the output size of the current layer.
(a) Convolutional layer with a 3×3 kernel and stride 1. Each output receives input from one of the grayed inputs and from its eight immediate neighbors. The receptive field of the top left output node is marked with a dotted frame.
(b) Convolutional layer with a 3×3 kernel and stride 2. Again, each output receives input from one of the grayed inputs and its eight immediate neighbors. This time the kernel skips over every second position, reducing the size of the output.

For each weight parameter, a partial derivative of the loss function is calculated in a step called backpropagation. This is followed by an optimization step in which the partial derivatives are multiplied by a learning rate and subtracted from the current weight parameter.[27] These steps are repeated multiple times to gradually reduce the loss. This process is called gradient descent.
There are many algorithms that balance between memory requirements, robustness and convergence speed. Some use more complex mechanisms such as momentum, where the gradient is filtered to include gradients from previous steps. Another possible mechanism is adaptive modification of the learning rate. One popular algorithm using these techniques is Adam, which combines the advantages of multiple previous algorithms like AdaGrad and RMSProp.[16]
Each training step over the whole dataset is called an epoch. Epochs are usually split into smaller batches that can be processed in parallel.

2.3 How to analyze the results

The previously described loss function specifies the direction of the training. The result of the loss function is called «train loss» when the network is applied to the training dataset. When the network is applied to the test dataset, the result of the loss function is called «validation loss». To analyze the training on a higher level, additional statistics are used. First, the predictions are divided into different sets: as shown in Table 1, the number of true and false positives corresponds to the number of predicted positives that were correct or incorrect. Analogous, the number of true and false negatives corresponds to the number of predicted negatives that were correct or incorrect.

		ground truth label		
		positive	negative	total
predicted label	positive	true positive t_p	false positive f_p	$t_p + f_p$
	negative	false negative f_n	true negative t_n	$f_n + t_n$
	total	$t_p + f_n$	$f_p + t_n$	N

Table 1: The total amount N of ground truth data is divided into two labels, positive (*bee*) and negative (*notbee*). While testing, the network predicts the labels of the input data either right or wrong. The predictions of the network are divided into true positives, false positives, false negatives or true negatives. These numbers are used to evaluate the quality of the predictions. The sum of true positives and false negatives is the amount of positive ground truth labels. True positives were correctly predicted as positive labels while false negatives were positive labels predicted as negatives. This applies analogously to the sum of false positives and true negatives.

With these values, different analysis functions can be calculated, which will be described in the following. The functional definitions are taken from Powers[14].
One important measure to describe the quality of a classifier is accuracy. It is the percentage of the correct predictions over the test dataset and is calculated by dividing the sum of the true positives and the true negatives by the total of the dataset:

$$accuracy = \frac{t_p+t_n}{t+n}$$

The recall is used to identify all real positive cases. It is the percentage of the correct predictions over all positive examples from the test set and is calculated by dividing the true positives by the sum of the true positives and the false negatives:

$$recall = \frac{t_p}{t_p+f_n}$$

The precision is used to calculate the proportion of predicted positive values that were real positives. This is done by dividing the true positives by the sum of the true positives and the false positives:

$$precision = \frac{t_p}{t_p+f_p}$$

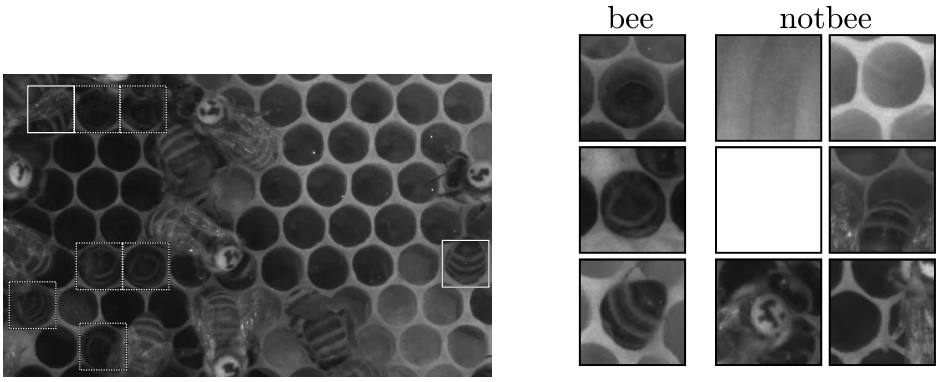
Another commonly used measure for the accuracy of a classifier is the F_1 score which is calculated as the harmonic mean of precision and recall:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision+recall}$$

The stated measures are often used to compare different classifiers, but they can also be used to track the modifications of one single classifier.

3 Ground Truth

Two ground truth datasets are needed to train a neural network: one for the training, containing around 80% of data, and a smaller one for the testing, containing the remaining amount of data.
The basis for creating the ground truth datasets were the videosets of the BeesBook project. There are sets from the different years of recording - 2014, 2015, 2016, 2018 - which differ in the camera setup. Only videos from the 2018 recording set were used, since future recordings will continue to use the same setup.
In 2018 an entire colony, 2775 bees in total, was marked by adding binary markers on the bees' thoraxes. These markers make it possible to differentiate between the bees and track them individually.[18] However, the markers are not visible when a bee enters a cell. This is why the ground truth set to detect these bees needs to consist of cell images with bees in it. This section will describe how the ground truth datasets were created. They were expanded with more negative examples and images during the work.
100 random frames were taken from 32 different videos of the 2018 recording set and split into two datasets: 75 images were designated as the source for the training dataset (*original_training*) containing the corresponding details for training as the *detail_training* set. The remaining 25 images were set aside as the source for the test dataset (*original_test*) containing the corresponding details for testing as the *detail_test* set. In each of the 100 source images, all bees located in cells (about 30 per image) were manually marked, as seen in Figure 4a, by using a simple image editing software (gimp). If the bee's entire body was in the cell, it was marked with one blue pixel. If part of the abdomen was still outside of the cell, the bee was marked with one red pixel. The differentiation of the way a bee is located in the cell is insignificant in the following. However, it could be useful for future work, as mentioned in Section 6. Bees with a still visible marker or thorax were not marked.
Even though the average size of a cell in the images used is 54×54 pixels, 78×78 regions centered around the marked positions were cropped and stored as individual detail images. This was done to provide enough margin for lossless data augmentations as described in Section 4.2. Details with a size of 54×54 pixels could cause cropped edges when rotated, as seen in Figure 5a. Since the diameter of the average cell size is about 76.8, 78×78 details provide the needed buffer, as seen in Figure 5b.
In addition to the *bee* details, a proportional number of negative 78×78 samples was taken from each image at random positions with at least 78 pixels distance to the nearest marked bee. Every *notbee* detail was checked manually to prevent that some bees in honeycomb cells were accidentally left unmarked and thus could appear as negative examples.
After this the ground truth data is build up by two datasets with positive and negative examples as seen in Figure 4b. The resulting size of each dataset is shown in Table 2.



(a) (b)

Figure 4: All bees located in cells were marked in 100 images to create the ground truth datasets. After marking the bees, random positions with a distance of at least 78 pixels to any marked bee were chosen as negative examples. Both the positive and negative examples were cropped to 78 × 78 pixels and were added to their respective dataset as details.
(a) The bees were marked with blue or red pixels depending on their depth in the cell. In this figure, the cells are framed with solid lines (red) or dotted lines (blue) for better visibility. The marking pixels are located in the center of each frame.
(b) Examples of cropped *bee* and *notbee* details from the training dataset. The *notbee* details correspond to random parts of the recorded image, for example empty cells, the wooden frame of the comb, and bees that are not located inside a cell. In this figure, black frames were added around the details for better visibility.

(a) (b)

Figure 5: The average size of a honeycomb cell in the images used is 54 × 54 pixels. To prevent image-loss, the details were cropped in the size of 78 × 78 pixels.
(a) A 54 × 54 pixels detail rotated 45°. Due to the small size and the missing margins, the edges are cut off.
(b) A 78 × 78 pixels detail rotated 45°. After rotating, the image can be cut to 54 × 54 pixels without loss.

dataset	detail_training	detail_test
images	75	25
details (*bee*)	2,276	825
details (*notbee*)	22,500	7,500
details (total)	24,776	8,325
ratio	75%	25%

Table 2: The dataset used for training was the *detail_ training* set, containing the *bee* and *notbee* details from the corresponding 75 raw images. The amount of *notbee* details is 300 for each raw image. With a batch size of 200, each training epoch was split up to 124 batches. To test the network, the *detail_ test* set containing the *bee* and *notbee* details from their corresponding 25 raw images was used. In total, 33,101 details were created.

4 Implementation

This section will discuss the basic implementation of the neural network, as well as the extension to being applicable to the raw images. The evaluation of the final architecture, considering localization and heatmaps, will be covered in Section 5.

4.1 The Neural Network

The neural networks were implemented using the PyTorch framework[3]. The first neural network, which was implemented mainly to get acquainted with the PyTorch workflow and the supported functionalities, was inspired by a simple network used in the PyTorch image recognition tutorial[6]. The network feeds the input through several convolutional and linear layers and then returns the output.
The initial network was adapted to the input data and the necessary output classes. The input was a 32×32 tensor resulting from a scaled grayscale 54×54 pixel image. The neural network reduced the tensor to a 1×2 tensor as output. The output size of two tensor elements was chosen to represent the two classes *bee* and *notbee*. The layers used for this are shown in Figure 6. Each layer is activated by a «rectified linear unit» (ReLU), which returns 0 for negative outputs and the unchanged value x if x is positive. The first runs of the neural network used the *bee* and *notbee* details obtained from 100 images and were trained and tested for 50 epochs. Results of two different runs can be seen in Figure 7. The initial threshold used to differentiate between predicted bees and predicted negative examples was arbitrarily set to 0.8. This means that if the output of the network was above 0.8 while testing, the prediction would be considered a predicted bee.
These initial runs were not yet used to truly evaluate the results of the network, but to improve the design of the network and create the needed graphs and values to make an evaluation possible. The values used were the results of the loss function of the training phase as well as the results of the loss function, the accuracy and F_1 score of the testing phase.
As loss function, CrossEntropyLoss[4] was adopted from the PyTorch tutorial. BCELoss, which was used for binary classification later on, was not applicable since the output tensor still contained two elements.
Testing and modifications of the network were repeated until it produced good results. However, this kind of neural network only works for the defined input size of 32×32 tensors. It can be applied to detail images in the size of a cell for training and testing but not to images of the entire honeycomb. Testing on larger inputs is desirable because it would

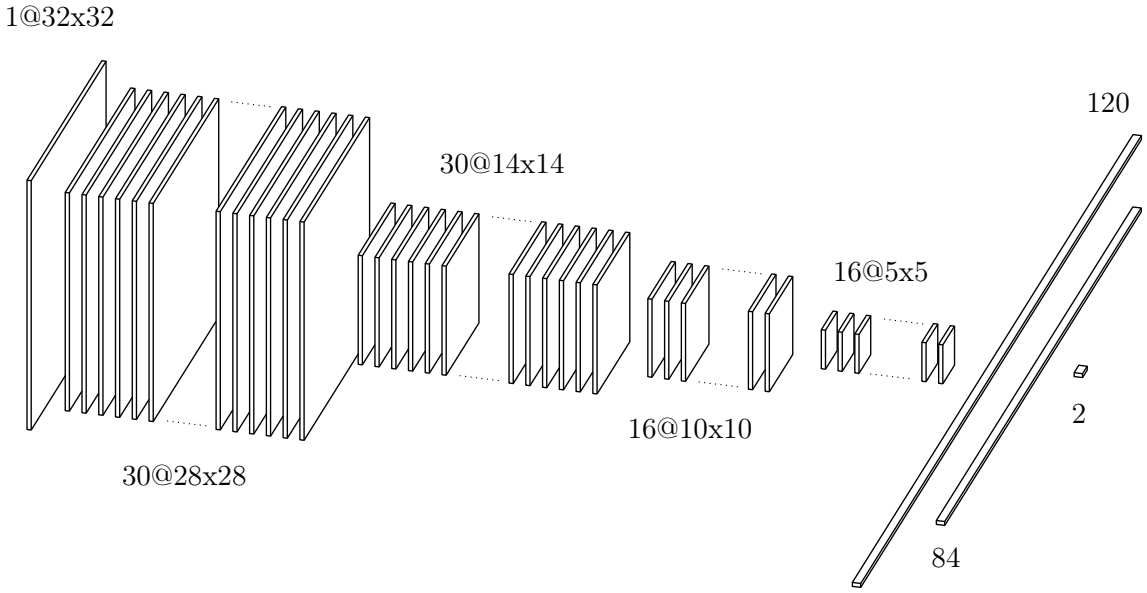
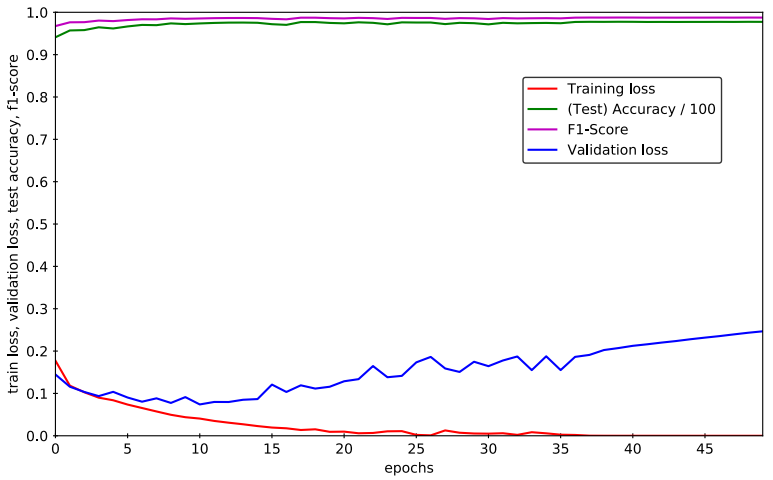


Figure 6: The basic neural network consisted of a 32 × 32 input layer, 6 hidden layers, and the 1 × 2 output layer, predicting the *bee* and *notbee* labels of the image. The first hidden layer uses convolutions with 5 × 5 kernels to return 30 28 × 28 tensors. On these tensors, max pooling was applied, cutting the size of the previous layer in half. These two steps were repeated: a convolution with a reduced set of 5 × 5 kernels returned 16 10 × 10 tensors, turned into 5 × 5 tensors by another max pooling step.
The tensor was reduced to the 1 × 2 output tensor using 3 fully connected linear layers.

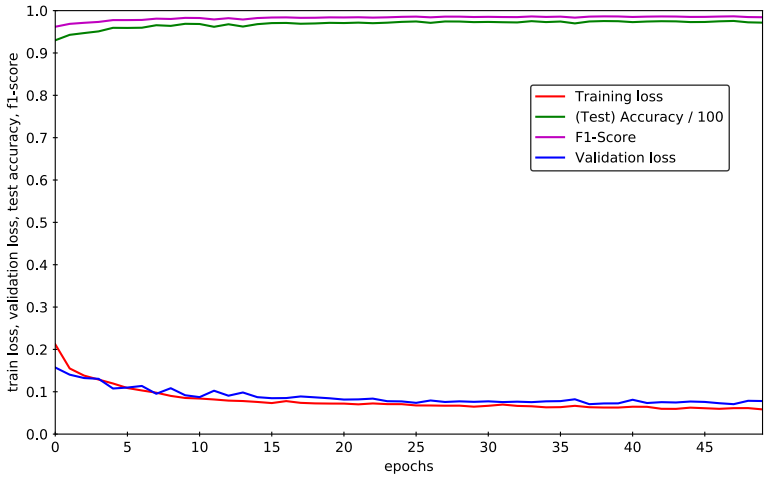
allow to produce heatmaps with spatially resolved prediction strengths rather than just a single prediction. These heatmaps could be used to localize the bees in honeycomb cells. To make the network capable of being applied to images of arbitrary size, the pooling layers and the fully connected layers have to be replaced. The solution is to change the network architecture to a fully convolutional neural network. This is described in Section 4.3.

4.2 Data Augmentation

To build a useful model as mentioned in Section 4.1, the validation loss must not increase while the train loss is decreasing, otherwise the model is overfitting to the training dataset, as can be seen starting from epoch 8 in Figure 7a.
Overfitting means that a neural network's predictions are better on the training data than on the test data. This case occurs when the network consistently learns the same mistakes.[23]
There are many possibilities to prevent this condition, such as dropout, which randomly disables parts of the network during training, or batch-normalization, where inputs are normalized over an entire batch during batched training.[21, 24] Another very effective method, data augmentation, is used for this network.
Data augmentation allows to to enlarge the given dataset by using different techniques to manipulate the images.[12]
PyTorch offers a package[5] with a wide range of data transformations which was used to randomly augment the images. The augmentations used were executed in random order to allow the biggest possible variance of the images. They were also applied randomly, which means that all of the images were manipulated with different augmentations. It is possible



(a)



(b)

Figure 7: The plots contain the losses of both training and testing of the neural network as well as the accuracy and F_1 score from the testing phase.
The threshold used for testing was 0.8 which means that all output values above 0.8 were taken as a bee in a cell and all values below as a negative example.
(a) The neural network was trained and tested with non-augmented details for 50 epochs. While the train loss decreases, the validation loss increases. This is an indication for overfitting. The trained model does not generalize well to the used test data.
(b) After applying data augmentations, the train loss decreases more slowly than without the augmented details. However, the validation loss decreases and there is no visible indication of overfitting.

that some images were not augmented at all.
The following transformations were used: rotation with an angle between 0° and 180°, horizontal flip, vertical flip, change of brightness and contrast with a percentage between 0% and 50%.
The impact of data augmentation on an overfitting neural network can be seen in Figure 7.
All images were center cropped to a size of 54 × 54 pixels after the listed augmentations. The previously cropped images were scaled down to 32 × 32 pixel images, converted to grayscale, and transformed into a tensor.
The 32 × 32 pixel input resolution was inspired by the *CIFAR10* and the *80 million tiny images* datasets. Both are flagship datasets when it comes to training and testing neural networks, since they contain a lot of image data. Despite their low image resolution, they work well. Apart from that, the low resolution allows lower memory consumption and the network runs faster than with datasets of the same size but higher image resolution.[8, 7]

4.3 Fully convolutional neural network (CNN)

Due to the fixed tensor sizes mandated by the fully connected linear layers, the architecture of a regular neural network as described in Section 4.1 is not able to test on images larger than the defined input. To be able to not only decide if a bee is in the given detail image of a cell, but also to localize bees in cells in an image of a whole honeycomb, the network architecture needs to be modified.
The solution is a fully convolutional neural network. This kind of architecture is intended for images as an input and therefore is able to work itself through an image of any size.
The three main types of layers to build a convolutional neural network are convolutional layers, pooling layers and fully-connected layers, as they are known from regular neural networks. Again, each layer is activated by a ReLU. However, CNNs do not contain linear layers. In short, a convolutional neural network architecture is a list of layers to transform an image volume into an output volume.[2]
To change the network architecture into a CNN, the linear layers had to be replaced by convolutional layers. Six convolutional layers were used to reduce the 32 × 32 input tensor to a 1 × 1 output tensor. Each convolutional layer, except for the last one, was activated with a ReLU-layer. The last layer was activated by a sigmoid layer. The sigmoid function

$$S(x) = \frac{1}{1+e^{-x}}$$

limits the output of the network to a range of 0 and 1. This means that negative outputs approach 0 and positive values approach 1.
The input was a 32 × 32 pixel grayscale image transformed into a tensor. As seen in Figure 8, the input can be minimized to a smaller tensor with the necessary information from the corresponding kernel-sized image section with different kernel and stride sizes. The smaller tensor is then used for the next convolutional layer. These steps are repeated until the resulting output is a 1 × 1 tensor. The output tensor contains the necessary information of the 32 × 32 input tensor. This way, the network is able to correctly detect whether what was shown in the input image is a bee in a cell or not.
Just like the neural network in Section 4.1, the values used to evaluate the results of this architecture were the loss function output of the training phase as well as the results of the loss function, the accuracy and F_1 score of the testing phase. The loss function was changed to binary cross-entropy loss (BCELoss[4]) since we now face a binary classification problem: the only two classes are *bee* which corresponds to 1 and *notbee* which corresponds to 0. Figure 9 shows nine examples of details with their true label and their prediction. The predictions on the *notbee* details are far below 1, whereas most of the predictions on

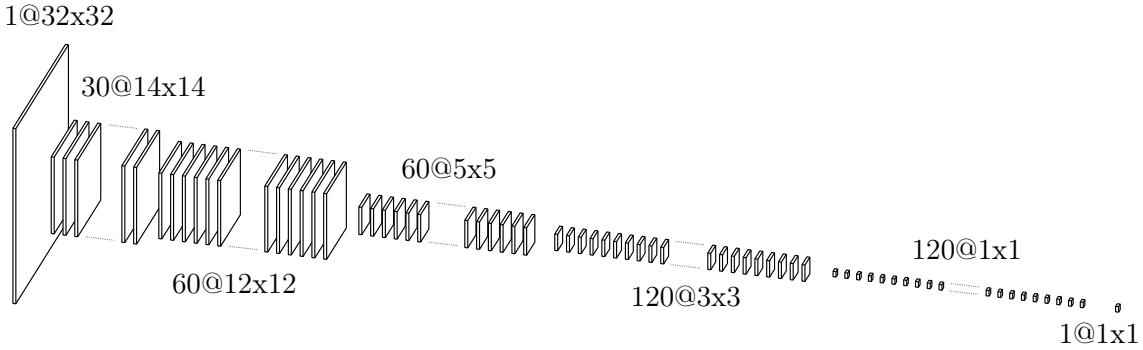


Figure 8: The CNN consists of an input layer, five hidden layers and one output layer. During training, the input layer is a 32 × 32 tensor, resulting in a 1 × 1 output tensor. However, the network can be applied to tensors of any size. The first hidden layer uses convolutions with 5 × 5 kernels and stride 2 to return 30 14 × 14 tensors. The second layer applies 3 × 3 kernels and results in 60 12 × 12 tensors. The third layer uses convolutions with 3 × 3 kernels and a stride of 2 to return 60 5 × 5 tensors. The fourth layer applies 3 × 3 kernels and results in 120 3 × 3 tensors. The fifth layer applies 3 × 3 kernels and results in 120 1 × 1 tensors. The output layer combines the outputs of 120 1 × 1 tensors into the final 1 × 1 output tensor to predict the image's label.

the positive examples are close to 1. Overall this is a sign that the network's classifications on the test set are with high confidence.
After changing the neural network to a convolutional neural network, it can be applied to raw images of the entire honeycomb. This allows to produce heatmaps and to predict the locations of bees in cells.

4.3.1 Extension to full images

To evaluate if the network is able to not only detect but also to localize bees in cells, it must be tested on the raw images of the entire honeycomb. To this end, the next step is to extend the neural network to a more realistic environment and test it on the images of the honeycomb, called «raw images» in the following.
Even though the architecture of a convolutional neural network already allows to test on the images of the entire hive, a few adaptations have to be made: because of the scaling and the different steps of the network, the coordinates of bees in cells found by the net will not match the coordinates of the manually marked bees, for better understanding see Figure 10. The previous steps need to be reversed to convert the coordinates found to the original position in the raw image.
However, before thinking about how to undo previous steps to get the predicted original position, how does the neural network allow to localize bees in cells?
In Section 4.3, the network architecture is described for 32 × 32 input tensors, resulting in an output of 1 × 1 tensors that contain a value between 0 and 1. It works the same way for a 2370 × 1778 tensor, except that the output is now a 586 × 438 tensor. Each value in the tensor is a prediction for the corresponding pixel in the 586 × 438 pixel image (Figure 11a).
Non-maximum suppression was applied to filter the values that predict bees in cells. First, a maximum filter was used to get the highest output-values in a given radius. The chosen radius was 8 pixel because it is the approximate cell size in the 586 × 438 pixel images.

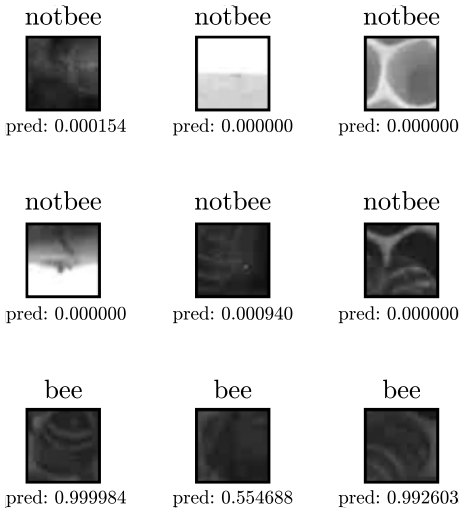


Figure 9: For each image tested, the network returns its prediction on the label. The predictions on the *notbee* details approach 0.
The predictions on the first and the third *bee* detail approach 1, while the second *bee* detail is not that close to 1. This example of a bee is not as clear as the other ones since the bee is in the cell entirely. The results were plotted while testing after 100 epochs of training on the dataset described in Table 3.

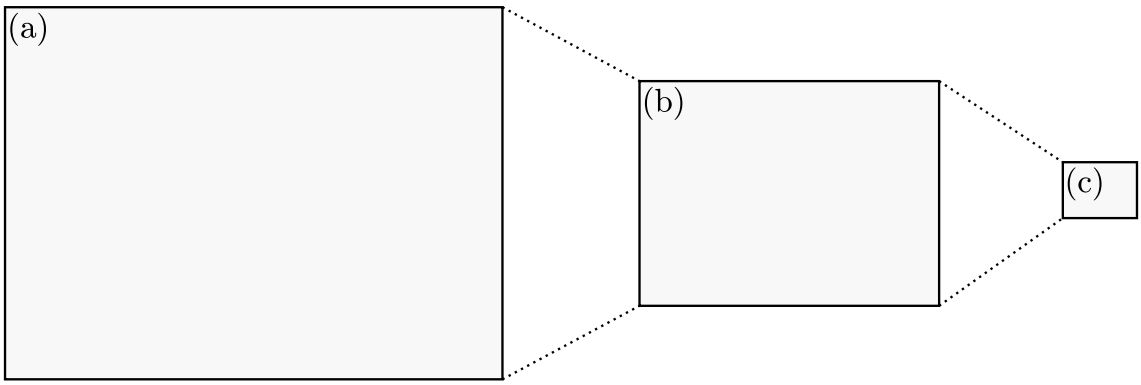


Figure 10: Relative sizes of unscaled and scaled input images and output tensor.
a) The input images are 4000 × 3000 pixel frames, taken from the video recordings mentioned in Section 1 and 3.
b) Like the details, the inputs were scaled by a factor of $\frac{32}{54}$, to 2370 × 1778 pixels before being processed by the network.
c) The output of the neural network applied to these inputs is a 586 × 438 tensor. Each output value corresponds to a 32 × 32 region of the input tensor, or a 54 × 54 region of the source image.

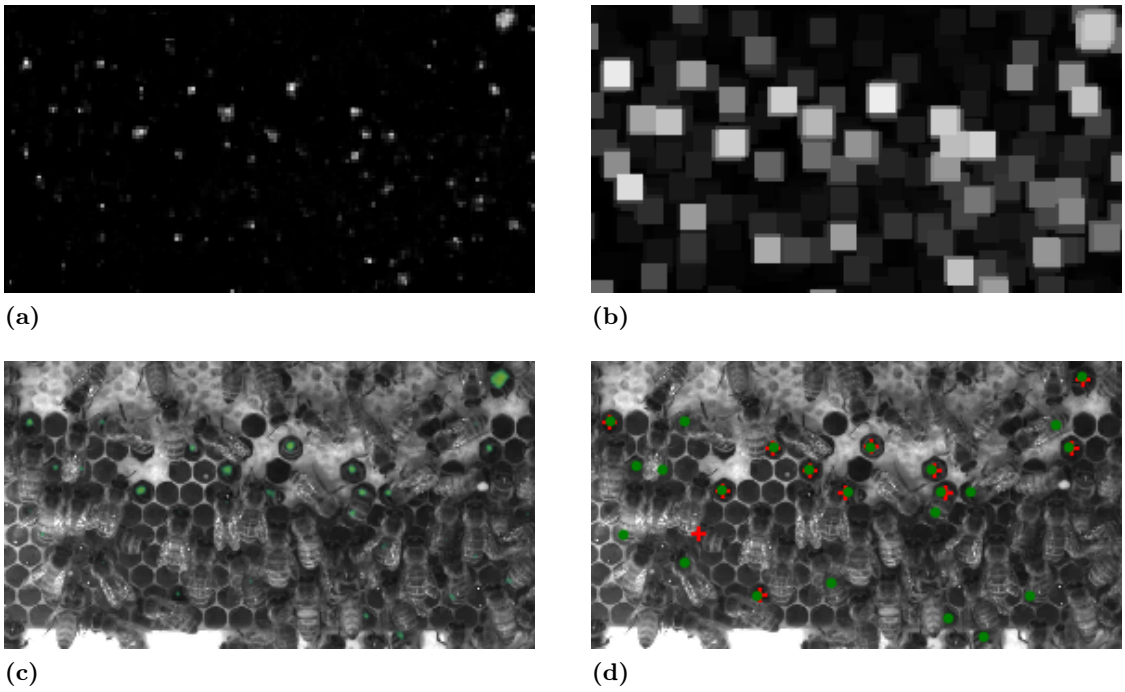


Figure 11: The step-by-step process to create a heatmap and find the original position of the coordinates found by the network.
(a) First, the CNN is applied to the scaled 2370 × 1778 image. The output is a 586 × 438 heatmap with values between 0 and 1, depending on the prediction.
(b) To filter the maxima in a given radius, non-maximum suppression is applied. To this end, a maximum filter with 8 × 8 kernel size is applied to the CNN output, resulting in an image of superimposed squares with the same value as the corresponding local maximum. Then the output and maximum filtered output are compared. The positions where the two are equal are recorded, those are the local maxima.
(c) Heatmaps were generated by scaling and superimposing the adapted output onto the raw image to visualize the predictions of the network.
(d) The coordinates found are calculated to fit the coordinates of the raw image. The resulting positions (green) are compared to the manually labeled ones (red).

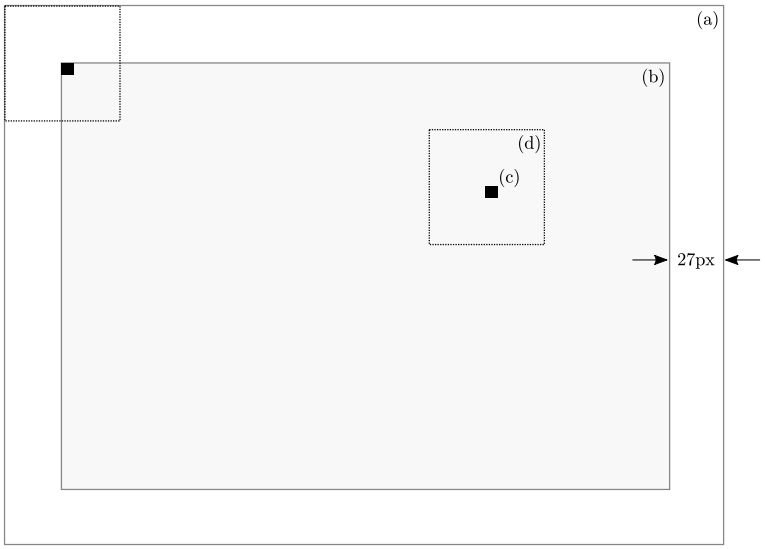


Figure 12: To generate a heatmap the network was applied to the downscaled 2307 × 1778 image. The output was a 586 × 438 tensor. Each heatmap value (c) has inputs from a 32 × 32 region of the scaled input image, corresponding to a 54 × 54 region (d) of the unscaled 4000 × 3000 image (a). To overlay the heatmap onto the raw image correctly, it has to be scaled back up to 3946 × 2946 and overlaid with a margin of 27 pixels on each side (b).

The maximum filter sets all values within the 8 × 8 pixels to the corresponding maximum. This creates different superimposed squares with the respective maxima (Figure 11b).
To only obtain the values at which a bee is predicted, the results of the maximum filter are equated to the maxima of the output. The resulting image has all pixels where the raw output value is larger than 0.8 and where the raw output and the maximum filtered output are equal set to 1 (true) and all other pixels set to 0 (false). The positions of the white pixels are equal to the positions of the local maxima (within their 8 pixel surrounding) of the output.
The results of these steps were visualized by overlaying the heatmaps onto the raw images (Figure 11c) and by marking both the ground truth positions and the detected positions in the raw images (Figure 11d). The heatmaps were generated by scaling the output of the network up to 3946 × 2946 and placing it over the raw 4000 × 3000 pixels image as shown in Figure 12. To mark the detected positions, a list of the heatmap's local maxima was created from the results of the non-maximum suppression. The steps of the network and scaling have been reversed to get the coordinates' position in the raw image (Figure 11d). This calculation is divided into multiple steps because the network applies a convolution with stride 2 to an intermediate tensor with odd width and height. The resulting tensor's dimensions are halved, but rounded up. This rounding has to be reversed.

$$xvalue = \lfloor(((xvalue + 0.5 + 1 + 1) * \tfrac{883}{442} + 1 + 1) * 2 + 2 - 0.5)\rfloor \cdot \tfrac{54}{32}$$

$$yvalue = \lfloor(((yvalue + 0.5 + 1 + 1) * \tfrac{1179}{590} + 1 + 1) * 2 + 2 - 0.5)\rfloor \cdot \tfrac{54}{32}$$

Figure 13 shows each step of the network on a raw image to retrace the calculations. Each value of the 586 × 438 output tensor corresponds to the center point of a 54 × 54 detail of the raw image, therefore each reversed step calculates the center point in a previous layer. First, 0.5 is added to move the position into the pixel center. Then, each time a

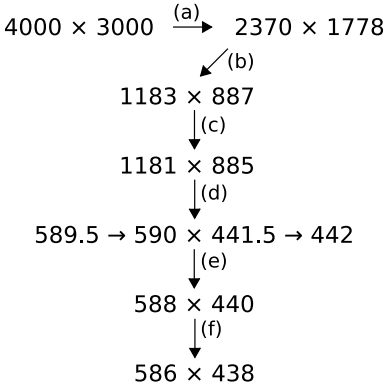


Figure 13: The input is a 4000 × 3000 pixel image.
(a) The raw image is scaled by the factor $\frac{32}{54}$.
(b) The first step of the CNN uses 5 × 5 kernels applied with a stride of 2. The resulting tensor width and height are thus reduced by 4 and halved.
(c) The second step uses 3 × 3 kernels with a stride of 1, subtracting 2 from width and height. In the following steps, the kernel size remains 3 × 3.
(d) 2 is subtracted from the width and the height. Since the stride is 2, width and height are halved and rounded up.
(e) 2 is subtracted from the width and height.
(f) 2 is subtracted from the width and height, the resulting image size is 586 × 438.

convolution subtracted some borders from the width and height, half of that has to be added back (only one border side, since the position is measured from the top left).
For each time the stride of a convolution halved width and height, the value is multiplied back by 2. The multiplication with $\frac{1179}{590}$ is a special case to reverse the rounding in step (d) of Figure 13. After reversing each step of the network, the previously added 0.5 is subtracted again. The final step is to multiply the result by $\frac{54}{32}$, which reverses the scaling.
The marked ground truth coordinates were compared to the calculated coordinates by finding the nearest neighbor. Calculated coordinates which were not within a radius of 27 pixels around a ground truth marker were considered false positives. If there were more than one coordinate within the radius, only the one closest to the ground truth marker was considered a true positive. The other coordinates were considered false positives. If no matching local maximum was found in the given radius, the respective coordinate was considered a false negative. As described in Section 2.3, these values enabled to calculate the prediction, the recall and the F_1 score.

dataset	detail_training	detail_test
images	75	25
details (*bee*)	2,276	825
details (*notbee*)	41,250	13,750
details (total)	43,526	14,575
ratio	75%	25%

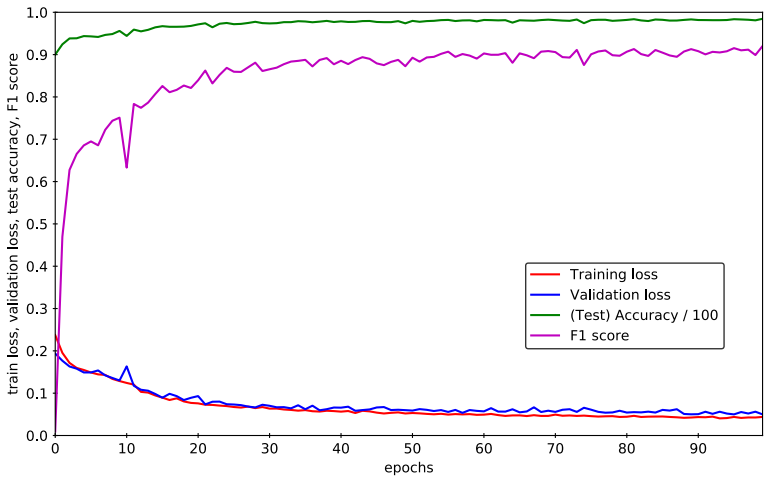
Table 3: The previously described dataset (Table 2) was extended by using 550 *notbee* details instead of 300. The number of detail images in the *detail_training* set and in the *detail_test* set was raised to 58,101 in total. The epochs while training the network now included 218 batches in the size of 200.

5 Evaluation

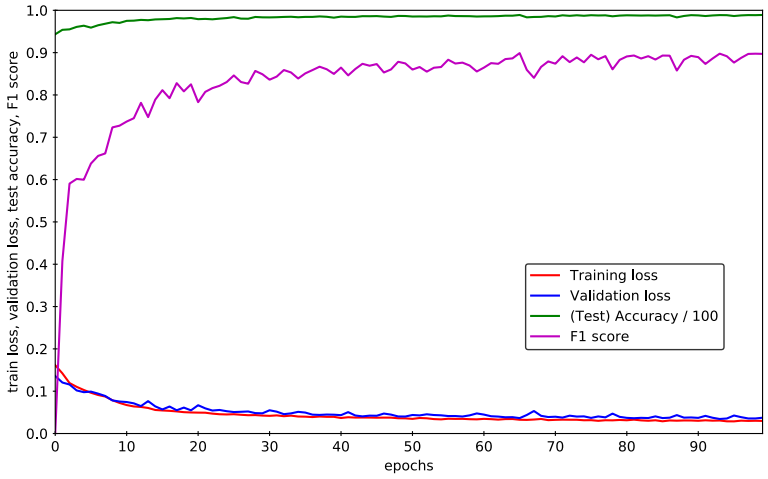
In the previous sections, the basic architecture of the network was explained as well as the changes that were necessary to apply it directly to the raw honeycomb images. In this section, the results of the testing on the details from the *detail_test* set and on the raw images from the *original_test* set are evaluated.
While testing on the raw images, the data augmentation and the ground truth datasets were expanded in an effort to improve the spatial predictions. To gain comparable results, each iteration the network was trained for 300 epochs on the *detail_training* set. After each epoch it was tested against the *detail_test* set and the *original_test* set.

5.1 Analysis of the test runs

The network described in Section 4.3 was trained on the *detail_training* set. Validation loss, accuracy, and F_1 score were plotted over the number of epochs together with the train loss.
The number of *notbee* details was increased to test the influence of the amount of negative examples in the dataset on the training and testing of the network. Instead of using 300 *notbee* details, the amount was increased to 550 details. For an average of 30 78×78 *bee* examples in each image, there is space for up to about 1972 sensible *notbee* examples in the raw 4000×3000 images, by area. Because of random sampling and depending on whether overlap is allowed, this number could be both a lot higher or lower. So there are still potential gains by increasing the number of negative examples even more, but the given increase should already give an outlook on how the extension of the negative examples can be expected to affect the training.
This change was only plotted for 100 epochs with the initial threshold of 0.8 since during these epochs it could already be seen that the increased number of negative examples only has a small influence on the results. The differences are values of about 0.01 and 0.03, as can be seen in Figure 14.
The new dataset (Table 3) was tested for 300 epochs with a threshold of 0.99 to evaluate the results (Figure 15a). This threshold was determined as the best threshold for the current iteration of model and dataset after comparing 100 thresholds between 0.01 and 0.99 over the last epoch of testing on the *original_test* set. The thresholds to be used for the following iterations are determined in the same way.
The test accuracy is initially above 0.9 and consistently increases up to about 0.98 over the 300 epochs. This is a sign that the network is generalizing well to the test data. However, there is the danger that this is an effect of similarities between the training and



(a)



(b)

Figure 14: The loss function of training and validation dataset, test accuracy, and F_1 score for the initially used dataset (2) and the new dataset (3) over 100 training epochs.
(a) The train loss decreases from 0.24 to 0.08 in the first 20 epochs. It keeps decreasing at a lower rate until a value of about 0.04. The validation loss starts slightly below the train loss at about 0.19. It slowly decreases to 0.05, ending up slightly above the train loss. The accuracy steadily increases from 0.9 to 0.98. Starting from 0, the F_1 score strongly increases in the first 20 epochs to about 0.82. It then levels out to about 0.92.
(b) Both train and validation loss start at around 0.15 with the validation loss slightly above the train loss. In the first 20 epochs, the loss reaches 0.05 and then decreases slowly to a value of about 0.03. The validation loss decreases to 0.05 in the first 15 epochs, afterwards it slowly approaches 0.04. The accuracy slowly approaches 0.99 starting from about 0.95. In the first 17 epochs the F_1 score strongly increases from 0 to about 0.82. From there on it levels out to 0.89.

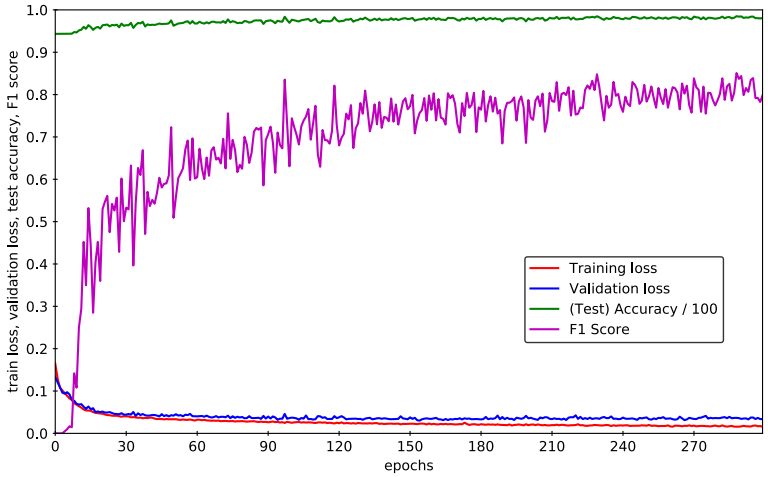
test datasets, and that the network might not generalize well to data from different sources. The F_1 score is moderately good. It increases as well, but in contrast to accuracy, it is lower and not as stable. The F_1 score fluctuates with an amplitude of about 0.1. The reason for the F_1 score being lower than the accuracy is that it is not affected by correctly predicted *notbee* details but only by the correctly predicted *bee* details and the wrongly predicted details.
Both train and validation loss values are initially small (below 0.2) and keep decreasing over the 300 epochs. The train loss is consistently slightly below the validation loss over time. However, their difference seems to be increasing very slowly. This could be an indication that with longer training, the difference increases further and thus leads to overfitting.
The results show that the network is able to differentiate whether the detail shows a bee in a honeycomb cell or not. Since the aim is not only to differentiate between *bee* and *notbee* but to detect and localize bees in cells on images of the whole honeycomb, the network is tested on raw images in the following.

5.2 Application to raw images

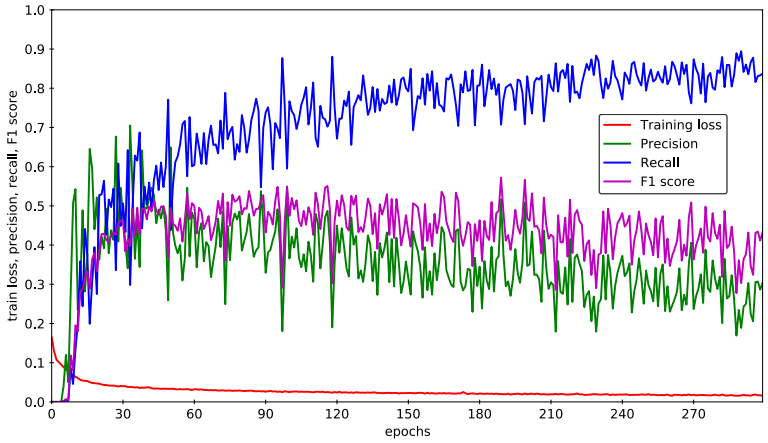
The same training results from the previous section were used to test the network against the *original_test* set. The accuracy cannot be plotted for runs on the raw images since the number of true negatives is not well defined. The ground truth data for the *original_test* set only consists of the manually marked positions while every other position is counted as a negative example. Figure 11d shows the values given to evaluate: the red crosses are ground truth markers corresponding to the manually marked positions. The green points are the positions predicted by the network. Each ground truth marker that is not accompanied by a predicted marker within a radius of 27 pixels is counted as false negative. Similarly, a predicted marker without a close-by ground truth marker is counted as false positive. If a ground truth marker and a predicted marker overlap they are counted as true positive. To visualize the evaluation of the test, precision, recall, and F_1 score were again plotted over the number of epochs together with the train loss as seen in Figure 15b.
The quality of the network when applied to the *original_test* set is expected to be worse than when applied to the *detail_test* set since the network is not tested on the same input format as it is trained on. This can be the reason for the reduced F_1 score.
The increase of the recall while the prediction is decreasing is an indicator that the network detects too many false positives. This is why the threshold must be set high (threshold = 0.99), otherwise the percentage of false positives would be higher than the current 69% of falsely predicted bees in cells. The increasing recall and decreasing precision are the reason for the decreasing F_1 score. Possible ways to mitigate that could be the extension of the data augmentation with translations, and replacing the labels 0 and 1 with a label strength between these two values. This modification can increase the precision and subsequently increase the percentage of correctly predicted bees. This is covered in the following.

5.2.1 Label strength

The network is already able to localize bees but only with a mediocre precision. To improve the precision of the predicted positions, the network is trained to produce sharper peaks in the heatmaps. This can be achieved by suppressing predictions around the ground truth markers slightly with increased distance. To this end, the data augmentation is extended with translations of the *bee* details. A random offset between 0 and 15 is used to move the details vertically and horizontally. The label strength presented to the network is set



(a)



(b)

Figure 15: The network was trained for 300 epochs. The results of the training were used to test the network on the *detail_ test* set (results shown in (a)) and the *original_ test* set (results shown in (b)). Before the sets were tested over all epochs, the best threshold was determined. For this, all images from *original_ test* were tested over thresholds between 0 and 1 in steps of 0.01 after the last epoch.Before the sets were tested over all epochs, the best threshold was determined after the last epoch, by testing all images from the *original_ test* set over thresholds between 0 and 1 in steps of 0.01. The threshold used eventually is 0.99.
(a) In the first 20 epochs, train and validation loss are strongly decreasing from a value about 0.15 to a value about 0.05. They decrease for the remaining 280 epochs, this time at a lower rate. The train loss approaches 0.01 while the test loss stops at 0.03. The test accuracy starts at 0.94 and slowly increases to 0.98 over the whole testing phase. The F_1 score starts at 0 and strongly increases over the first 15 epochs to about 0.5. While it still increases slowly for the remaining epochs, the value is fluctuating. It converges to 0.8.
(b) The same training results were used to test the network on the raw images. The precision is heavily fluctuating between values of about 0.4 and 0.7 for the first 34 epochs. The fluctuation lessens over the remaining epochs. All in all, the precision is slowly decreasing to a value of about 0.3. The recall is increasing from 0 to about 0.7 in the first 40 epochs. It also fluctuates with an amplitude of about 0.4 in the first 40 epochs. The recall still increases for the remaining epochs to about 0.8, while the fluctuation lessens to an amplitude of about 0.1. The F_1 score, that is calculated by precision and recall, increases from 0 to about 0.5 in the first 40 epochs. For the remaining epochs it slowly converges to 0.4.

to 1 only for 0 offset, but decreased with increasing offset. The *bee* details from the *detail_ training* set were cropped again, this time to 130 × 130 pixels. This provides enough buffer to add translations to the data augmentations described in Section 4.2 without loss of detail. For the translated detail images, the label strength presented to the network is reduced with increasing length of the translation offset. As function between offset length and label strength a gaussian curve with $\sigma = 4$, normalized to 1 at the origin, was chosen:

$$f(x) = e^{-\frac{1}{2}(x/\sigma)^2}$$

Figure 16 shows the set label curve and the resulting output values returned by the network in dependence of the offset norm. The values show a similarly strong decrease with increasing offset norm as the curve, but at low offset values the output never reaches 1. Apart from this a large number of values are close to 0 regardless of the offset.
Both the testing on the *detail_ test* set, and on the *original_ test* set were carried out with threshold 0.6, which was determined as the best threshold value. This is not surprising, given the low output values for predicted *bee* details at small offsets.
As can be seen in Figure 17a, the accuracy of the test on detail images is not much different compared to the result of training without an offset. This can be explained by the fact that it is dependent not only on the true positives, as the F_1 score, but also on the true negatives, which are not affected by the translation augmentation. The F_1 score is affected by the translations, which is why it is lower than without offsets. The honeycomb cell containing the bee is not centered in the positive training examples and with increasing offset the label strength is decreased. In contrast, all positive test examples are perfectly centered. The validation loss slowly approaches the train loss. Both are decreasing. All values, the increasing F_1 score and the decreasing losses indicate that the network works better for testing on details if no offset is used.
However, the testing on the *original_ detail* set has visibly improved. Instead of decreasing to the last epoch, precision and F_1 score increase until they level out to values between 0.3 and 0.4, as shown in Figure 17b. Precision and recall still have a big difference of about 0.2 since the percentage of correctly predicted bees is only 29%. To increase the precision of the predictions, the reducing of the label strength is weakened by using a higher sigma. The $\sigma = 15$ was chosen to reach a label strength of about 0.5 at an offset length of 15. The corresponding graph can be seen in Figure 18. The graph shows that the network output has a tendency to decrease with increasing offset, as requested. However, that decrease is much less pronounced than specified by the label strength curve. It remains to be seen whether this could be improved by longer training, or whether the network architecture is just not complex enough to reproduce the desired label strength curve.
The best threshold was determined and set to 0.96 for testing on the details and on the raw images. The test accuracy and F_1 score are shown in Figure 19a. The accuracy did not change much compared to the accuracy seen in Figure 15a and 17a. However, compared to $\sigma = 4$, the F_1 score improved by about 0.2. The probable reason is the training with a higher label strength. It presents the network with a much stronger, more consistent signal, especially at small offsets when most of the identifying features of a bee in a cell are still within the detail image.
The extended augmentation of the *detail_ train* set can be the reason for the validation loss running slightly below the train loss while testing on the *detail_ test* set. The training set is augmented with translations and reduced label strength depending on the translation vector. This is a more complex problem and, given the same network and number of training epochs, results in increased train loss. The test set is not augmented with translations

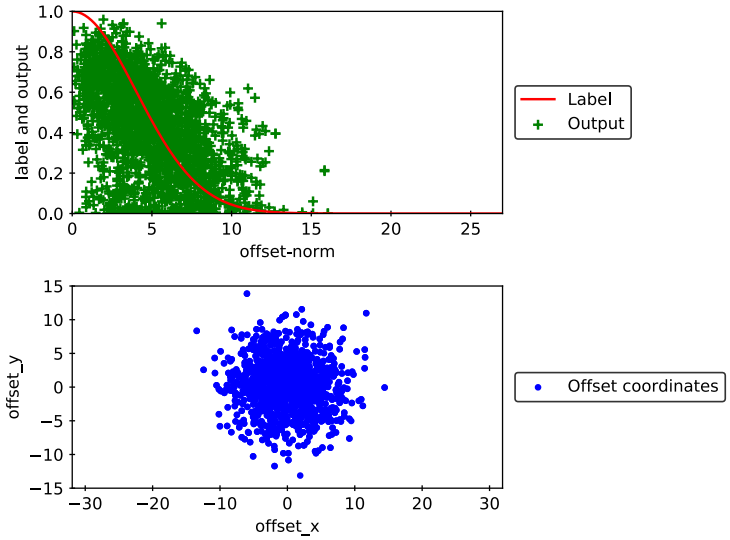
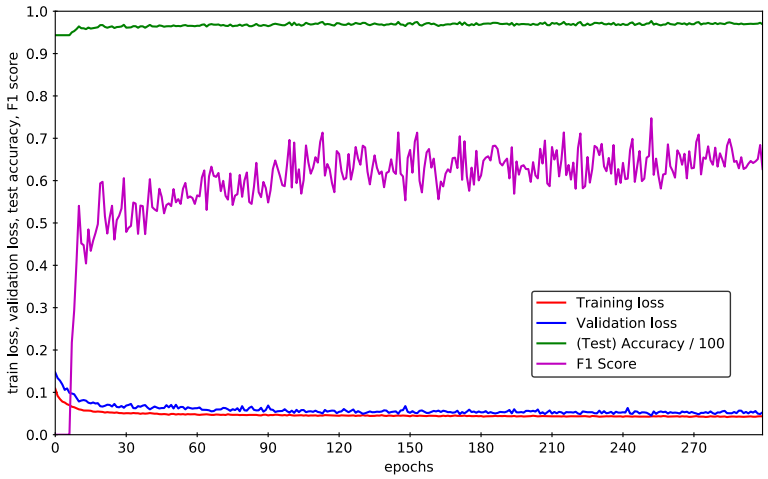
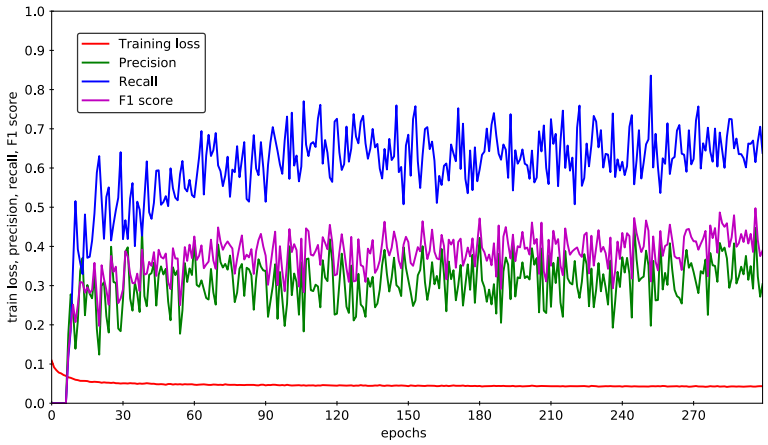


Figure 16: The upper plot shows the label and output values over the norm of the translation used on the input details in the last epoch of training. The red line corresponds to the label values with which the network was trained. It follows the probability density function of a normal distribution with $\sigma = 4$, normalized to 1.0 at offset 0. Each green point corresponds to the output value of a *bee* detail from the *detail_training* set. For a perfect training result, they would be expected to be close to the red line. In this run, most of the output values appear to loosely follow the shape of the curve with an inaccuracy of ± 5. Notably, at low offset length the output values stay below the curve. Only few outputs are above 0.9. A second cluster of output values can be found near 0 at all offsets.
The lower plot visualizes the chosen offsets which were randomly picked from a normal distribution with $\sigma = 4$. Offsets with a larger absolute value than 15 either horizontally or vertically were discarded and redrawn.



(a)



(b)

Figure 17: The network was trained for 300 epochs, using a random offset up to 15 pixels translation and offset dependent label strength with $\sigma = 4$. The best threshold to use for the testing in both graphs was 0.6.
(a) In the first 15 epochs, the train loss decreases from 0.1 to 0.06. It still decreases slowly up to epoch 200 where it levels out to 0.04. The validation loss starts a little higher at 0.15. It decreases to 0.07 for the first 20 epochs and then slowly decreases to 0.05. Starting at 0.94, the accuracy increases to 0.96 in the first epochs. It slowly increases to about 0.97 in the remaining epochs. The F_1 score strongly increases from 0 to 0.54 for the first 11 epochs. Afterwards, it increases slowly and levels out to about 0.65.
(b) The network was tested on *original_ test* over the same training results. The precision increases from 0 to about 0.37 in 14 epochs. It does not seem to increase or decrease any further from there on but it fluctuates between 0.27 and 0.4. The recall strongly increases from 0 to 0.63 in the first 21 epochs. Until epoch 107 it increases to 0.77 but much slower. Afterwards it slightly decreases and levels out between 0.6 and 0.7 without signs of increasing or decreasing. The F_1 score increases from 0 to about 0.36 in the first 19 epochs then it slowly increases to about 0.38 with a lot of noise.

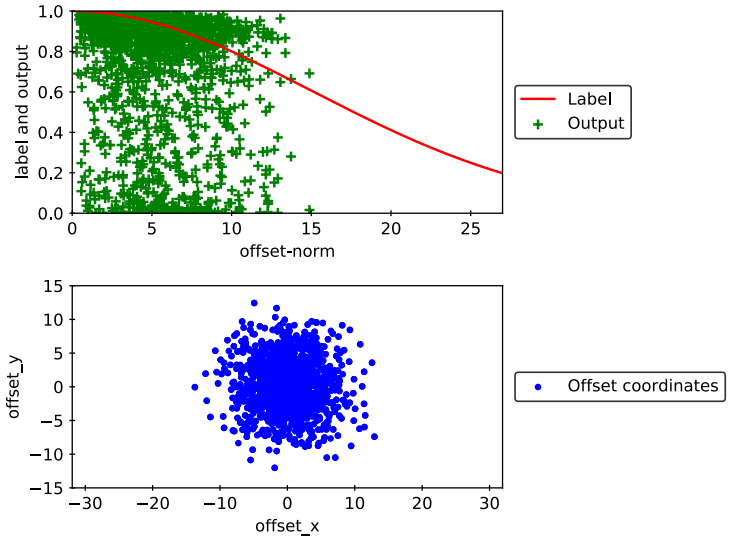
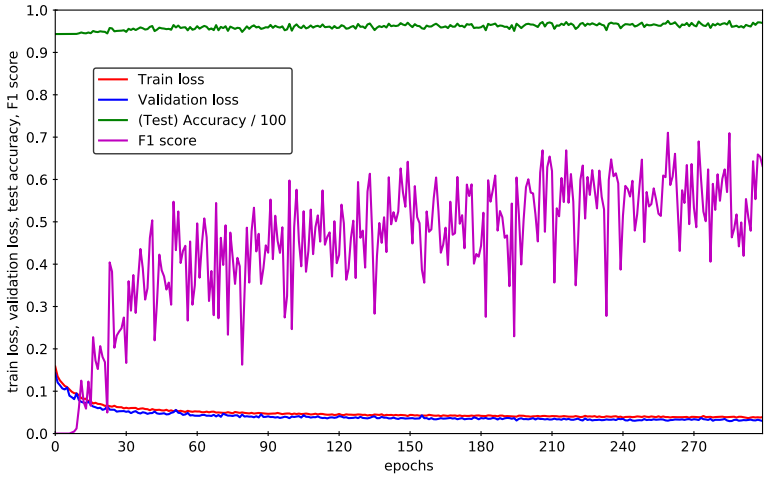


Figure 18: The upper plot shows the label and output values over the offset norm in the last epoch of training. The red line follows the probability density function of a normal distribution with $\sigma = 15$, normalized to 1.0 at offset 0. Each green point corresponds to the output value of a *bee* detail from the *detail_training* set. The green points would be expected to cluster close to the red line with perfect training. Instead, there is a cluster of values close to 0. The output values close to 1 show a slight tendency to decrease with increasing offset-norm, but they do not follow the curve as closely as expected. Further, there are quite a few apparently random values in the middle.
The lower plot again visualizes the chosen offsets, which were picked from the same normal distribution with $\sigma = 4$. Offsets with a larger absolute value than 15 either horizontally or vertically were discarded and redrawn.

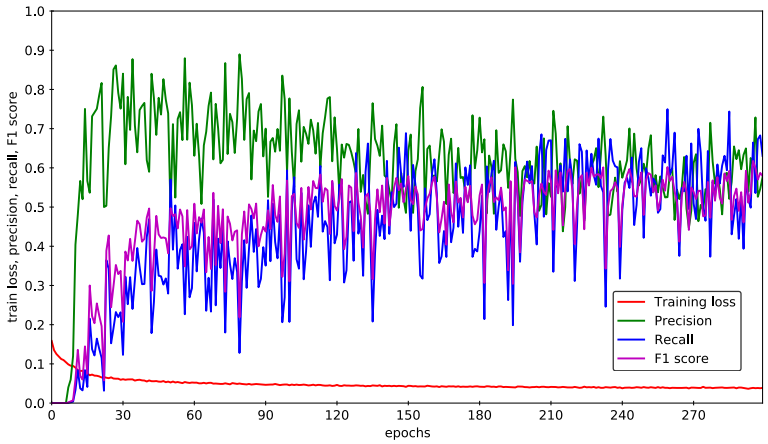
and still presents nicely centered detail images of bees in cells to the network, or negative detail images of another part of the honeycomb altogether, but nothing in between.
The testing on the *original_test* set visibly improved (Figure 19b), precision and recall level out to a similar value in the last 150 epochs, causing an improved F_1 score. However, while the recall slowly increases, the precision seems to still decrease in the last epochs. This can be an indicator that longer training could cause a further increase of this difference. Analysing the choice of the sigma for the label strength might be beneficial to stabilize the values.
So far, with about 60% correct predictions and a precision of about 0.6, the current network architecture produces the best results compared to the previous iterations.

5.2.2 Extended datasets

After seeing the big impact of changing the network architecture and the small impact of extending the amount of *notbee* details on the training it is still unclear what impact extending the whole dataset has on the detection quality. 50 images were marked additionally to the already used 100 raw images. The ratio of the new dataset is shown in Table 4. To compare the influence of the modified dataset to the influence of the modified implementation described in Section 5.2.1, this run was made without using a random offset or label strength. The threshold determined as 0.99 by the method described in Section 5.1 was used to test the network on both the extended *detail_test* set (Figure 20a) and the



(a)



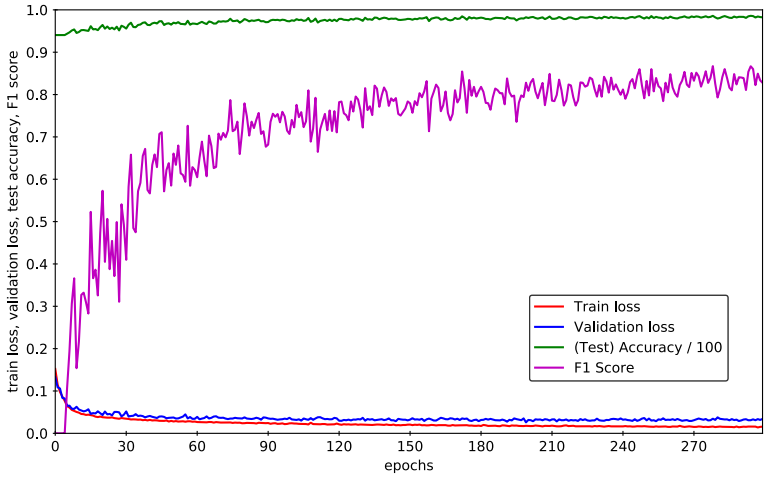
(b)

Figure 19: The network was trained for 300 epochs, using a random offset up to 15 pixels translation and offset dependent label strength with $\sigma = 15$. The best threshold was determined to be 0.96. This threshold was used for both graphs.
(a) Both train loss and validation loss start from a value of about 0.16, with the network tested on the *detail_test* set. The losses decrease to a value of about 0.05 in the first 20 epochs. In the remaining epochs they still decrease, this time much slower, converging 0.03. During the testing phase, the validation loss runs slightly under the train loss. The test accuracy starts from about 0.94 and slowly increases over the epochs to about 0.97. The F_1 score, starting at 0, strongly increases to about 0.54 in the first 50 epochs. It increases over the remaining epochs to about 0.64, while it is fluctuating with an amplitude up to 0.4.
(b) The same training results were used to test the network on the *original_test* set. In the first 15 epochs the precision strongly increases from 0 to about 0.75. After the peak, it slowly decreases until epoch 200 with a lot of noise. In the last 100 epochs of testing it seems to level out to about 0.58. The recall steadily increases from 0 to about 0.63. It is also heavily fluctuating with an amplitude of up to 0.4. The F_1 score increases from 0 to about 0.49 in the first 40 to 45 epochs. Afterwards, it increases at a lower rate to a value of 0.58. The amplitude of the fluctuations decreases over the epochs.

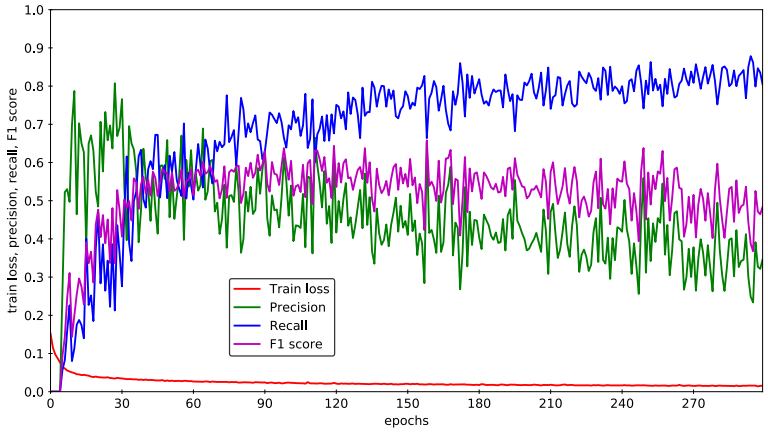
dataset	detail_training	detail_test
images	120	30
details (*bee*)	3,452	1,044
details (*notbee*)	66,000	16,500
details (total)	69,452	17,544
ratio	80%	20%

Table 4: The *detail_training* and *detail_test* set described in Section 5.1 were extended by marking 50 additional raw images and creating their corresponding details. This way, the amount of data in the datasets containing the raw images, *original_training* and *original_test*, was raised to 150 images in total. 45 of the new images were used to extract additional details for the *detail_training* set, while the remaining 5 raw images were used to extend the *detail_test* set. As a consequence, the ratio of *detail_training* and *detail_test* was changed to 80:20 instead of the previous 75:25 ratio. The number of batches in each training epoch increased to 348. In total, 86,996 details images were created.

extended *original_test* set (Figure 20b).
The results do not differ significantly to the results of the previous dataset without using translation and label strength, as described in Section 5.1 and 5.2. Still, each value, except accuracy, train loss and validation loss, is slightly better than while testing on the previous *detail_test* and *original_test* set. The percentage of correctly predicted bees in the raw images was raised from 31% to 34.6%. However, these changes could be caused by the random batch selection while training.



(a)



(b)

Figure 20: The network was trained for 300 epochs without using offsets but with enlarged datasets of 150 raw images in total. The best threshold to test the *detail_test* set and the *original_test* set was determined to 0.99.
(a) The network was tested on the *detail_train* set. The train loss and the validation loss both decrease from 0.15, the validation loss slightly below the train loss, to 0.05 in the first 15 epochs. During the remaining epochs, the train loss approaches 0.01 while the validation loss slowly decreases to about 0.03. The accuracy steadily increases from 0.94 to 0.98. In the first 46 epochs, the F_1 score increases from 0 to about 0.7 with strong noise. Afterwards it keeps increasing at a lower rate up to 0.83 with weakening noise.
(b) The same training results were used for testing the *original_test* set. The precision strongly increases from 0 to about 0.8 during the first 28 epochs. It then steadily decreases to about 0.35. In the first 45 epochs, the recall increases to about 0.7, starting from 0. For the remaining epochs it slowly increases to about 0.8, with a seemingly weakening fluctuation. The F_1 score increases from 0 to about 0.6 in the first 65 epochs. Afterwards, it decreases to about 0.48.

6 Conclusion and Outlook

In this thesis a ground truth dataset was created and a convolutional neural network was designed and trained on the dataset, to detect detail images of bees located in honeycomb cells. The best threshold for the binary classifier was determined for each modification of the network by using the *original_ test* set. Training runs with a higher rate of predicting false positive predictions on the test set required a high threshold to increase the precision of the network. Training runs which produced a higher rate of false negatives (missed ground truth markers) instead required a lower threshold. The modifications used were different data augmentations, an extension of the amount of *notbee* details, the usage of offset dependent label strength, and an extension of the whole ground truth dataset.
The results of the network slightly improved by extending both the amount of *notbee* details and the whole dataset. Overall, high accuracies on the test datasets were achieved quickly while training. Consistently high accuracy from the first training epoch could be an indication that the training and test datasets are too similar to guarantee the model generalizes well to unknown data. This hypothesis could be tested by training on and cross-testing with larger datasets from more diverse sources, such as the video recordings from 2014, 2015, and 2016, which were captured with a different setup. The extension to an even bigger dataset, also including data from the earlier video recording sets, would probably make the training even better. The same applies to the adjustment of the ratio between *bee* details and *notbee* details to the ratio of appearance in the raw images, as already mentioned in Section 5.1. However, dataset modifications do not affect the training as much as modifications of the implementation do. Apart from this, dataset modifications cause higher memory usage, and the so far used 64 gigabyte RAM would have to be increased.
Looking at the graphs, a closer analysis of the value used for the sigma in label strength seems to be beneficial. The network architecture using $\sigma = 15$ returned the best results, but there still is room for improvement. Choosing a sigma between $\sigma = 4$ and $\sigma = 15$ is expected to improve the results. Nevertheless, increasing the network complexity seems to be required to achieve really good label strength results.
Another way to use or to further improve on the results could be using the network on videos instead of single images and returning videos with corresponding heatmaps for each frame. With this information, behavioral patterns like a bees' length of stay in a honeycomb cell or how far the bee went into the cell could be recognized. These patterns could allow to classify behaviours such as cell cleaning, nursing, repairing and building new cells, which would give valuable information that could be used to identify the caste of a bee. The ground truth dataset is already prepared to be used for training the network on different depths of bees in cells by using different color markers as described in Section 3. However, the network architecture would have to be adapted to it.
An interesting avenue for further research would be extending the BeesBook localizer with the results of this work. This would enable it to keep tracking bees when they enter honeycomb cells, despite the non-visible tag.

References

[1] *BioroboticsLab*. URL http://berlinbiorobotics.blog/, visited on 10/18/2019.

[2] *Convolutional Neural Networks (CNNs / ConvNets)*. URL http://cs231n.github.io/convolutional-networks/, visited on 10/18/2019.

[3] *PyTorch*. URL https://pytorch.org, visited on 10/26/2019.

[4] *TORCH.NN*. URL https://pytorch.org/docs/stable/nn.html, visited on 10/18/2019.

[5] *torchvision.transforms*. URL https://pytorch.org/docs/stable/torchvision/transforms.html, visited on 10/18/2019.

[6] *Training a Classifier*. URL https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html#sphx-glr-beginner-blitz-cifar10-tutorial-py, visited on 10/18/2019.

[7] Alex Krizhevsky. *Convolutional Deep Belief Networks on CIFAR-10*, 2010. URL https://www.cs.toronto.edu/~kriz/conv-cifar10-aug2010.pdf, visited on 10/18/2019.

[8] Antonio Torralba, Rob Fergus, William T. Freeman. *80 million tiny images: a large dataset for non-parametric object and scene recognition*. IEEE, 2008. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.73.4858&rep=rep1&type=pdf, visited on 10/18/2019.

[9] Benjamin Wild, Leon Sixt, Tim Landgraf. *Automatic localization and decoding of honeybee markers using deep convolutional neural networks*, 2018. arXiv:1802.04557v2, URL https://arxiv.org/abs/1802.04557, visited on 10/18/2019.

[10] Brian R. Johnson. *Division of labor in honeybees: form, function and proximate mechanisms*. Behavioral Ecology and Sociobiology, 64:305–316, 2010. DOI:10.1007/s00265-009-0874-7, URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2810364/, visited on 10/18/2019.

[11] Chris Nicholson. *A Beginner's Guide to Neural Networks and Deep Learning*. URL https://skymind.ai/wiki/neural-network, visited on 10/18/2019.

[12] Connor Shorten, Taghi M. Khoshgoftaar. *A survey on Image Data Augmentation for Deep Learning*. Journal of Big Data, 6, 2019. DOI:10.1186/s40537-019-0197-0, URL https://link.springer.com/content/pdf/10.1186%2Fs40537-019-0197-0.pdf, visited on 10/18/2019.

[13] David Baracchi, Alessandro Cini. *A Socio-Spatial Combined Approach Confirms a Highly Compartmentalised Structure in Honeybees*. Ethology, 120(12):1167–1176, 2014. DOI:10.1111/eth.12290, URL http://onlinelibrary.wiley.com/doi/10.1111/eth.12290/abstract, visited on 10/18/2019.

[14] David Powers. *Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation*. Mach. Learn. Technol., 2, 2008. URL http://www.flinders.edu.au/science_engineering/fms/School-CSEM/

publications/tech_reps-research_artfcts/TRRA_2007.pdf, visited on 10/20/2019.

[15] Dhruba Naug. *Structure of the social network and its influence on transmission dynamics in a honeybee colony.* Behavioral Ecology and Sociobiology, 62(11):1719–1725, 2008. doi:10.1007/s00265-008-0600-x.

[16] Diederik P. Kingma, Jimmy Ba. *Adam: A Method for Stochastic Optimization*, 2014. arXiv:1412.6980v9, ulr https://arxiv.org/abs/1412.6980, visited on 10/20/2019.

[17] Fernando Wario, Benjamin Wild, Margaret J. Couvillon, Raúl Rojas, Tim Landgraf. *Automatic methods for long-term tracking and the detection and decoding of communication dances in honeybees.* Frontiers in Ecology and Evolution, 2015. doi:10.3389/fevo.2015.00103, url https://www.frontiersin.org/articles/10.3389/fevo.2015.00103/full, visited on 10/18/2019.

[18] Franziska Boenisch, Benjamin Rosemann, Benjamin Wild, Fernando Wario, David Dormagen, Tim Landgraf. *Tracking all members of a honey bee colony over their lifetime*, 2018. arXiv:1802.04557, url https://arxiv.org/pdf/1802.03192.pdf, visited on 10/18/2019.

[19] Jakob Mischek. *Probabilistisches Tracking von Bienenpfaden.* Master's thesis, Freie Universität Berlin, Berlin, 2016. url https://www.mi.fu-berlin.de/inf/groups/ag-ki/Theses/Completed-theses/Master_Diploma-theses/2016/Mischek/index.html, visited on 10/18/2019.

[20] Katarzyna Bozek, Laetitia Hebert, Alexander S Mikheyev, Greg J Stephens. *Towards dense object tracking in a 2D honeybee hive*, 2017. arXiv:1712.08324v1, url https://arxiv.org/abs/1712.08324, visited on 10/18/2019.

[21] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting.* Journal of Machine Learning Research, 15:1929–1958, 2014.

[22] Olaf Ronneberger, Philipp Fischer, Thomax Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*, 2015. arXiv:1505.04597v1, url https://arxiv.org/abs/1505.04597, visited on 10/18/2019.

[23] Pedro Domingos. *A Few Useful Things to Know about Machine Learning.* Communications of the ACM, 55(10):78–87, 2012. doi:10.1145/2347736.2347755, url https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf, visited on 10/18/2019.

[24] Sergey Ioffe, Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* CoRR, abs/, 2015. arXiv:1502.03167, url http://arxiv.org/abs/1502.03167, visited on 08/23/19.

[25] Thomas D. Seeley. *The Wisdom of the Hive: The Social Physiology of Honey Bee Colonies.* Harvard University Press, Cambridge, 1995. isbn 978-0-674-95376-5.

[26] Thomas D. Seeley. Honeybee ecology: A study of adaptation in social life. Princeton University Press, Princeton, 1985. isbn 0-691-08391-6.

[27] Victor Zhou. *Machine Learning for Beginners: An Introduction to Neural Networks*. URL https://victorzhou.com/blog/intro-to-neural-networks/, visited on 10/19/2019.

[28] Zhi-Yong Huang, Gene E. Robinson. *Worker-worker interactions mediate hormonally regulated plasticity in division of labor*. Proceedings of the National Academy of Sciences, 89(24):11726–11729, 1992. DOI:10.1073/pnas.89.24.11726, URL https://www.pnas.org/content/89/24/11726, visited on 10/18/2019.

Statutory Declaration

I declare that this thesis was written independently and only with the help of the indicated sources and resources. The thesis was neither submitted nor published in the same or similar form during another examination procedure.

Berlin, 31 October 2019

Sophie Zabel