BACHELOR PAPER

Term paper submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Engineering at the University of Applied Sciences Technikum Wien - Degree Program Computer Science

Activity tracking and monitoring of fish using TensorFlow and object detection

By: Stephan Mayrhofer

Student Number: 1610257092

Supervisor: FH-Prof. Dipl.-Ing. Alexander Nimmervoll

Wien, September 5, 2019

FH University of Applied Sciences
TECHNIKUM WIEN

Declaration

“As author and creator of this work to hand, I confirm with my signature knowledge of the relevant copyright regulations governed by higher education acts (see Urheberrechtsgesetz /Austrian copyright law as amended as well as the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I hereby declare that I completed the present work independently and that any ideas, whether written by others or by myself, have been fully sourced and referenced. I am aware of any consequences I may face on the part of the degree program director if there should be evidence of missing autonomy and independence or evidence of any intent to fraudulently achieve a pass mark for this work (see Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I further declare that up to this date I have not published the work to hand nor have I presented it to another examination board in the same or similar form. I affirm that the version submitted matches the version in the upload tool.“

Wien, September 5, 2019 Signature

Kurzfassung

In dieser Arbeit wird die Möglichkeit untersucht, Fische mithilfe von TensorFlow und Objekterkennung zu verfolgen. Mittels Raspberry Pis, AWS und TensorFlow wird in Kombination mit dem Objekterkennungsalgorithmus Faster R-CNN einem "faltenden" neuronalen Netzwerk vermittelt, wie Fische aussehen. Anschließend werden 319 Fischvideos mit sich ändernden Parametern wie Schwellenwerten und Lerndaten analysiert. Durch Optimierung dieser Parameter kann eine hohe Genauigkeit beim Zählen von Fischen in einem Becken erreicht werden. Das Verfolgen von Fischen mit dem vorgeschlagenen Aufbau ist ebenfalls möglich und liefert eine individuelle und durchschnittliche Geschwindigkeit, aus der Rückschlüsse auf den Hunger gezogen werden können, wodurch ideale Fütterungszeiten angedeutet werden. In Bezug auf die Gesundheit können Zebrafische erfolgreich als gesund, krank oder tot eingestuft werden, während Blaue Sonnenbarsche sich anders verhalten, was eine zuverlässige Unterscheidung mit denselben Parametern unmöglich macht. Schließlich scheint es undenkbar zu sein, Fische mit nur einer Kamera in verschiedene Größen einzuteilen. Eine mögliche Lösung ist jedoch bereits vorhanden.

Schlagworte: Internet of Things, Convolutional Neural Network, Object Detection, TensorFlow, Tracking

Abstract

This paper explores the possibility of tracking fish using TensorFlow and object detection. With the help of Raspberry Pis, AWS and TensorFlow in combination with the Faster R-CNN object detection algorithm, a convolutional neural network is being taught what fish look like. Next, 319 videos of fish are being analyzed with changing parameters such as thresholds and learning data. By optimizing those parameters high accuracy in counting fish in a tank is achievable. Tracking fish using the proposed setup is possible as well and will provide individual and average velocity, from which conclusions can be made in regards to hunger, thereby indicating ideal feeding times. In terms of health, zebrafish can successfully be classified as healthy, sick or dead, while bluegill behave differently, making a reliable distinction with the same parameters impossible. Finally, the classification of fish into different sizes appears to be unfeasible using only one camera, however, a potential solution already exists.

Keywords: Internet of Things, Convolutional Neural Network, Object Detection, TensorFlow, Tracking

Contents

1 Introduction

The first chapter of this paper describes the origin of the idea behind it as well as the problem it is trying to solve. Furthermore, the structure of the entire paper is layed out and relevant related work is presented.

1.1 Motivation

The area of land on our planet which can be used for agriculture is constantly decreasing. One of the reasons for this phenomenon is desertification of land, which is caused by weather extremes such as droughts and extreme precipitation, all of which originating from climate change. One of the ways of dealing with the decreasing availability of land is to make use of land which is inherently impractical for agriculture. Blue Planet Ecosystems [1] took that approach by developing a so-called Closed Loop Photo Protein Reactor (CLPPR), which later became part of Land-Based Automated Recirculating Aquaculture (LARA). This reactor simulates a real-life aquatic ecosystem, as it contains different units filled with water, all of which in turn are inhabited by various organisms.
In the first unit, microalgae are stored in glass tubes which are positioned in a very specific way, depending on the reactor's geographic location. The idea is to maximize the sunlight hitting these tubes in order to create ideal living conditions for the contained algae.
The second unit contains zooplankton, i.e. daphnia. Microalgae is the natural food source of zooplankton, which is why the rapidly growing algae population frequently gets pumped into the zooplankton unit for feeding purposes.
The same thing happens in unit three. Its population consists of either fish or shrimp, all of which naturally feed on zooplankton. The waste water created in this unit is recirculated back into unit one, providing a nutrient environment for the algae to grow in, thus closing the loop of the system.
The benefits of such a system are obvious: since sunlight in abundance is a benefit rather than the problem it would pose for traditional agriculture, it can be placed anywhere in the world - even in the desert, thereby making use of inherently unusable land. This opens up unforeseen possibilities, such as enabling countries like Dubai or Saudi Arabia to increase the food produced within their own borders and not having to heavily rely upon imports from other countries.
Building a CLPPR and placing it in the desert would not yield the expected amount of grown fish, however. If you were to leave the system unattended, it would go haywire - heavy mainte-

nance is required in order to make sure that water properties behave in the expected way, and that population growth of algae, zooplankton and fish follow the optimal course.

1.2 Problem

The maintenance of a Closed Loop Photo Protein Reactor (CLPPR) can - alongside physical maintenance such as replacing parts or cleaning - be grouped into four categories:

- Analysis of algae
- Analysis of daphnia
- Analysis of fish
- Control of system (valves, pumps, etc.)

Each of these categories can be split into a lot of subcategories, all of which are discussed in Chapter 1.4.
In this paper, a closer look will be taken at the analysis of fish in such a system. Ideally speaking, constant video surveillance of the tank would return information on its inhabitant's health, their velocity and quantity. At the same time, making use of sensors and Raspberry Pis, slightest changes of water quality can be detected and correlations between those changes and potential fluctuations of the fish population can be established.
For the purpos of this thesis, the evaluation of video footage will be the focus. Since counting myriads of fish by hand and making points on their health is humanly impossible, using some sort of computer-assisted analysis seems like an obvious choice to solve this problem.
As being able to distinguish between sick and healthy fish is crucial due to the fact that sick fish which are left in the tank for a long time would heavily impact the rest of the fish in a negative way, certain possible solutions immediately become obsolete. Using blob detection, for example, would most likely make it possible to find fish, however, there would be no way to tell apart the sick from the dead.
The same is true for motion detection. Only detecting fish in a given video or frame is simply not enough. The question therefore becomes whether it is possible to teach a neural network to identify optical properties of fish and, as a next step, have it count and track fish in a given video. Only by tracking fish for a long period of time - some species typically float on the same spot most of the time and only move through the tank once or twice a minute - is it possible to draw conclusions in regards to their health. Another benefit of artificial intelligence is that it should be able to differentiate between various species of fish and might even be able to make predictions on future health changes by observing certain behavior, i.e. the activity of individual fish during feeding.
In February 2019, Blue Planet Ecosystem was admitted into IndieBio's prestiguous four-month long startup accelerator program in San Francisco. During that time, all experiments required

for this thesis were conducted and all relevant data was collected. Due to the obvious constraints that are accompanied by young companies - mainly money and time - not all possible optimizations were explored, however, this thesis should pose a decent basis for tracking fish using object detection.

1.3 Structure

After presenting both the motivation behind this thesis as well as the problem it is trying to solve in Chapter 1, Chapter 2 is dedicated to Internet of Things (IoT). Here, various applications of IoT are explored, and especially the term Big Data is presented. Due to the relevance for this thesis, Raspberry Pis are looked at closely.
In Chapter 3, machine learning and neural networks are being observed. Most important for successfully answering the question asked in Chapter 1.2 is to figure out which kind of data is needed for ideal AI training as well as the functionality and properties of the machine learning library TensorFlow, which is why there is a subchapter dedicated to each of these topics.
With the theoretical description of relevant related topics completed, Chapter 4 revolves around the practical experiment conducted in order to reach a conclusion. First, all necessary preparations in terms of software and hardware are described. Next, the experiment is executed and relevant data is collected.
What's left is to present the experiment's results and its implications as well as discuss possible improvements and issues in Chapter 5, before drawing a conclusion from this paper's results in Chapter 6.

1.4 Related work

The following paragraphs will provide a short description of other scientific work related to the topics of this thesis.

- Lukas Schiermayer - *'Evaluate and control states of algae cultures by monitoring sensor data using Raspberry Pis'*:
  This paper focuses on the algae unit of Blue Planet Ecosystem's CLPPR. Using various sensors, water properties are constantly being monitored. The collected data is stored in a database, and by visualizing those data points, ideal growing conditions are being investigated.

- Sebastian Stricker - *'Using computer vision to detect and monitor Daphnia with a Raspberry Pi'*:
  This paper is akin to the one that is currently being read, as aquatic organisms are being monitored using computational aid. However, rather than looking at fish, this paper tries to

detect, count and classify zooplankton in a video in order to make projections on population growth and optimized growing environments. Also, rather than using TensorFlow and object detection, a mixture of blob detection and motion detection is being implemented.

- Johannes Fessler - *'Centralized controls of Raspberry Pi IoT devices'*:
  When confronted with hundreds or thousands of IoT devices, all of which needing to communicate with either each other or a centralized server, a lot of issues arise: How to synchronize said devices, make them accessible to each other but not the outside world, and seamlessly add or remove devices from this system. These and many other questions are being answered in this paper.

2 Internet of Things

The Internet of Things (IoT) describes an interconnected system of computing devices capable of communicating with each other over a network. The new key ingredient in comparison to the regular Internet is the possibility of machine-to-machine communication free from human interaction [21]. While consumers mostly benefit from IoT by consumer applications such as smart homes, a lot of money is to be made using IoT, as executives believe IoT to be the most important emerging technology, ranking it even higher than artificial intelligence and robotics. The following chapter will provide a closer look at the aforementioned applications of IoT as well as link this topic to the experiment at hand.

2.1 Applications

The following Figure shows a possible taxonomy of IoT. Regarding to it, IoT can be broadly classified into the categories health-care, environmental, smart city, commercial and industrial:

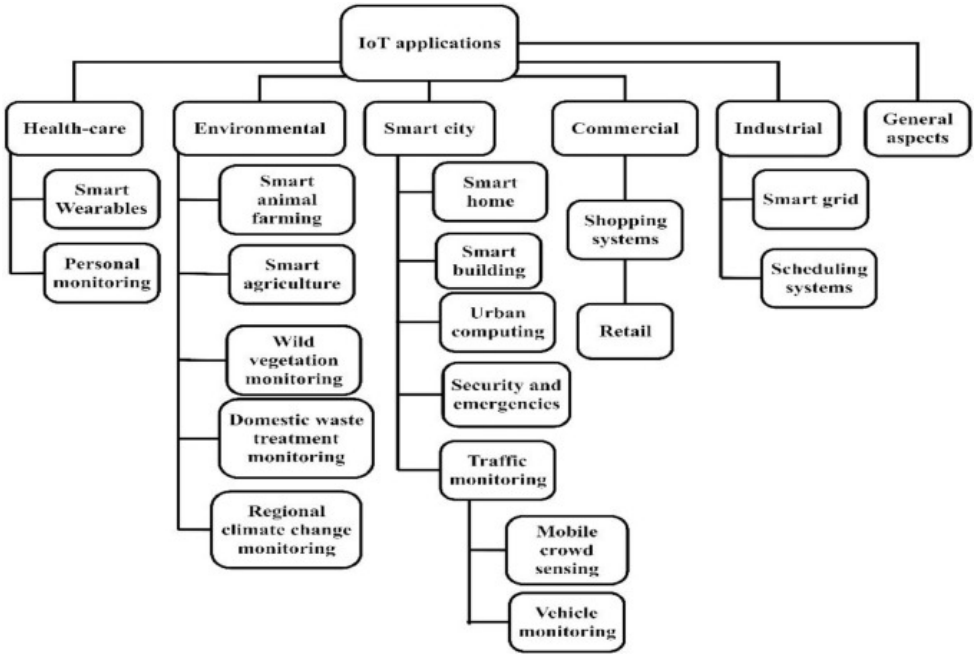

Figure 1: Taxonomy of IoT applications [30].

Health-care Among the things IoT devices do well are data collection and analysis for research and monitoring, both of which can be very useful in terms of health-care. Examples

for this so-called 'Smart Healthcare' are devices that remotely monitor patients' health as well as emergency notification systems. Blood pressure, heart rate and even pacemakers can be checked and authorities can be informed in case of malfunctions or measurements diverging from the norm. Some hospitals have even started introducing 'smart beds' which are able to detect when patients attempt to get up [9].

Environmental In terms of environmental monitoring, the most prominent IoT devices are ones that monitor air or water quality, soil conditions or other relevant properties of the surrounding [37]. By using sensors to gather as much information as possible, faster reactions to potential problems can be achieved. Additionally, in some cases, predictions about future challenges can be made, thus getting rid of the problem before it even occurs.

Smart city Smart city contains the subcategory most noticable for end consumers - smart home. Devices like smart fridges, self-controlling air conditions and security systems have been making lives easier for many people for years. In recent years, however, smart IoT devices have been used more and more in a larger scale, i.e. entire cities. One example is smart parking, which makes use of the fact that more and more cars are equipped with network capabilities as well as environmental sensors, thereby constantly gathering information on their surrounding. With an ever-growing population and a constant increase in cars on the road, smart parking tries to offer a step towards more sustainability [14].

Commercial With consumers being more and more informed about prices of competitors due to the availability of information about products or services, while at the same time being exposed to a large system of collecting data about shopping behavior, retailers are forced to satisfy their consumer's demands [2]. Making offers tailored to individual consumers rather than a group of people is expected by an increasing amount of customers and can only be achieved by becoming part of the data-collection scene, which can most easily be achieved by using IoT devices.

Industrial By introducing IoT devices to manufacturing processes, rapid production of new products as well as dynamic response to product demands and real-time optimizations of supply chain networks can be achieved [15]. Additionally, environmental data collection goes hand in hand with industrial IoT applications, for example in agriculture. Farming techniques can be automated by collecting information such as temperature, humidity or wind speed, thereby enabling farmers to take care of more land while being more efficient at it [26].

2.2 Relevance / Big Data

The amount of data we collect continuously keeps rising, so much so that at some point conventional means don't suffice anymore when it comes to evaluating and analyzing that data. For this amount of data, the term "Big Data" has been introduced [8].
While data becomes more and more important in today's businesses, the fact remains that many companies are simply not ready to handle the amount of data they are suddenly faced with [31]. As a solution, system architectures such as the CPS 5C level architecture have been developed to minimize the human role in data acquisition, analysis and prediction and thereby not only allowing for much more data to be processed but also to be more precise and get more reliable results while doing so [20].
For the purposes of this thesis, the amount of data collected would be humanly manageable as only one data set of several hundred data points would be looked at, however, once the experiment was concluded and production would start, data points were expected to come in much quicker than that, making it crucial to already consider big data challenges during the experimentation.
As such large amounts of data are being generated and processed by IoT devices such as sensors and Raspberry Pis, one established way of handling that data is using cloud computing [4]. This outsourcing of processing, storing and managing of data allows companies to monitor more data than they could on their own. By offering data storage and computing power on demand, providers like Amazon Web Services (AWS), Microsoft cloud services and IBM cloud services help their users to better handle and understand the data they are collecting.

3 Convolutional neural networks

Convolutional neural networks (CNNs) specialize in processing data with a grid-like topology such as images, which is why they are most commonly used to analyze visual material. Making use of several different layers, each neuron processes data of its receptive field to detect simple patterns like lines and curves. By combining the results, more complex patterns such as faces and objects can be recognized.

3.1 Architecture

Typically, CNNs are composed of three different layers: a convolutional layer, a pooling layer and a fully connected layer.

Convolutional layer The convolutional layer is the biggest and most important part of a CNN, as it does most of the computational work and greatly impacts the future output [27]. To put it simply, filters otherwise known as kernels are being defined. The input image is then separated into kernel-sized portions. Both the image portions as well as the kernel are being represented as a matrix. By multiplying the matrices of the filter and each portion of the image, a two-dimensional representation of the image is being created. This representation is called 'activation map' and holds information of the kernel's response of each spatial position of the image.

Pooling layer Once the convoltional layer has finished applying its filters, the pooling layer takes the resulting activation map for further processing. There are several pooling functions which can be applied, with the most popular one being `max pooling`, which reports the maximum output from certain areas of the image. Essentially, the pooling layer interprets the activation map using thresholds, thereby grouping the input into matches and mismatches.

Fully connected layer Finally, the matches found by the pooling layer are translated into a new matrix and then transcribed to a new filter representing the initial input image. Doing so, filters for different objects can be created. When confronting the CNN with a new image, it goes through all the aforementioned steps and, when applying the final filter, comes up with a score for each filter. The highest score should ideally represent the desired image.

3.2 Data needed for training

There are a lot of details to consider when choosing a fitting data set for what wants to be achieved. Moreover, with machine learning and neural networks being a comparatively new field with rapid growth, much of what is considered to be ideal is still being researched, which means that there is no established answer for many questions concerning this topic.
One of these questions concerns the amount of data needed to train a neural network as well as the properties of that data. While quite a few examples can be found online, the data deemed necessary for each experiment vastly differs, ranging from tens of images per class to tens of thousands.
One of these examples is the National Data Science Bowl, a science competition where the goal was to classify images of plankton with a given set of data [10]. In total, 121 different classes of plankton had to be distinguished using around 30.000 images, with some of the classes having fewer than 20 example images.
Two of the key concepts the winning team implemented were data augmentation and pre-processing. In terms of pre-processing, adapting the size of all images to one semi-consistent size was necessary, as the given images differed greatly in scale and aspect ratio. By approximating all images to the same size, similarities between images containing the same class of plankton would be easier to detect.
Data augmentation can be very helpful when it comes to artificially increasing the size of the available data. The following augmentation parameters were used in the case of the plankton classification:

- **rotation** of the image with a random angle between 0° and 360°
- **translation** of the image with a random shift between -10 and 10 pixels
- **rescaling** of the image with a scale factor between 1/1.6 and 1.6
- **flipping** of the image (yes or no)
- **shearing** of the image with an angle between -20° and 20°
- **stretching** of the image with a random stretch factor between 1/1.3 and 1.3

By augmenting the data at hand, the available data pool can be increased virtually endlessly, as all of these methods can be combined in various ways. The reason why augmentation of data might be necessary in the first place becomes apparent when considering Figure 2 and Figure 3 respectively.

While making quick progress and becoming better at identifying fish within the first couple of thousand steps of repetitve training, the learning curve quickly stagnates due to the low amount of training images. Additionally, the tube in the water was falsely identified as a fish, most likely due to a poor composition of learning data.

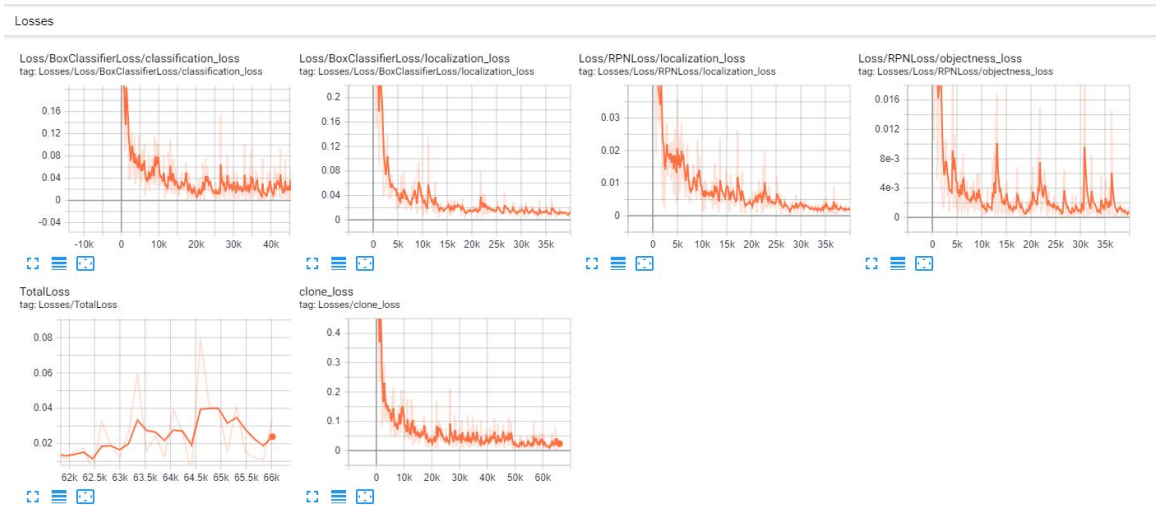

Figure 2: Learning curve of neural network over about 66.000 steps.

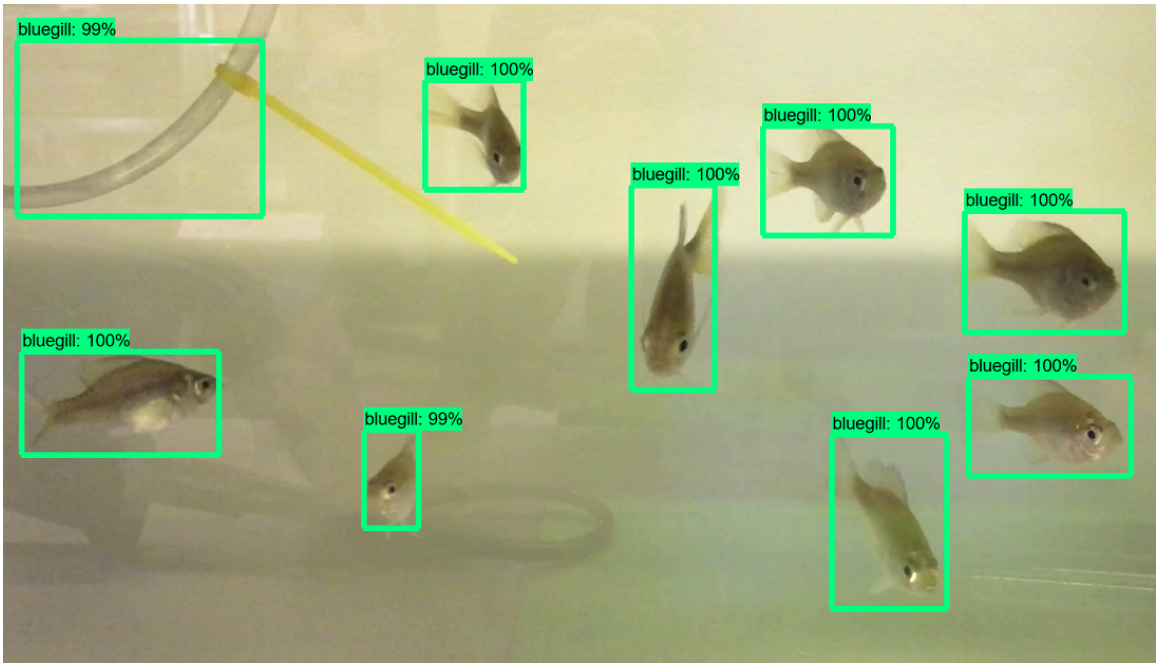

Figure 3: False positive after about 66.000 steps of learning with 120 images.

3.3 TensorFlow

TensorFlow is a free and open-source software library with a heavy focus on machine learning applications such as neural networks [17]. It was developed by Google, who released it in November 2015. It is available on Linux, macOS, Windows, Android and iOS. TensorFlow is still being worked on, as its latest stable release was on June 19, 2019.
Most developers working with TensorFlow choose to do so in `Python` or in `C`, as TensorFlow provides an application programming interface (API) for both. However, it is also possible to use TensorFlow in combination with `C++`, `Go`, `Java`, `JavaScript` and `Swift`, with third-party packages available for `C#`, `R`, and `Julia`, among others.

3.3.1 CPU vs. GPU

TensorFlow can make use of either the central processing unit (CPU) or the graphics processing unit (GPU) of the machine it is being used on. Given the power of today's GPUs, using them will yield much better results than using CPUs, as shown in this blog post [25]. This discrepancy is especially noticable when working with (high definition) image or video input, as a rise in file size leads to a lot more processing power needed.
Knowing this disparity is incredibly important, as it will affect the choice of which TensorFlow version to install. If possible, it is advised to make use of the GPU, however, the benefit of TensorFlow also running on CPU only is that it can be run on very low-level machines with poor graphics chips or non at all such as Raspberry Pis. In fact, guides on how to run TensorFlow on Raspberry Pis can be found on the TensorFlow website [38].
In terms of hardware requirements, `NVIDIA CUDA` [29] needs to be supported by the GPU in use if TensorFlow is to be run with it. A list of GPUs and their compatibility as well as their compute capability can be found on the CUDA website [28].
Another thing that needs to be considered is the version compatibility of TensorFlow and CUDA. Backwards compatibility is not a given, which means that the different versions need to be checked.

3.4 Choice of algorithm

With object detection finding its way into more and more fields such as video surveillance, anomaly detection or self-driving cars, it is no surprise that a large number of object detection algorithms exist, all optimized for different use cases. Among he most prominent and well-established ones are Region-CNN (R-CNN) (including Fast R-CNN and Faster R-CNN), You Only Look Once (YOLO) and Single Shot Multibox Detector (SSD) [32]. More recently, Region-based Fully Convolutional Network (R-FCN) has been added to the list.

R-CNN R-CNN is a CNN-based deep learning object detection approach, with the derivations Fast R-CNN and Faster R-CNN for faster detection rates.
As described previously, conventional approaches have filters slide over the input image to search for desired objects. While this approach works, it can come with certain challenges - variation in object size or ratio depending on the angle from and the distance to the camera can slow this approach down quite significantly.
For this reason, R-CNN uses selective search [24] to generate around 2000 region proposals on which to focus on [34]. Next, image classification can be done on the suggested regions rather than the entire image. Since the area was reduced drastically, a regular CNN approach can be taken to classify the chosen areas. Using regression, the regions can be further refined before calculating final scores for all leftover regions.
Unfortunately, R-CNN takes a huge amount of time due to the region proposals being very slow. With the selective search algorithm being fixed, no learning can happen during that stage, which means that nothing to be done about a poor selection of regional proposals.

Fast R-CNN Due to the aforementioned drawbacks, a new solution had to be found, and just like with R-CNN, Ross Girshick played a big role in finding it [16]. The crucial difference between R-CNN and Fast R-CNN is that rather than feeding the CNN with region proposals it is fed with the input image. From that, a convolutional feature map is generated, which helps identifying region proposals.
Since the convolution operations now happen only once and after the creation of the feature map rather than 2000 times for each image, training time can be reduced by a lot. When looking at testing time, there is a huge difference between excluding and including region proposals, making them the bottleneck for Fast R-CNN algorithm performance [33].

Faster R-CNN With region proposals being the bottleneck for Fast R-CNN and the selective search algorithm directly impacting the time needed to come up with those region proposals, selective search was what had to be improved next in order to optimize R-CNN even further. However, rather than improving the selective search algorithm, a new approach to region proposal was introduced by Shaoqing Ren et al. in this paper [36].
While the input image is still passed to a CNN, rather than using selective search, a separate network is used to come up with region proposals. The resulting improvement in speed finally allows for real-time object detection.

YOLO While all R-CNN algorithms use region proposals to break the input image down into separate pieces, YOLO - as the name suggests - only looks at the input image once and splits it into bounding boxes [23]. For each bounding box, the single CNN needed for this algorithm calculates class probabilities. All scores above a pre-defined threshold will be detected as the required objects.

While YOLO is magnitudes faster than R-CNN, it struggles with small and indistinct objects, such as one individual specimen in a school of fish.

SSD SSD speeds up the process in comparison to Faster R-CNN by eliminating the need for the region proposal network. SSD still manages to match the accuracy of Faster R-CNN by using lower resolution images, which also boosts the performance speed [40]. Using small convolution filters, location and class scores can be computed even quicker than using Faster R-CNN [18]. However, small objects can only be detected in earlier layers, where information is less accurate for classification.

R-FCN In region proposal network approaches, the fully connected layers do not share information with the previous layers, which slows the process down. Additionally, a lot of complexity is added by the fully connected layers. In order to circumvent this, the fully connected layers are removed and its processes are moved to before the pooling layer [22], generating score maps. After the pooling is done, all region proposals will make use of the same set of score maps and come up with an average score, which hardly takes any computational power [32]. Also, learning has been removed from after the pooling layer, which makes R-FCN even faster than Faster R-CNN.

4 Experiment

This chapter describes the practical experiment needed to answer the question asked in Chapter 1.2. It is split into two sections, firstly describing the preparations for the experiment, and secondly its execution.

4.1 Setup

Before the experiment can actually be conducted, quite a bit of setup is necessary. Some sort of aquarium - or tank - needs to be acquired to house the fish and, at the same time, offer an environment which allows to be monitored using a camera. Also, setting up a Raspberry Pi (or a similar IoT device) is crucial, as the camera will need to be connected to something in order to function. The impact of different camera settings such as brightness and exposure time must not be underestimated. Finally, in order to ensure automation of video analysis, cloud computing hosts such as AWS can be used.

4.1.1 Tanks

The tanks used for this experiment were custom-made using laser cutting technologies and acrylic glass. They are 50x32x32 centimeters (width x depth x height), with a glass thickness of 0.7 centimeters. The choice of thickness was made based on two factors: the glass needed to be thick enough to withstand the water pressure while being translucent and allowing camera recordings through the glass.
For the purposes of this experiment, the tank was a part of a closed loop system such as the one described in Chapter 1.1. This means that it was connected to both an algae tank as well as a zooplankton tank. As a result, wide variance in water color was observed during the course of the experiment. Over time, the water color changed from regular water color to a dark yellowish color, making it difficult even for human eyes to spot fish in the tank reliably.
In order to get rid of as much light reflection as possible, the tank was covered with regular white paper on all sides except the front, where the camera was placed. Additionally, it was surrounded by a cardboard box, so that as little outside light as possible could interfere with the video footage.

4.1.2 Raspberry Pi

In order to capture the video footage necessary for this experiment, a Raspberry Pi 3 B+ and a OV5647 Raspberry Pi Mini Camera Video Module were used. Using crontab [7], a file used to schedule certain events executed on an operating system, recording intervals were set to 45 minutes. Due to the fact that the entire setup was located within a working space, the lights were turned off between 10pm and 8am, which is why it made no sense to keep recording during that time. An external light source was introduced to test and see whether it was possible to keep recording at those times, however, bluegills - the species used during these experiments - react very strangely to light, especially if the light is very bright (which is usually the case if the light source is located close-by). With an LED strip attached to the tank, the fish would immediately start grouping together in a corner of the tank, thus making it much harder for the algorithm to distinguish between the different individuals.
In order to use the Raspberry Pi camera, no external packages need to be installed. In any case, running `sudo apt-get update` and `sudo apt-get upgrade` is recommended when first booting a new Raspberry Pi. The Raspberry Pi 3 B+ comes with Raspbian 9 (Stretch) already installed. It includes a lot of pre-installed packages, which is why only `boto3` had to be added manually. This was necessary to connect to AWS S3 buckets, the data storage of choice for all collected videos and data.

4.1.3 Camera

The camera used for this experiment is an OV5647 Raspberry Pi Mini Camera. It is capable of recording 1080p video footage with 30 frames per second and 720p video footage with 60 frames per second. It was used to create the image data set for the neural network as well as to record the videos of fish which then were evaluated.
The setup of the camera is relatively straight-forward. After connecting it to the Raspberry Pi, the camera was strategically placed in front of the tank in a way that allowed a maximum coverage of the tank's content without including too much of the side areas, which usually showed the reflection of close-by fish. Since the algorithm can't tell a reflection from a real fish, it was crucial to keep the amount of reflection at a minimum for an ideal detection rate.
Using gaffer tape and a small metal grappler, the camera was grounded at a specific spot. In terms of settings, only a few things needed changing. Firstly, in order to keep learning times of the neural network relatively low, the camera's resolution was decreased from 1080p to 720p for all pictures taken. This more than halved the amount of pixels, thus greatly speeding up the learning process.
In terms of video footage, 1080p did not seem to improve the detection rate of the algorithm, which is why it was reduced to 720p. This way, file size was decreased significantly, thereby also speeding up the upload time to the AWS S3 file server. 60 frames per second were deemed unnecessary, and after some testing the recording settings were changed to 720p and 25fps.
The only other really relevant thing to note is that it is important to choose the right exposure

time before starting the actual recording. If this is disregarded, the first one or two seconds of the video might be black, meaning that valuable recording time is wasted.
Ideally speaking, at least two cameras would be used simultaneously in order to better determine the position of each fish in the tank, thus drastically simplifying the calculation of their size. Unfortunately, due to the circumstances, there was only a limited amount of time as well as restricted amount of space in order to conduct all the experiments, which is why priorities had to be set. In this case, count, velocity and health were prioritized over classification of growth, which is why approaches such as the one proposed in this thesis [5] were considered, but never explored any further.

4.1.4 AWS

The aforementioned setup resulted in file sizes of about 20MB per recorded video with a video length of 30 seconds. With 18 videos taken each day (one video every 45 minutes between 8am and 10pm), storage size needed to keep all files would increase by about 350MB every day, which meant that some sort of file server was advised.
In this case, AWS S3 buckets were chosen to store the videos. Since not only the input video (the video taken by the Raspberry Pi camera) but also the output video (the video created using TensorFlow and CV2) was of relevance and needed to be stored, several buckets were set up on AWS S3.
Additionally, AWS EC2 instances were used to run the analysis of the videos. Initially, the author's laptop was used for analysis, however, evaluating 30 seconds of video took anywhere between four minutes and thirty seconds and seven minutes and thirty seconds, depending on the current workload of the machine.
While the laptop used a GTX 950M graphics card and an Intel Core i5 7200U @ 2.50GHz CPU, the chosen AWS EC2 instance (a p3.2xlarge [35]) was equipped with a Tesla V100 GPU with 16GB memory and 8vCPUs. This upgrade resulted in a decrease of evaluation time of approximately 400%, as the evaluation of each file went down to about 60 seconds. This included the analysis of a given video, drawing rectangles around each fish and tracing each detected fish with a black line, saving the new file and converting it using the multimedia framework `ffmpeg` to decrease file size, create a JSON file out of the detected coordinates, uploading both the original file and the newly created file to the AWS S3 bucket and deleting the files locally.
In terms of TensorFlow, the AWS EC2 GPU instance was almost usable straight away. Both TensorFlow CPU and TensorFlow GPU were pre-installed, including different versions of Python and the latest release of Conda. All that was left to do was to copy the directory containing TensorFlow models, inference graph and training data which was used for this thesis to the AWS EC2 GPU instance [13]. After running the following commands from within the `models/research` folder and the `models/research/slim` folder respectively, everything was taken care of and the video analysis could be executed without any errors:

- `python setup.py build`

- `python setup.py install`
- `pip install -e .`

4.1.5 Image Labeling

In order to teach the CNN what objects to look for in the input videos, it had to first be trained with images of that object. As stock images of bluegill proved to be difficult to find - especially in larger quantities - and better results typically were achieved when learning data and the data to evaluate were similar, a decision was made to create the training data using the same setup that would later be used for creating the input videos.
However, this also meant that those images had to be labeled, i.e. the parts of each image containing the desired object had to be marked, so that the CNN would be able to distinguish between bluebill and non-bluegill. One way of feeding TensorFlow with such a learning data pool was to add an eXtensible Markup Language (XML) file to each image containing the coordinates of each object and its class name - in this case, only bluegill.
This meant going through 490 images and marking each visible bluegill individually. While there would have been ways to automate this process (for example by using some other sort of artificial intelligence to identify and highlight bluegills in each picture), rather than outsourcing the labeling only to go through each picture later to check for mistakes, the labeling was done manually to ensure optimal learning data.

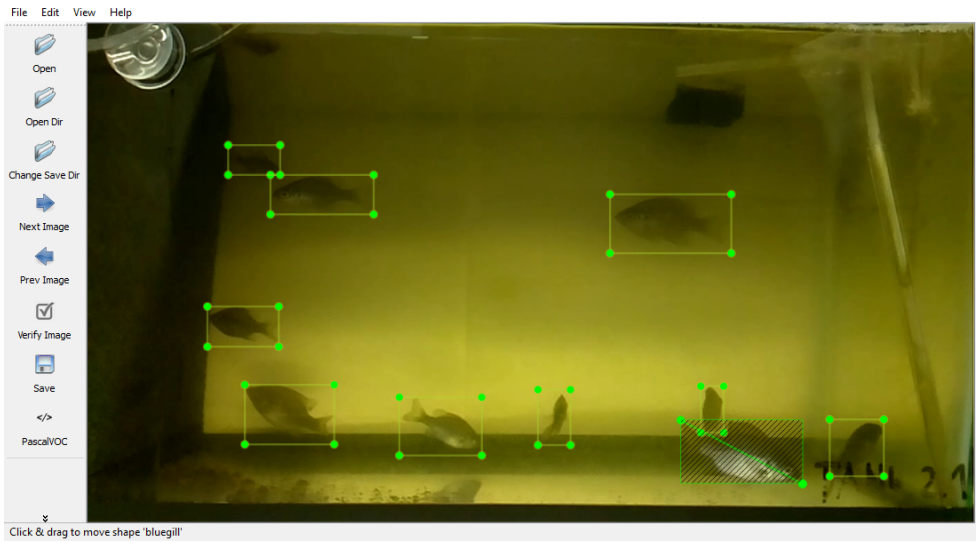

Figure 4: Example of manually labeling an image by drag & dropping a rectangle over each distinguishable bluegill.

4.2 Execution

Due to the choice of manually labeling the data pool, an approach had to be found that promised good results with a comparatively small amount of learning data. While it is usually recommended to use thousands or even tens of thousands of images for training (i.e. to solve the famous ImageNet challenge [19]), great results have been achieved using only a handful of images per class [10].
The basis of the algorithm used for this thesis comes from GitHub [13], where a user called `EdjeElectronics` (Evan Juras) published the source code as well as a tutorial on the required setup. It was chosen due to the many similarities to what this paper is trying to achieve - a small learning data pool consisting of manually labeled images, and the use of the Faster R-CNN object detection in algorithm in combination with TensorFlow.
This section describes the execution of the experiment, meaning that it depicts the necessary changes made to the algorithm from GitHub, how different values and properties of the input video were extracted and what those values were used for.

4.2.1 Acquisition of relevant data

Since all deliberations made for this thesis are based on the system depicted in section 1.1, the following values were selected as relevant:

- *Count*:
  Constantly counting all fish in the same tank might seem unimportant at first glance, however, it serves as some sort of validation of the algorithm's accuracy. As all other values rely on a dependable detection of fish across the entire video, having this parameter as a reference for how well the algorithm is doing at a certain point seemed very useful (especially since the real amount of fish in the tank were common knowledge). Also, a change of number of fish in the tank was not expected, which made this parameter even more viable as an indicator for the algorithm's precision.

- *Velocity / Activity*:
  An interesting aspect of fish is how much and how quickly they're moving, as it allows deductions about their health. Especially in a feeding scenario, (hungry) fish tend to move around much quicker than normally, as they are trying to collect all the fish feed in direct competition with all other fish in the tank. This means that, by choosing different feeding intervals, the average velocity of all fish in the tank should serve as an indicator for how hungry the fish as a whole are at the point of feeding. Ultimately, ideal feeding intervals should be found this way, so that fish are not overfed while not being hungry at the same time.

- *Health*:
  One of the most important things to find out is whether a fish is sick or even dead, as

dead fish can rot, causing ammonia to increase, thereby harming other organisms in the tank. While it is difficult to deduce the behavior of a sick fish (especially since different species of fish might behave differently when sick), telling a dead fish apart from a healthy one should be quite simple: a dead fish should not move anymore (if the slight movement caused by the current in the tank is ignored). Unfortunately, it is not quite as simple as that, as some fish species - such as the bluegill used for this experiment - often times don't move for many seconds at a time, even when healthy. This means that different ways of detecting dead fish might need to be explored to prevent false positives.

- *Classification / Growth*:
  Just like with algae and zooplankton, tracking fish population growth seems very useful, not least because fish need to reach a certain size before being sold. Ideally, fish would be classified into various size groups (i.e. small, medium and large) to see how long it takes an average fish to grow and to enable the grower to make predictions about when and how much fish they will be able to sell.

4.2.1.1 Count

TensorFlow returns an array of scores out of the box, indicating the percentage of security for each detected object/fish. This means that it is as simple as defining a threshold for that percentage and then counting all elements in the returned array that are above that threshold.

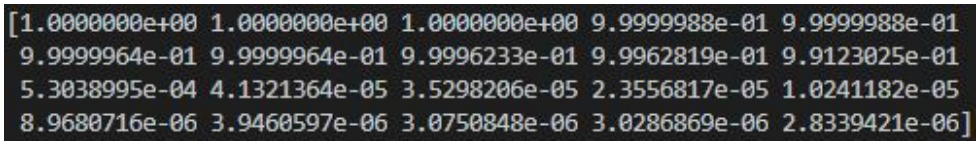

Figure 5: Output of array containing percentage of security for each detected object.

In the figure above, for example, ten fish were detected with a certainty of over 99% each. The array has been shortened to a length of 20, as it is sorted from biggest to smallest, and the eleventh entry is already as low as about 0.05%. In many frames, however, whether or not a detected object is a correct match or not is not as obvious as in this example. Especially when the object in question is moving a lot and is therefor presented from various angles, certainty can drop significantly. This means that the number of correctly detected objects will shift depending on your threshold. If the object in question is supposed to be tracked (i.e. found in every frame), the threshold might have to be lowered by quite a bit. In this case, a threshold of 97% was defined, after looking at score arrays of various frames and using different input videos to get an indication for the accuracy of the algorithm as a whole.

4.2.1.2 Velocity / Activity

With the fish already successfully being detected, calculating their velocity becomes possible. By saving the normalized coordinates consisting of x, y and z values (with z always being 0) of each detected fish into a list for each frame and comparing the current frame's list to previous one's at the end of each frame, an estimation of tracking all fish is possible.
It is a given that fish can only swim a certain distance over the duration of one frame (in this case, videos were filmed with 25 frames per second). This maximum distance of travel has been defined as 0.05 in a normalized tank of 1 by 1 dimensions after observing one single fish in a tank and calculating the longest distance of coordinates between any two frames.
Assuming that this is true, it can also be assumed that if the distance between coordinate 1 in frame 1 and coordinate 1 in frame 2 is greater than that threshold, the two coordinates must not belong to the same fish. By looping through all coordinates from frame 1 and comparing them to all coordinates from frame 2, it is possible to tell which coordinates don't match.
What is much more difficult is to find out which coordinates do match - after all, if the detection algorithm is working well, there should be a matching set of coordinates for each frame. Logically, the most likely scenario is that the coordinates which are closest together usually belong to the same fish. While this might not always be true, it certainly is a good estimation. As the setup only allowed for one camera, thereby limiting the coordinates to two dimensions, calculating the distance between any two coordinates was much less precise. Still, the desired result was to get an estimate rather than exact values, which is why this approach was deemed appropriate.
Another thing that needed to be considered is the fact that sometimes the algorithm would detect a different amount of fish in consecutive frames (i.e. 9 fish in frame n and 11 fish in frame n+1). In order to not correlate any set of coordinates to coordinates from the next frame twice, which could happen if two fish were swimming very closely together, thus returning the same coordinates as the closest ones, the list containing less coordinates was looped through first, so that there would always be enough coordinates to choose from in the other list.
Once the coordinates were matched, the distance between them could be calculated using the Euclidean distance approach, meaning that the square root of the sum of the squared differences between the x, y and z coordinates was the sought distance.

$$d(a,b) = \sqrt{(a_1^2 - b_1^2) + (a_2^2 - b_2^2) + (a_3^2 - b_3^2)}$$

Figure 6: Euclidean distance between a and b in three-dimensional space.

Since the coordinates were already normalized, simply adding up all calculated corrdinate distances would result in the sum of travelled space of all fish over one frame. Dividing by the amount of detected fish would then give the average fish velocity.

4.2.1.3 Health

Having calculated the velocity of each fish as well as the average velocity of all of them combined, making deductions regarding their health was now possible. It is important to note, however, that different species of fish will behave differently when being sick. For instance, while most of this experiment was done using bluegills, the fish used in the very beginning were zebrafish. Zebrafish are much smaller than bluegills, and at the same time much more active. While bluegills tend to be very still for multiple seconds at a time, only to then quickly move for a second or two before becoming still again, zebrafish basically constantly move around. When sick, however, zebrafish tend do move much less and much slower than their healthy cohabitors, which is why having calculated the velocity of all fish was very useful.

Sick fish detection By first calculating the average velocity and then comparing that to each individual velocity, anomalies in behavior could be detected. By having an actual sick fish in the tank amongst healthy ones, reasonable thresholds for how much slower than the average a sick fish usually moves could be ascertained. It could be observed that a sick individual would move with less than 25% of the speed of the average fish, which is why 0.25 was chosen as a threshold for the ratio between the calculated velocity and the average of all velocities.
To increase precision, two additional factors were considered. First, the standard deviation for all velocities was calculated and included in the formula for the detection of sick fish. Secondly, fish would have to be detected for several frames in a row in order to be considered sick or dead. This was implemented to help the fact that sometimes a fish would be in the process of turning around at the very beginning of the detection, meaning that its velocity would be comparatively low at that time. In that case, the fish would show up as sick right away, even though it wasn't. By not starting the calculations right away, this could be circumvented. For this experiment, a fish had to be detected for ten frames in a row before calculating whether it was healthy or not. Given that sick and dead fish tend to be detected very reliably due to the fact that they hardly move, this seemed like a logical addition to the algorithm. At the same time, fish that were constantly lost by the detection algorithm were never sick or dead, because in order to be lost, they had to move a lot.

```
1     # Velocity of the current fish
2     vel = calculate_velocity_from_line(line)
3
4     # If velocity is smaller than the average times the calculated
          threshold (0.25) - standard deviation, a sick fish is detected
5     if vel < (avgVel * float(CONFIG['Thresholds']['ThresholdFishSick'])
          - stdev):
6         # print("Fish probably sick")
7         sickFishCoords.append(line[0])
8         sickFish += 1
```

Code 1: Code to detect sick fish

Dead fish detection Distinguishing between sick and dead fish was more complicated than simply lowering the aforementioned threshold. The reason for this is, that dead fish still move quite a bit, simply because of the natural current within the tank (mainly created by other swimming fish).
For this reason, a different approach was taken. While a sick fish would move very slowly, it would still make its way around the tank eventually. In contrast, a dead fish will always stay more or less at the same spot, with slight movements in either direction due to the current. Additionally, a dead fish will almost always be detected by the TensorFlow algorithm, as it is virtually static. By not only tracking the distance each detected fish is travelling, but also the area of the tank it traversed, it quickly became apparent that the minimum and maximum coordinates of a dead fish would always be very close to each other, while those of a sick fish weren't.
Again, a threshold needed to be defined, this time for the distance between minimum and maximum x/y coordinates a dead fish would not overcome.

4.2.1.4 Classification / Growth

With TensorFlow already drawing rectangles around the outline of each detected fish, calculating the size of every individual fish could be as easy as multiplying length and height of each of those rectangles. However, using only one camera, as the fish moves through space, the rectangle moves with it. When looking at the camera, a fish will need a much smaller rectangle to be captured than when looking side-ways. The same is true for fish that are close to the camera versus fish that are far away from it - being close to the camera will make every fish appear much bigger. Additionally, when a fish is in the process of changing direction, meaning that its body is L-shaped, the rectangle around it will contain a lot of space which is not occupied by the fish.
For these reasons, it is the opinion of this paper's author, that calculating the size of fish in the tank with only one camera is not possible without a huge margin of error, thus making the attempt impractical.

Using two or more cameras at the same time would definitely come with some challenges, though. First of all, one Raspberry Pi can only trigger one camera at a time, so the concurrent recording of two videos needs to happen using two different devices. Once this has been achieved, the calculation of the object's position needs to be taken care of. A solution to this issue has been explored in this paper [5], where dual synchronized orthogonal webcams were used as video source and a mathematical model that accounts for projection errors served as a basis for calculating the fish's size.

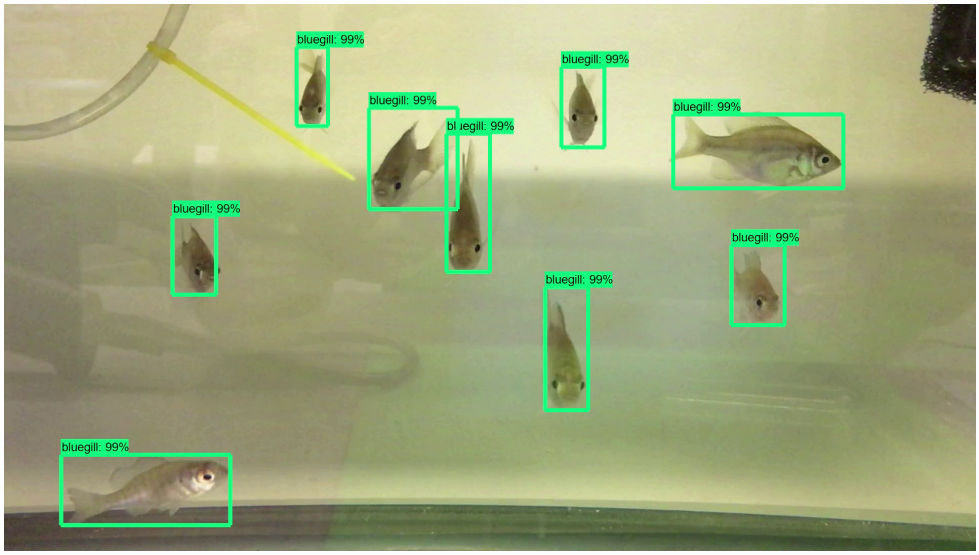

Figure 7: The top right and bottom left fish appear the same size due to their distance to the camera, however, the top right fish is almost twice as big in reality.

4.3 Optimization

After successfully extracting all relevant data described in Chapter 4.2.1 (except for growth due to the camera restiction), optimization of various parameters could start. This includes thresholds as well as the inference graph which was created using a particular set of learning and training images.

4.3.1 Inference graph

The inference graph will change depending on the set of images the neural network is provided with. Different sets of training and/or testing images will result in distinct inference graphs, therefore yielding varying outcomes.
The obvious question is which set of images will create the inference graph with the most accurate results. Unfortunately, it is hard to predict the potential implications a change in the images provided will have, which is why the best course of action seems to be to test a handful of inference graphs. According to the author of the base algorithm this thesis followed [13], the following guidelines should be observed:

- Have at least a total of 200 images.
- Make sure the file size as well as the resolution is not too big (200KB and 720x1280).
- Split the image set into roughly 80% for training and 20% for testing.

While 200 images might seem insufficient, the literature does not provide a concrete number of images necessary for training. Some examples can be found using thousands of images in

their data sets [39] [12], whereas others use less than one thousand. As the author of the introduced tutorial managed to create a card classifier with as little as 311 images and this thesis managed with 375 images in the end, the number of images required seems to depend on the underlying usecase. As proven during the National Data Science Bowl [10], even less than 20 images can suffice for a single class if done correctly.
In terms of splitting the images into training data and testing data, the ratio depends on the size of the image pool. In general, it is recommended to use the suggested 80/20 ratio for relatively small image pools, while a ratio of 99/1 is recommended for large amounts of data [11].
Regarding the file size, high-quality images contain more information but require more computing power to being processed, which is why in most cases the file size and aspect ratio of learning data is artificially reduced [3]. Ideally speaking, the images used for training should be comparable to the ones the neural network should later evaluate, since the objects need to look similar in order to being detected.

4.3.2 Thresholds

For the purposes of this thesis, a configuration file was created containing a large number of adjustable booleans and thresholds. Some of them regarded only the video output of the algorithm, while others directly impacted the result by adapting values used in the calculation.

```
1   [Thresholds]
2   # security needed to detect as fish [0 - 1] =
3   thresholdfishdetected = 0.97
4   # speed of sick fish compared to average [0 - 1] =
5   thresholdfishsick = 0.25
6   # minimum frames needed to decide if fish is dead =
7   thresholddeadfishframestoconsider = 10
8   # movement threshold to detect fish as dead [0 - 1] =
9   thresholddeadfishminmovement = 0.035
10  # count detection rate needed [0 - 1] =
11  thresholdfishmaxfoundlikely = 0.05
12  # distance a fish can swim in one frame [0 - 1] =
13  maximumfishdistance = 0.05
```

Code 2: Extract of the configuration file, showing the most relevant thresholds.

thresholdfishdetected This is one of the most important thresholds, as it decides whether or not the detected object gets counted or discarded. Basically, the array shown in Figure 5 is compared to this value, and all entries greater than or equal to it are counted as fish, while all others are not. While the security of detection for most fish usually is above 95%, especially when confronted with weirdly bent fish (i.e. when changing swimming direction) the percentage can drop to below 50%, which is why it might be useful to lower this threshold accordingly. However, unless the image is completely free of interferences (in short - any other objects) the

algorithm is bound to detect some of them falsely as a fish when lowering the threshold by too much. This means that the threshold should only be lowered if it is more important to detect all fish all the time than it is to get the count right.

thresholdfishsick This value is used to distinguish between sick and healthy fish. As sick fish tend to move much slower than healthy ones, especially when compared over several frames, this threshold was chosen to be as low as 0.25. As described in Chapter 4.2.1.3, this value would definitely have to be changed (or re-conceptualized entirely) depending on the examined species.

thresholddeadfishframestoconsider Due to the fact that dead fish hardly move, in general it should be easy to detect them reliably over a long period of time. This threshold is used to only consider a fish as dead once it has been detected for x frames in a row. This not only saves computational power, it also ensures that fish don't mistakenly get detected as dead whenever they are changing direction or hovering for a bit.

thresholddeadfishminmovement When doing the calculations described in Chapter 4.2.1.3, a value needed to be established to describe the maximum length a dead fish could possibly move due to the natural current present in the tank. After some testing, this value was set to 0.035, which is about 1/30 of the width and height of the captured image. While the distance an average fish can swim in one frame was limited to 0.05, this threshold does not concern one singular frame but the entire duration of recording. It is important to note, though, that this value will have to be changed depending on how far the camera is away from the tank and whether or not the image is focused on the tank and nothing but the tank. Since the values are normalized, having the tank's surrounding be part of the video footage would alter those values quite significantly.

thresholdfishmaxfoundlikely Rather than saving the calculated count value for each frame, one overall value for each video was computed using the following function:

```
1    def max_fish_found(CONFIG, fishSumMaxTotal,
         detectedFishAmountForEachFrame, frameCounter):
2        thresholdFishMaxFoundLikely =
             float(CONFIG['Thresholds']['thresholdFishMaxFoundLikely'])
3        finalMaxFound = 0
4
5        for amount in range(fishSumMaxTotal, 0, -1):
6            maxCount = detectedFishAmountForEachFrame.count(amount)
7            maxRatio = maxCount / frameCounter
8            if maxRatio >= thresholdFishMaxFoundLikely:
9                finalMaxFound = amount
```

```
10                print("Amount of fish probably " + str(finalMaxFound) +
                      ", with a ratio of " + str(round(maxRatio, 4)))
11                break
12            else:
13                print("Discarding " + str(amount) + " max fishes with a
                      ratio of " + str(round(maxRatio, 4)))
14
15        return finalMaxFound
```

Code 3: Function to calculate the number of fish detected in a given video.

The amount of fish detected in each frame is saved in a list which is passed to this function. Because there will always be false positives in the form of reflections of the fish on the side of the tank, other objects in the tank or reflections from outside light-sources on the image, simply taking the highest amount of fish detected simultaneously did not seem to be exact, which is why this threshold was introduced. Logic dictates that false positives will only happen every so often, while real fish should be detected most of the time. This means that if the highest value only occurs very rarely, it probably contains all fish as well as at least one false positive. By going through the list of detected fish from top to bottom and discarding values with a low occurence (in this case, lower than this threshold), false positives can be discarded relatively reliably.
This has two major implications. Firstly, the shorter the underlying video is, the less precise the result will be. Secondly, changing this threshold will hugely affect the resulting count variable, which is why it is crucial to test this parameter. The more exact the algorithm is, the less important this threshold becomes.

maximumfishdistance This variable helps making the tracking of fish more accurate. When matching the detected fish of two consecutive frames, the distance between each fish is calculated and compared to this value. As a fish can only travel a certain distance over one frame, distances that are larger than this threshold can be ignored, as they must belong to a different fish. Just like some of the other thresholds, this value was estimated and tested after observing several different videos and fish.

4.3.3 Object detection algorithm

The underlying object detection algorithm was a version of Faster R-CNN. It was chosen due to the fact that it manages to detect in real-time, which might be relevant in later stages of the projects, when data from thousands of tanks of fish start pouring in. Another reason was that it appears to perform much better on small objects than SSD. While the yielded results using Faster R-CNN were excellent from the start, an argument can be made for re-doing the experiment using algorithms such as SSD or R-FCN. Due to the time constraints as well as the immediate success of Faster R-CNN a different object detection algorithm has not been

used during this experiment, however, implementations of SSD [6] as well as R-FCN [41] for TensorFlow can be found on GitHub, meaning that testing them in the same environment is feasable and might turn out to improve the results even further.

5 Results & Discussion

This chapter concerns itself with the results of the experiment described in Chapter 4. Whilst simply presenting some of the results, other results deserve a closer look and are therefore being discussed further.
The collected data (fish count, amount of sick and dead fish and fish velocity of each video) was saved in an InfluxDB database instance and visualized using Grafana, as Grafana has an interface for InfluxDB and can therefore easily connect to it. In all three scenarios, all 319 videos were analyzed.

Scenario 1 For the first iteration of the inference graph, 490 images from four different angles (front, left, right, top) were taken and labelled.

Figure 8: Count of fish using the first iteration of the inference graph.

When taking a closer look at Figure 8, several things become apparent:
First of all, with an average count rate of 8.62, the algorithm is quite a bit off from the actual 11 fish in the tank. However, with only 5 counts of 12 it seems to encounter a very small amount of false positives.
Secondly, the longer the measurements go on, the further they deviate from the desired value. This can easily be explained by the deterioration of water quality over time, as the fish tank was directly connected to water containing algae and zooplankton.
Finally, after examining some of the videos with a very low count score, another issue was highlighted: clustering. In certain situations, the entire swarm of bluegill would cluster, thereby making it very hard for the algorithm to detect fish that were obscured by others. This, in combination with the decline of water quality, was decisive for many of the amiss measurements.

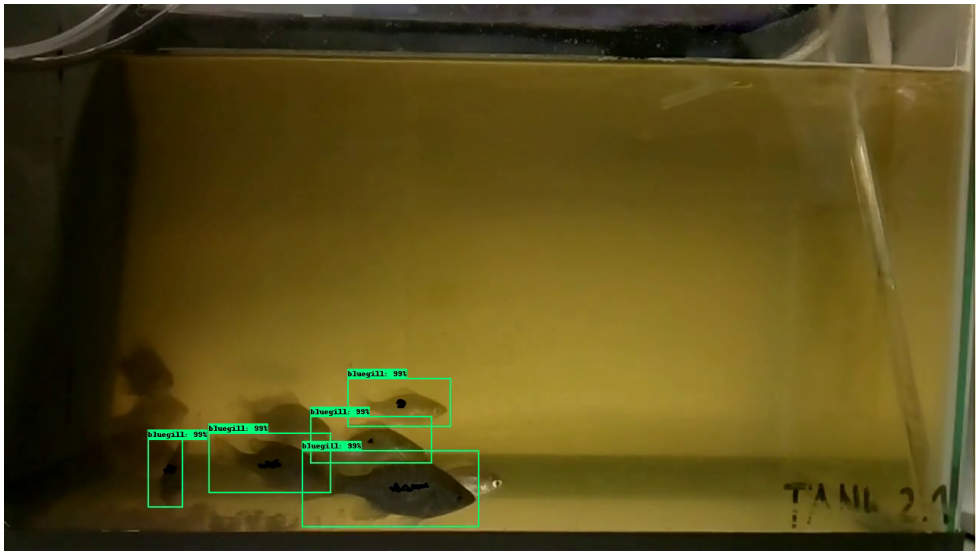

Figure 9: Fish clustering in one corner of the tank in poor water quality.

Scenario 2 Using the same inference graph but making a slight adjustment to `thresholdfishmaxfoundlikely` (described in section 4.3.2), thereby requiring count values to be measured only 2% of the time rather than 5% of the time, the graph changed slightly:

Figure 10: Count of fish using the first iteration of the inference graph and a reduced threshold of accepted count values.

When comparing Figure 8 and Figure 10, the following deductions can be made:
The later values are still significantly lower than the earlier ones, further supporting the assumption that the reason behind the decline lies not with thresholds or the algorithm itself but the poor water quality at that point in time.
With 11 counts of 12 and 3 counts of 13, quite a bit more false positives have now been accepted. However, the amount of times 11 fish were correctly identified increased from x to y.
At the same time, many of the measurements with comparatively low values have increased, thereby pushing the average to 9.1. This means that by accepting more false positives, a better

estimate of the correct value can be achieved.
Lastly and most importantly, as depicted in Table 1, the amount of correct measurements almost doubled from 33 to 61 when adapting the threshold. Additionally, the entire graph shifted towards the correct value. While the three most common results in scenario 1 where a count of 8, 9 and 10, scenario 2 most commonly resulted in a count of 9, 10 or 11.

```
> select count(Count) from Fish where TankId = '6' and Count=11
name: Fish
time count
---- -----
0    33
> select count(Count) from Fish where TankId = '8' and Count=11
name: Fish
time count
---- -----
0    61
```

Figure 11: Database output of correct fish count for scenario 1 and scenario 2.

Scenario 3 As the change of the `thresholdfishmaxfoundlikely` threshold seemed to have resulted in more accurate measurements overall, another iteration of the inference graph was due. Rather than adding more images to the pool of learning data, after extensively examining all current images, a decision was made to remove all images made from the top of the tank, as the videos were always taken from the front, thereby depicting the fish very differently. Thus, the pool of images was reduced from 490 to 375, and the neural network was taught what fish looked like once again. Consequently, all 319 videos were re-evaluated using the newly created inference graph with the `thresholdfishmaxfoundlikely` still set to 0.02.

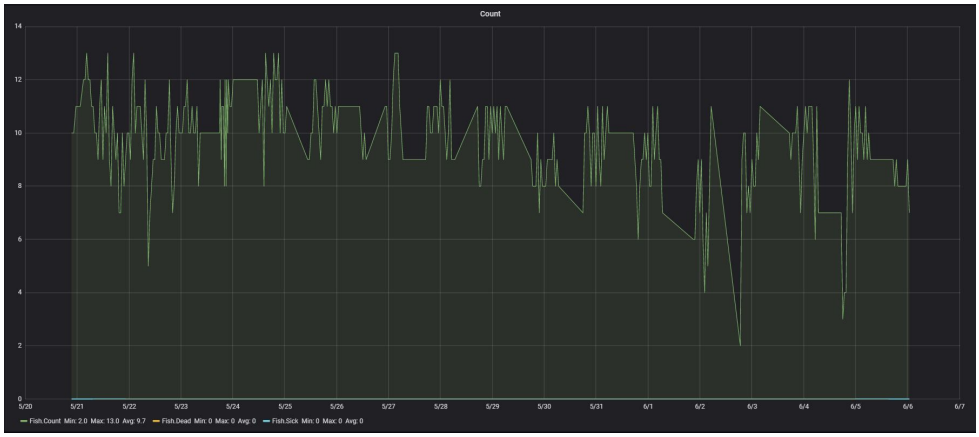

Figure 12: Count of fish using the second iteration of the inference graph and a reduced threshold of accepted count values.

As displayed in Figure 12, the average rose again, this time to 9.7. Most importantly, a strong decrease in of low measurements could be detected, as shown in Table 1. While the results

were still centered around a result of 10 rather than 11, the number of times 11 fish were correctly identified increased again. At the same time, however, the number of false positives rose significantly. As Figure 13 clearly shows, the reason for this was, that the algorithm now had a much easier time detecting reflections of fish as well. One way to circumvent this issue would therefore be to minimize the reflection on the image even further by optimizing the angle of the camera. Unfortunately, there was no apparent way of getting rid of the reflection part of the tank without losing a part of the tank completely. Changing the angle of the camera would only result in a different part of the tank being reflected, and moving it in closer would cut part of the tank, thereby introducing the possibility of not having all fish on the video.

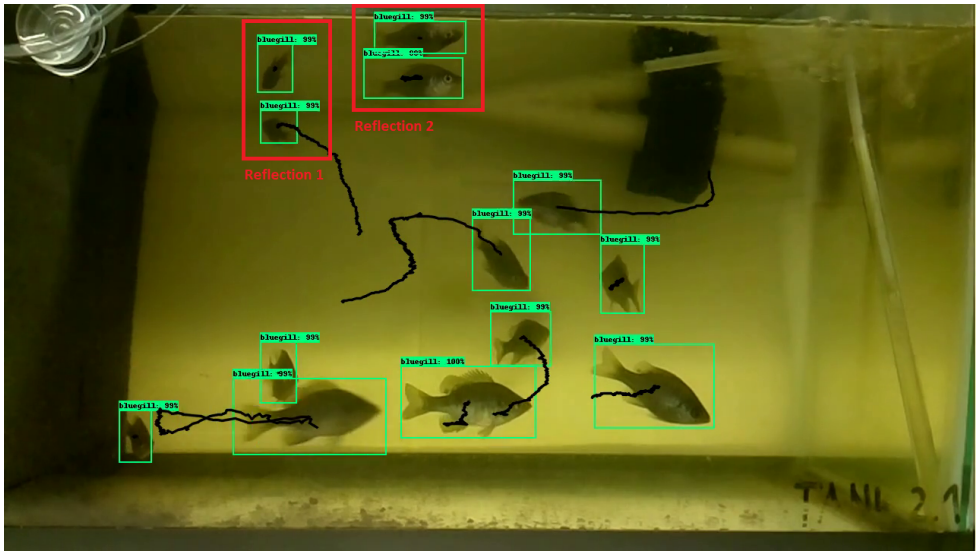

Figure 13: The two fish at the very top of the tank were counted twice due to the reflection, thereby pushing the count to a total of 13.

Table 1 gives an overview of all measurements of each scenario. While the most commonly counted value was 10 for each scenario, both the average as well as the amount of times 11 was correctly identified as the number of fish in the tank increased with each scenario. As going through new iterations of inference graph and re-evaluating all videos cost resources in the shape of time and money, it was not practical to add any additional iterations at this point. However, further improvement of the results could definitely be achieved by constantly tweaking thresholds, inference graph and video properties (angle, water quality, etc.).

$$\text{time per step} * \text{necessary steps} = \text{time needed for inference graph}$$

$$0.1 \text{ seconds} * 60,000 = 100 \text{ minutes}$$

$$\text{time per video} * \text{amount of videos} = \text{time needed for video evaluation}$$

$$1 \text{ minute} * 319 = 319 \text{ minutes}$$

$$\text{sum of time} * \text{price for AWS instance} = \text{cost of full iteration}$$

$$419 \text{ minutes} * 5.1 \text{ cent/minute} = 21.37 \$$$

Figure 14: Time and money needed for one new iteration of inference graph and a re-evaluation of all videos.

	Tank 6 (Scenario 1)		Tank 8 (Scenario 2)		Tank 9 (Scenario 3)	
	Amount	%	Amount	%	Amount	%
0	0	0.00	0	0.00	0	0.00
1	0	0.00	0	0.00	0	0.00
2	3	0.92	2	0.62	1	0.31
3	1	0.31	2	0.62	1	0.31
4	5	1.54	3	0.92	3	0.92
5	11	3.38	3	0.92	2	0.62
6	18	5.54	18	5.54	5	1.54
7	36	11.08	27	8.31	16	4.92
8	65	20.00	51	15.69	37	11.38
9	66	20.31	57	17.54	65	20.00
10	82	25.23	81	24.92	80	24.62
11	**33**	**10.15**	**61**	**18.77**	**70**	**21.54**
12	5	1.54	11	3.38	30	9.23
13	0	0.00	3	0.92	9	2.77
14	0	0.00	0	0.00	0	0.00

Red = Top 3 results. **Bold** = Correct fish count. Underlined = Most common result.

Table 1: Amount of count and respective ratio of all three different scenarios.

Sick and dead fish As previously stated, the algorithm for differentiating between healthy, sick and dead fish was tested on zebrafish rather than bluegill. The reason was that sick zebrafish drastically alter their movement, while bluegill do not. Additionally, a sick zebrafish was present, which meant that proper testing could occur.
Another reason not to use bluegill in this case was that they are static most of the time, which made it hard to tell a dead fish from a healthy one by simply looking at the travelled distance within a certain amount of frames. Since the video length was limited to 30 seconds, and bluegill often don't move for minutes, it seemed impractical to test the algorithm on them.
One possible approach would be to limit possible dead fish detections to the upper and lower boundaries of the tank, as dead fish will always float either on the bottom or the top, depending on the amount of time they have been dead already. Additionally, the orientation of each fish could be taken into account, as the bodies of dead fish tend to take weird shapes and curvatures. This would still not get rid of all the false positives of healthy fish which just happened to be at that section of the tank and not move or which were turning direction at that time, which is why for the purposes of this thesis, sick and dead fish detection was only tested on zebrafish.
The following images show how several zebrafish were successfully detected as healthy, sick and dead:

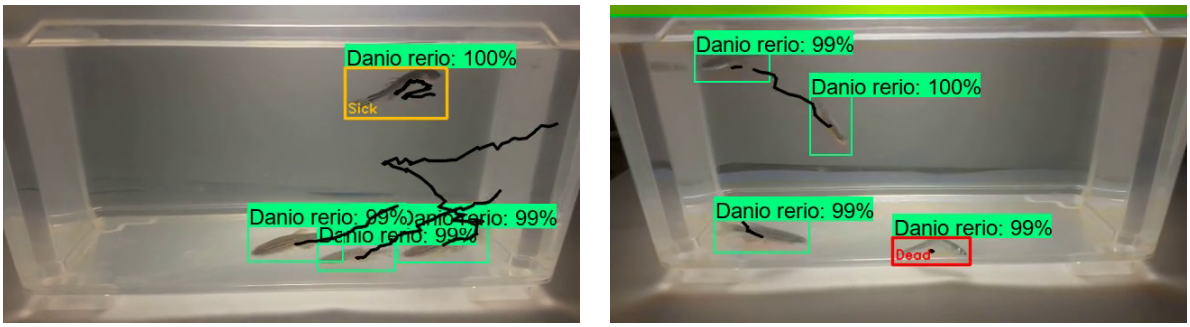

Figure 15: Sick and dead fish detection of zebrafish.

Velocity Figure 16 shows a graph of the average fish velocity as well as its standard deviation for each video in scenario 3.

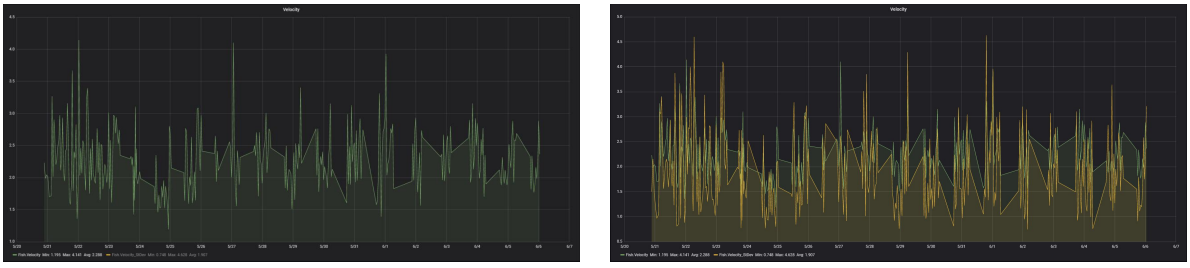

Figure 16: Average velocity and velocity plus standard deviation.

When only considering the average velocity of all fish, very little outliers can be found. In fact, only three values can be found which get close to being double the average velocity - on May

22nd, May 27th and June 1st. As these spikes are so far out of the norm, the underlying videos were looked at closely. Upon inspection the reason for the spike in velocity quickly became apparent - all of the videos were taken while the fish were being fed.
This helped prove two points. Firstly, even when swimming this quickly, tracking of fish worked just as well. There had been some fear of the fish being displayed blurry when moving fast, but that turned out not to be the case. Additionally, this proved the theory of increased movement speed when hungry fish were being fed. It therefore stands to reason that a correlation between hunger and movement speed during feeding can be made, thus helping to find optimal feeding cycles.
Taking the standard deviation into account, more deductions can be made. A lot more spikes can be found in this graph, because of the bluegill's natural behavior. They are stationary most of the time, with the occasional sudden movement. However, when grouped together closely, often times one of the fish will chase away another to make more room for itself. This means that, while most of the fish are hardly moving, two of them are swimming very quickly, thereby creating spikes in the standard deviation graph.
Interestingly, when only looking at the exact data points of the feeding videos, a very low standard deviation is expected as all fish should be relatively equally hungry. A spike in both graphs would therefore signal that at least one of the fish hardly moved during feeding, indicating health issues for said fish. Such a scenario can be seen on June 1st, for example. This could be a first step towards solving the sick fish detection for bluegill.

6 Conclusion

While the detection and tracking of fish using object detection and TensorFlow can definitely be achieved, there are a lot of things to be considered and many possible variables to be optimized. Making sure of having decent environmental conditions (lighting, water quality, camera setup, etc.) is just as important as choosing the right training data for the algorithm and tweaking thresholds until they fit individual needs. Most importantly, when it comes to investigating health, various behavioral patterns of each species of fish need to be considered and the algorithm might have to be adapted accordingly.

6.1 Future prospects

Counting, tracking and calculating the velocity of fish has been achieved, however, there is still a lot of potential in terms of health and size classification. Some ideas have already been proposed in this paper, such like the dual camera setup [5] and the correlation of velocity and standard deviation during feeding.
Since this problem originated from Blue Planet Ecosystem's CLPPR idea, more issues will arise once the scaling up of the prototype starts and fish will have to be tracked within a 40-foot shipping container. As it seems impractical to cover such a tank with myriads of cameras in order to get video footage of the entire fish population, methods need to be developed to work around that.
When using mathematical models and statistics to estimate the fish population judging by only a small snippet of water, the count might even turn out relatively precisely, however, identifying sick or dead fish might be difficult when those fish are not moving by the camera. Therefore, it might be necessary to develop a method of making all fish swim by a certain camera-surveyed spot. This would also make classification of size much easier. However, it is more likely to find a solution to this using engineering rather than computer science.

Bibliography

[1] *Blue Planet Ecosystems*. https://www.blueplanetecosystems.com/. (Accessed on 08/14/2019).

[2] *How IoT's are Changing the Fundamentals of "Retailing"*. https://trak.in/tags/business/2016/08/30/internet-of-things-iot-changing-fundamentals-of-retailing/, May 2017. (Accessed on 08/23/2019).

[3] AHMED REDA AMIN EL-BARKOUKY: *Mathematical modeling for partial object detection.*. https://pdfs.semanticscholar.org/efec/a7e76f4a819c0c91e47945e04989b30860e4.pdf?_ga=2.18852217.1960961484.1567506972-1401901494.1567164359, December 2014. (Accessed on 09/03/2019).

[4] ANNA KOBUSINSKA, CARSON LEUNG, CHING-HSIEN HSU, RAGHAVENDRA S., VICTOR CHANG: *Emerging trends, issues and challenges in Internet of Things, Big Data and cloud computing*. https://www.sciencedirect.com/science/article/pii/S0167739X18311270, June 2018. (Accessed on 08/26/2019).

[5] B. ZION, A. SHKLYAR, I. KARPLUS: *In-vivo fish sorting by computer vision*. https://www.sciencedirect.com/science/article/pii/S0144860999000370, June 1999. (Accessed on 08/14/2019).

[6] BALANCAP: *GitHub - balancap/SSD-Tensorflow: Single Shot MultiBox Detector in TensorFlow*. https://github.com/balancap/SSD-Tensorflow, April 2017. (Accessed on 08/30/2019).

[7] CHOICE, A.: *Crontab – Quick Reference*. https://www.adminschoice.com/crontab-quick-reference. (Accessed on 09/05/2019).

[8] CHRIS SNIJDERS, UWE MATZAT, ULF-DIETRICH REIPS: *"Big Data": Big Gaps of Knowledge in the field of Internet science*. https://www.ijis.net/ijis7_1/ijis7_1_editorial.pdf, 2012. (Accessed on 08/26/2019).

[9] CRISTIANO ANDRÉ DA COSTA, CRISTIAN F. PASLUOSTA, BJÖRN ESKOFIER, DENISE BANDEIRA DA SILVA, RODRIGO DA ROSA RIGHI: *Internet of Health Things: Toward intelligent vital signs monitoring in hospital wards*. https://www.sciencedirect.com/science/article/pii/S0933365717301367, June 2018. (Accessed on 08/23/2019).

[10] DIELEMAN, S.: *Classifying plankton with deep neural networks*. http://benanne.github.io/2015/03/17/plankton.html, March 2015. (Accessed on 08/28/2019).

[11] DIETMAR SCHABUS: *Machine Learning 2*, September 2018. (Accessed on 08/23/2019).

[12] DIVYANSH JHA, ROHIT SINGH: *Swimming pool detection and classification using deep learning*. https://medium.com/geoai/swimming-pool-detection-and-classification-using-deep-learning-aaf4a3a5e652, July 2018. (Accessed on 09/03/2019).

[13] EDJEELECTRONICS: *EdjeElectronics: How to train a TensorFlow Object Detection Classifier for multiple object detection on Windows*. https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10, April 2019. (Accessed on 06/02/2019).

[14] FADI AL-TURJMAN, ARMAN MALEKLOO: *Smart parking in IoT-enabled cities: A survey*. https://www.sciencedirect.com/science/article/pii/S2210670718327173, May 2019. (Accessed on 08/23/2019).

[15] FRANCESCO SOTTILE, STEFANO SEVERI, FRIEDBERT BERENS, AND MAURIZIO SPIRITO: *M2M Technologies: Enablers for a Pervasive Internet of Things*. https://www.academia.edu/6866526/M2M_Technologies_Enablers_for_a_Pervasive_Internet_of_Things, 2014. (Accessed on 08/23/2019).

[16] GIRSHICK, R.: *Fast R-CNN - IEEE Conference Publication*. https://ieeexplore.ieee.org/document/7410526, February 2016. (Accessed on 08/30/2019).

[17] GOOGLE: *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. http://download.tensorflow.org/paper/whitepaper2015.pdf, November 2015. (Accessed on 08/27/2019).

[18] HUI, J.: *SSD object detection: Single Shot MultiBox Detector for real-time processing*. https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06, March 2014. (Accessed on 08/30/2019).

[19] IMAGENET: *Large Scale Visual Recognition Challenge 2016 (ILSVRC2016)*. http://image-net.org/challenges/LSVRC/2016/index#comp, 2016. (Accessed on 09/03/2019).

[20] JAY LEE, BEHRAD BAGHERI, HUNG-AN KAO: *A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems*. https://www.sciencedirect.com/science/article/pii/S221384631400025X, December 20114. (Accessed on 08/26/2019).

[21] JERETTA HORN NORD, ALEX KOOHANG, J. P.: *The Internet of Things: Review and theoretical framework*. https://www.sciencedirect.com/science/article/pii/S0957417419303331, May 2019. (Accessed on 08/23/2019).

[22] JIFENG DAI, YI CI LI, KAIMING HE, JIAN SUN: *R-FCN: Object Detection via Region-based Fully Convolutional Networks*, 2016.

[23] JOSEPH REDMON, SANTOSH DIVVALA, ROSS GIRSHICK, ALI FARHADI: *You Only Look Once: Unified, Real-Time Object Detection*. https://ieeexplore.ieee.org/document/7780460, December 2016. (Accessed on 08/30/2019).

[24] J.R.R. UIJLINGS, K.E.A. VAN DE SANDE, T. GEVERS, A.W.M. SMEULDERS: *Selective Search for Object Recognition*. https://link.springer.com/article/10.1007/s11263-013-0620-5, April 2013. (Accessed on 08/30/2019).

[25] LAZORENKO, A.: *TensorFlow performance test: CPU VS GPU*. https://medium.com/@andriylazorenko/tensorflow-performance-test-cpu-vs-gpu-79fcd39170c, December 2017. (Accessed on 08/27/2019).

[26] MIGUEL A. ZAMORA-IZQUIERDO, JOSÉ SANTA, JUAN A. MARTÍNEZ, ANTONIO F. SKARMETA: *Smart farming IoT platform based on edge and cloud computing*. https://www.sciencedirect.com/science/article/pii/S1537511018301211, November 2018. (Accessed on 08/23/2019).

[27] MISHRA, M.: *Convolutional Neural Networks, Explained*. https://www.datascience.com/blog/convolutional-neural-network, March 2019. (Accessed on 08/29/2019).

[28] NVIDIA: *CUDA GPUs*. https://developer.nvidia.com/cuda-gpus. (Accessed on 08/28/2019).

[29] NVIDIA: *CUDA Toolkit*. https://developer.nvidia.com/cuda-toolkit. (Accessed on 08/28/2019).

[30] PARVANEH ASGHARI, AMIR MASOUD RAHMANI, H. H. S. J.: *Internet of Things applications: A systematic review*. https://www.sciencedirect.com/science/article/pii/S1389128618305127, December 2018. (Accessed on 08/23/2019).

[31] POOYA TABESH, ELHAM MOUSAVIDIN, SONA HASANI: *Implementing big data strategies: A managerial perspective*. https://www.sciencedirect.com/science/article/pii/S000768131930028X, March 2019. (Accessed on 08/26/2019).

[32] REMANAN, S.: *Beginner's Guide to Object Detection Algorithms*. https://towardsdatascience.com/beginners-guide-to-object-detection-algorithms-6620fb31c375, April 2019. (Accessed on 08/30/2019).

[33] ROHITH GANDHI: *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms*. https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e, July 2018. (Accessed on 08/30/2019).

[34] ROSS GIRSHICK, JEFF DONAHUE, TREVOR DARRELL, JITENDRA MALIK: *Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation*. https://ieeexplore.ieee.org/document/6909475, September 2014. (Accessed on 08/30/2019).

[35] SERVICES, A. W.: *Amazon EC2 P3 – Ideal for Machine Learning and HPC - AWS*. https://aws.amazon.com/ec2/instance-types/p3/. (Accessed on 09/05/2019).

[36] SHAOQING REN, KAIMING HE, ROSS GIRSHICK, JIAN SUN: *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. https://ieeexplore.ieee.org/document/7485869, June 2016. (Accessed on 08/30/2019).

[37] SHIXING LI, HONG WANG, TAO XU, GUIPING ZHOU: *Application Study on Internet of Things in Environment Protection Field*. https://link.springer.com/chapter/10.1007%2F978-3-642-25992-0_13, 2011. (Accessed on 08/23/2019).

[38] TENSORFLOW: *Build from source for the Raspberry Pi*. https://www.tensorflow.org/install/source_rpi. (Accessed on 08/28/2019).

[39] VARSHNEY, K.: *Object Detection On Aerial Imagery Using RetinaNet*. https://towardsdatascience.com/object-detection-on-aerial-imagery-using-retinanet-626130ba2203, March 2019. (Accessed on 09/03/2019).

[40] WEI LIU, DRAGOMIR ANGUELOV, DUMITRU ERHAN, CHRISTIAN SZEGEDY, SCOTT REED, CHENG-YANG FU, ALEXANDER C. BERG: *SSD: Single Shot MultiBox Detector*. https://link.springer.com/chapter/10.1007%2F978-3-319-46448-0_2, September 2016. (Accessed on 08/30/2019).

[41] XDEVER: *GitHub - xdever/RFCN-tensorflow: RFCN implementation in TensorFlow*. https://github.com/xdever/RFCN-tensorflow, February 2017. (Accessed on 08/30/2019).

List of Figures

List of Tables

List of Code

List of Abbreviations

CLPPR Closed Loop Photo Protein Reactor

IoT Internet of Things

AWS Amazon Web Services

API application programming interface

CPU central processing unit

GPU graphics processing unit

CNN convolutional neural network

YOLO You Only Look Once

SSD Single Shot Multibox Detector

R-FCN Region-based Fully Convolutional Network

R-CNN Region-CNN

XML eXtensible Markup Language