Implementation

Problems and findings

Sieuwe Elferink
Intelligent Beehive

Introduction

For the Intelligent Beehive project I was tasked to make an module which could determine the health of the beehive by using only an Camera. To achieve this, research about beehive health and different detection technologies was done. After this research the findings had to be implemented. While research is needed in the real world it is always different than what the research says. That is why this document exists. This document consists out of a list of steps/problems during the implementation fase and how these steps where performed or how these problems where fixed.

Results

1- Population counter

The purpose of the population counter is to count the amount of bees in the beehive. The main outcome of the research was that to count the bees it is not needed to count every bee by itself. It is better to calculate the area of clusters of bees and then convert those area's to bee counts. So in order to achieve this a bee cluster detector which can detect a cluster of bees inside an image and give the area should be made.

Research

After having done research into what OpenCV algorithms I found an algarithm called cv2.FindCountours(). This function returns the countours of white areas in a masked image. These contours are areas with a certain size which is exactly what is needed for the population counter.

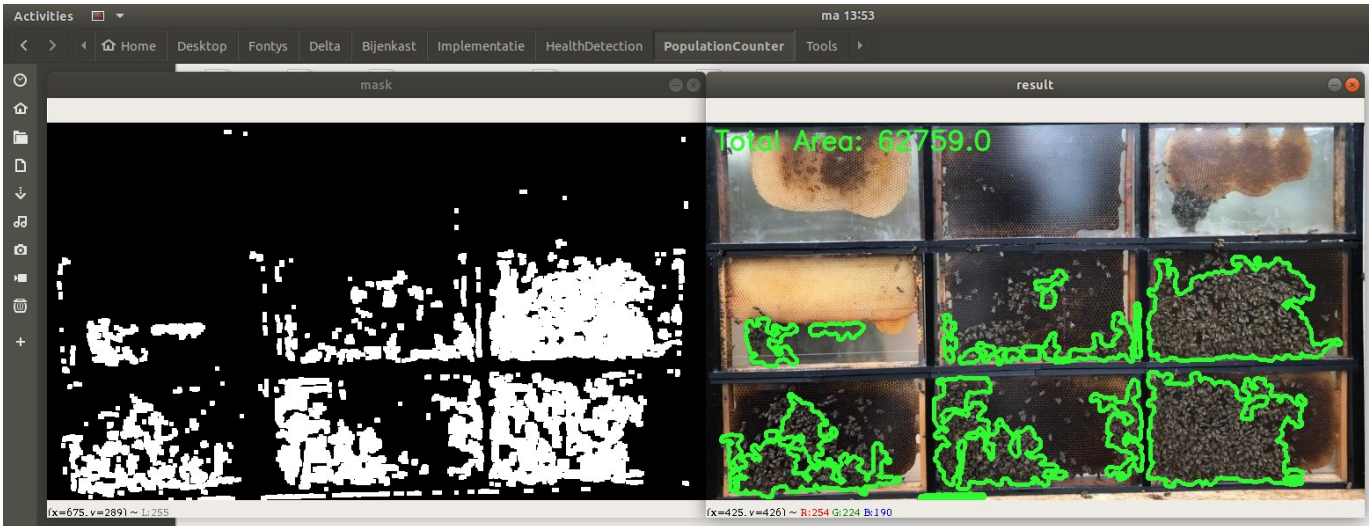
Implementation

After having done the research the first thing needed is an masked image where everything except the bee clusters is masked out. Because with this masked image which only shows the bee clusters, a contour of the bee cluster can be made.

1- Getting the HSV mask

The mask needed is an HSV mask which filters out all the colors between a certain range of HSV values. To find these HSV values a simple program downloaded from the internet can be used called HSVFilterPicker (can also be found in the health detection github). The only problem was that this program only takes an video and not an image. There was only an low quality Image of the beehive which was sent over WhatsApp. So instead of trying to rewrite the filter picker program a simple program which converts the image into a video was made using OpenCV and python. This program simply wrote the image every second to a video to create a video of that image. After this the HSV filter picker program worked perfectly and after a couple of hours tuning the filter the HSV filter values where found. These are [101, 57, 120] [108, 103,173].

2- Applying and improving the filter
After having found the filter it has to be applied. This can be done by passing the mask into the FindCountours() function. But to make sure the mask is working properly the Imshow function in OpenCV is useful. This function displays a window with the results of the mask. The mask can be seen on the right with the areas an the original picture on the left.



As you can see the mask works very well because the areas where the bees are are highlighted white and all the other areas are black. To make this mask work so well multiple different masks where tried with different HSV values. The final used mask uses these values [15, 26, 47] [26, 107, 88]. This mask is so different from the first one because the first mask was based on a low quality picture from Whatsapp which had a lot of reflection and compression problems. The new mask is made on an high quality image over Email. While some more images to test the program on would be handy, the stake holder did not respond on a question to make more images. But either way a single image is enough to create a solid program.

To make the mask work even better the CV function Erode was used. This function makes the separate little white dots bigger so that the bigger white dots connect with each other to create surfaces which the FindCountours() function can use to create contours from.

3- Filter areas
After having made a sufficient mask the FindCountours() function returns a list of contours with each having an area. Not all the areas the FindCountours() function returns are actually valid area. That is why some filtering is needed. The program loops over the list of returned contours and checks if the area is larger than 300. If this is not the case the contour is deleted. This is to make sure false positives are removed. After having done multiple tests the minimum area size of 300 gave the best results.

4- draw areas
For better debugging the areas are drawn on the screen using a green color. To do this the program loops over the list of accepted contours ans uses the draw contour function to draw the contour.

6- Split image
After testing was found that small clusters of bees where not picked up by the mask. This was because they where only a couple of pixels and so they where easily filtered out. To fix this the decision was made to split the image into a 3x3 grid so that every hive cell has its own image. You can see on the image below that the beehive also consists of a 3x3 grid. After creating an algorithm which splits the image, the images are individually analyses in a zoomed in size. This did improve the results a bit but because the testing image was not the best quality the zoomed in quality was very low. So if a better resolution image is used the performance increase will be even more.

7- calculate bee count
The total area of bees is now calculated by counting all the areas in the accepted contour list together. After having the total area the actual amount of bees has to be calculated. To do this the test image below was used.

The image goes trough the image splitter and each image with its analyzing result is shown like the image below.

This particular image shows an bee cluster with area 17229. This result is written down. After that the actual amount of bees is counted by hand using sight. This gave around 420 bees. After that the area of 1 bee was calculated by dividing the area by the amount of counted bees. This was done for every image as seen below:

area : bee count

2540 : 14 = 181

4337.5 : 30 = 144

17801.5 : 260 = 68

8761.5 : 75 = 116

11694.5 : 180 = 64

17229.0 : 420 = 41

average (181 + 144 + 68 + 116 + 64 + 41) / 6 = 102

Bee count formula = area / 102

The average area for one bee for all the images was taken which ended up with an area of 102 for 1 bee. So to calculate the amount of bees the area should be divided by 102. This gave 611 as result for the tested image above. This seems like an reasonable correct result.

2- Brood pattern

The Brood pattern AI is used to determine weather the Brood Pattern is healthy or not by training it on a large data set of healthy and unhealthy brood patterns. By learning the AI what is a healthy and an unhealthy brood pattern the AI can after the training determine by itself if it is healthy or unhealthy without having to program thousands of rules like traditional CV solutions need.

Research

First of all research was done on what the best detection models are and what the best way of training is. The best way for training in this situation is to use transfer learning. This is because with transfer learning you don't need a large data set to get accurate results (*which is very important especially in this situation like explained later in this document*). Also by using a pretrained model the difficult of training is decreased a lot which increases implementation speed. A list of pretrained models which can be used with transfer learning and that are build into Keras the framework used for this project can be seen below:

```
| VGG16
| VGG19
| ResNet50
| Xception
| InceptionV3         = Getraind op een heleboel verschillende dingen (1000 cl)
| InceptionResNetV2   =
| MobileNet
| DenseNet121
| DenseNet169
| DenseNet201
| NASNetMobile
| NASNetLarge
```

After having looked at all the models the InceptionV3 looked the most interesting, This is because it was pretrained on a lot of different classes which makes it very good in a lot of different situations because it can detect a lot of different features already.

Implementation

After having found the model and the way of training it was time to make it.

1- Data Set

The first part of training an AI is to get Data in what is called a Data Set. A lot of times a Data Set can be easily downloaded from the internet. But in this case nobody made a Data Set which has images of the brood pattern of a beehive with healthy or not healthy labeled on the image. Because of this the Data Set had to be made.

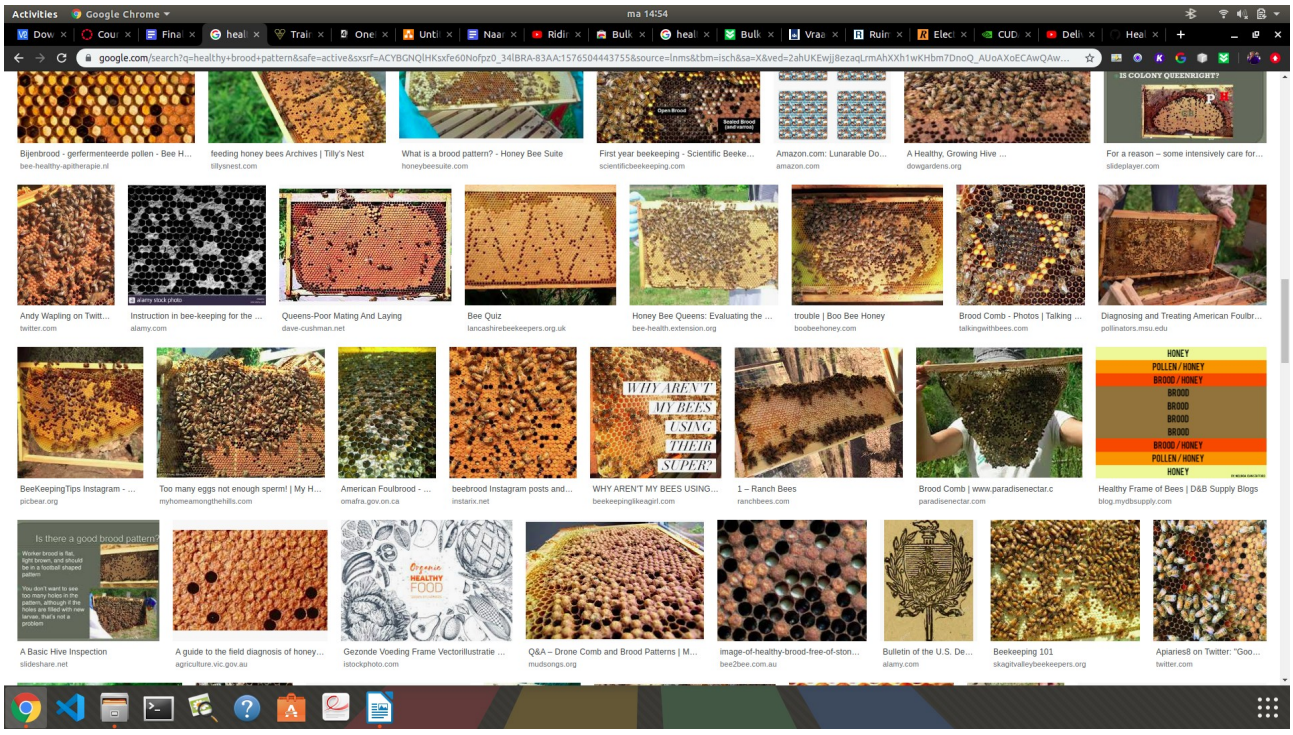
This is where the first problems came.

First off all there where not a lot of images of the inside of beehives, let alone images of the brood pattern which is a very specific part inside a beehive. And a lot of the images that where there where bad quality or from unusable angles.

Secondly when the phrase “healthy brood pattern” was searched on google images the results did not only return healthy brood patterns but also a lot of images of unhealthy brood patterns.

Thirdly there where also a lot of random other images like for example beekeepers working. This are also images we don't want to have in the data set because this will only confuse the network.

An image of the results can be seen below. Here you can see random images and also a mix of healthy and unhealthy brood patterns mixed while the search phrase was only for healthy brood patterns.



This made it impossible to use an automated down-loader to simply download all the images because then there would be a lot of bad images inside the data set which would decrease the models accuracy.

Eventually a dataset was made by manually selecting every photo and downloading it to a folder called “healthy” or a folder called “Unhealthy”. While this gave a high quality DataSet it was very time consuming. Eventually a DataSet of 100 images was made. 50 for healthy and 50 for unhealthy. This DataSet is called DataSetV1 in the Github.

Before DataSetV1 there was an earlier DataSet which consisted off the folders “healthy”, “AFB”, “queen problem” and “other problem”. Each folder containing images for that class. But after training an model I found out that classifying between all these classes was to complex. Also it was difficult to find images for all these classes. That is why it was changed from these 4 classes to only “healthy” or “unhealthy”

2- Training
After having made the DataSetV1 it was time for training. After writing an script inside Keras which transfer learns the last layers of the inception V3 it was time to run it. After a couple of bug fixes finally the first model was trained.

3- Bad results
After testing the model the results where really bad. The Accuracy of the model during training was around 97%. but when testing it was far below that. The model gave wrong results 70% of the time. It was better to guess the healthy of the brood pattern than to make the model predict it.

After this bad result a couple of different training techniques where tried with different epochs, learning rates and completely different scripts. But none of these gave a better result. A lot of the time the model got a bias for only one of the two health states. It the only gave “healthy” as result for all the images or the other way around. Also the fact that training took more than 3 hours on my laptop I decided to try a completely different approach.

4- New DataSet and other models
The first thing changed was to try a different DataSet using an auto down-loader. While this would give the problems explained earlier where the dataset would have a lot of bad images it was worth a try because the dataset could be much larger. After filtering out all the useless images which don't have anything to do with brood the pattern, the dataset called DataSetV2 consisted out off about 800 images. This was much better than the 100 images from DataSetV1.

After training InceptionV3 the result was still not great. The accuracy was around 40% which is still lower than randomly guessing. Also trying another model like ResNet50 or VGG16 did not improve the result.

This is actually a logical result because the DataSetV2 was really dirty. Inside the “Healthy” folder where a lot of images of unhealthy brood patterns and also the other way around. This confused the network because it couldn't understand what features are different because both classes had the same features because they both had the same type of images.

5- different framework and different models
After working on this for a couple of weeks I decided to change the framework called Keras to Tensor Flow. Tensor Flow had an script called “tensoflow for poets” which I used in the past to successfully train models. I used this script with the default parameters to train on DataSetV1. To my suprise this actually gave an very nice result with an accuracy of 82%. And after manually testing it on some random images it classified the images almost all the time correctly. This was a massive breakthrough. After changing the epochs, learning rate and making the DataSetV2 better for this framework by removing the validation folder because this folder was not used anyway I got a result of 91%.

After looking at the code of the script I found out it was not using InceptionV3 as its default model but actually MobileNet. This is an model based on InceptionV3 but made so that it works better in mobile use cases. The biggest change is that MobileNet has less parameters which can be trained to increase the training speed on slower mobile devices. This should make the model less accurate but the trade off so that it can be used for mobile is worth it.

So after changing the script to use InceptionV3 instead of MobileNet I ran the script. To my surprise the accuracy was around 40%. So again really low. Using DataSetV2 improved the accuracy a bit to 62%. But still much lower than the 91% of MobileNet on DataSetV1.

My theory for this unusual result is that because MobileNet has fewer trainable parameters it can train better on a smaller Data Set. This is because it can faster achieve the optimum classification accuracy because there are less parameters to tune. And thus if the Data Set is smaller it can train better than InceptionV3 which needs a large DataSet to train. That is why InceptionV3 had such bad results on DataSetV1. It was simply to small for InceptionV3. But for MobileNet it was large enough for good results.

3- Combining

After having trained a accurate Brood Pattern Ai and after having made a accurate population counter it is time to combine the two together. This was done by making a python script called “main.py”. This main file imports the function “Classify” from the brood pattern ai file and the “run” function from population counter file. The main file runs these two functions and then uses the results of these functions to create an estimation of the health status.

There is also an “main_demo.py”. This file does the exact same as the “main.py” but also shows the steps using OpenCV windows and delays to make the steps understandable for people who want to see the program work.

4- Trend Analysing

Is not made yet.

Sources

https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751

https://keras.io/applications/