Fish:rec – An AI-based fish recognizer tool

Fish:rec – Mesterséges Intelligencia alapú halfelismerő eljárás

Scientific Students' Associations Conference 2023

(Tudományos Diákköri Konferencia 2023)

By:

Reich Márton

Supervisors:

Dr. Sándor Baranya

Associate professor, Department of Hydraulic and Water Resources Engineering

Tikász Gergely

Assistant research fellow, Department of Hydraulic and Water Resources Engineering

Budapest, 2023

CONTENTS

Abstract

Monitoring fish population is essential, as fishes are key components of the aquatic ecosystem and changes in their population size and mix are important ecological indicators. Nevertheless, it is a labour intensive and time-consuming job to monitor fish populations, as the methods used today need scientists experienced at recognizing fish species. On top of that, these manual methods currently in use must be repeated frequently to get tangible outcomes, for a given period, at a given location.

This research proposes an Artificial Intelligence (AI) based solution to automatize fish population monitoring using machine learning and computer vision. For this, I deploy a deep learning based method which uses convolutional neural networks (CNN) for image recognition, and object detection. The fish monitoring tool is developed, trained and tested using a high number of annotated and augmented images from video footages captured at a fish pass. The test site is located at the River Tisza in Hungary at Kisköre, where different fishes, representative for the Hungarian rivers, are migrating. The paper introduces the AI model selection, the training and testing phases of the tool development, and provides a method for quantifying the fish migration.

1. Introduction

1.1. Conventional methods of fish monitoring

Fish monitoring is a favoured way to detect changes in the environment and determine water quality in numerous aquatic habitats. The usage of fishes as biological indicators has been cited in multiple studies and legislation related to aquatic ecosystems and water resources (Fausch et al., 1990; Water Framework Directive, 2000/60/EC; Kurtz et al., 2001). Numerous investigations concentrate on enhancing and safeguarding environmental conditions, which are important for aquaculture and nature conservation.

Biological indicators are preferred over chemical indicators in the conservation of aquatic environments because the chemical indicators often overlook environmental conditions and changes caused by anthropogenic impacts. While chemical and physical properties of water are excellent at detecting certain aspects of the water body, they cannot reveal changes in biotic integrity and the effects of human pollution. This view is reinforced by Oberdorff and Hughes (1922), who conducted a comparison between the Water Quality Index (WQI) and a modified version of the Index of Biotic Integrity (IBI) using chemical and biological monitoring techniques in the Seine River. The study found that IBI is a more sensitive and resilient measure for evaluating water quality. It should be noted, however, that this does not imply that biological monitoring methods are inherently preferable to chemical ones. The optimal application of both these methods is site-specific. However, biological monitoring is generally favoured over chemical monitoring.

Fish play a significant part in biological monitoring due to their importance as a pivotal species in aquatic environments. Various methods exist for assessing fish, including direct capture and examination practices, as well as non-invasive techniques.

Electric fishing, netting, and using the catch of many fishermen are the most commonly utilised techniques for evaluating fish populations and their health. (fao.org). Electrofishing employs a direct current between a submerged cathode and anode, taking advantage of the fact that fish swim towards the anode and can be stunned or captured. When carried out correctly, the fish suffers no lasting harm and can resume its normal mobility after two minutes of being caught. Seine, trawl and gill nets are the primary netting methods employed. Though they are commonly used and quite effective, each of them has a drawback. Seine nets are more effective in deep waters than in rivers and freshwater areas, where they may encounter difficulties. Trawl nets can be expensive, and gill nets are rather difficult to use due to their specific shape and size. While making use of the information coming from catches of fishers sounds like an efficient way to assess fish populations, since the professionals don't have to go through the process of catching the fishes, the efficiency depends on the ability and willingness of the third party to help.

The implementation of fish counters and observation techniques are stress-free methods that do not require capturing any fish. Two types of fish counters are primarily used: the resistive counter, which takes advantage of varying fish resistance to electric current. The hydroacoustic counter is a second type of fish survey method which utilises sound waves sent through the water and detects them with microphones. However, these methods can only be used for a limited number of fish species at a time, which is their main drawback. Alternative, observational methods can be employed, such as cameras or divers who quantify fish based on footage or visual observation.

Most of the methods require human expertise and are both time consuming and labour intensive. In addition, fork length (the length of the fish), and at times weight measurements, are required post fish capture to undertake a tangible survey.

1.2. AI based image analysis

The concept of image recognition emerged during the 1950s and gained momentum in the 1960s. The invention of the first digital camera marked one of the starting points of computer vision. Russel Kirsch and a group of researchers created a method to convert images into binary numbers that computers could interpret. Among the first digital images produced was a picture of Kirsch's son. Despite its small size, the image has now become iconic.

Prior to the emergence of computer technology, the analysis of images or video material was performed manually by an individual. However, researchers aimed to automate these tasks using computers. With the goal of imitating human vision and recognition, image recognition was developed through The Summer Vision Project (Seymour Papert, 1966) in 1966. The study aimed to enable the computer to distinguish between the background, possible objects, and chaos. Even though it was unsuccessful, this project is regarded as the origin of AI-based computer vision.

Understanding image structures by extracting 3D objects made image recognition a cutting-edge technology. The 1970s witnessed a surge in studies and innovative ideas based on computer vision. This period saw the development of the basics of computer vision programs that are still in use today, such as edge extraction and line labelling. Subsequent studies resulted in additional innovations such as contour models and shape recognition based on shading. Further development was carried out by a Japanese scientist called Kunihiko Fukushima to create a self-organising artificial neural network that was unaffected by changes in position. This invention, called Neocognitron, is recognised as the first deep neural network and the predecessor of today's convolutional neural networks (CNNs). This led to further research into feature-based recognition technologies. Feature-based recognition works by determining the object from features immutable from rotation or location.

In 2006, Fei-Fei Li founded Imagenet, a project to help scientists build difficult neural networks and overcome overfitting and underfitting. In 2010, Imagenet consisted of more than 3 million carefully labelled images. Imagenet has been a huge boost to the modern world of image recognition. The Imagenet Large Scale Visual Recognition Challenge (ILSVRC), a computer vision competition held in 2010, motivated researchers to think of innovative ways to solve image recognition problems and reduce error rates. This competition prompted a team from the University of Toronto to develop Alexnet. Alexnet used a convolutional neural network architecture and managed to reduce the error rate by 10% compared to previous models, which had an error rate of 15.3%. By 2017, the error rate dropped to 5% only.

Fast forward to today, models based on these algorithms are widely implemented and have gained popularity even among non-professionals. Examples of the many implementations of image recognition today are listed below:

> *Automated driving* – Also known as self-driving, automated driving is a dominant research topic in the field of computer vision. Some car manufacturers, including Tesla and Waymo, have already integrated this technology into their vehicles. Despite many worries about the dangers, self-driving is a future of vehicles.

> *Facial recognition in security systems* – Facial recognition is commonly associated with security, particularly in relation to mobile phones. Modern smartphones incorporate facial and biometric recognition algorithms to enhance security. In addition, facial recognition systems are employed in criminal identification, surveillance, and airport security.

Healthcare systems and medical diagnostics – AI has diverse and important applications in medical fields. Image recognition can help detect diseases and identify cancer cells. Many cancer treatments employing AI to identify cancer cells, then applying laser treatment with precise machines.

These are just a few examples. Image recognition is frequently utilized in society, often without individuals being aware of it. Despite the popularity of computer vision, there are several limitations. The primary issue is the insufficient amount of adequately sized datasets. In order to achieve a model that performs well enough to be utilized in real-world situations, tens of thousands or even hundreds of thousands of carefully labelled images are necessary. Acquiring these images can be problematic in certain sectors. Labelling is also a strenuous exercise, as it is both time-consuming and labour-intensive. In addition, image recognition can have difficulties in areas where image quality is poor, with the result that the algorithm malfunctions. Nevertheless, the advantages of artificial intelligence outweigh the disadvantages.

1.3. Image recognition in fish monitoring

Various studies applied image recognition on underwater footage in the past years (a little over 100 in the last 3 years according to Barbedo, 2022. In the words of Saleh et al. (2022), underwater environments raise 5 problems:

- lighting varies frequently, objects a little farther are much less bright,
- scenes are highly dynamic, scenes change quickly,
- depth and distance perception can be incorrect due to refraction,
- images are affected by water turbidity and light scattering,
- image data are under-sampled due to low resolution.

Due to these issues, a few research have gone into trying to enhance the images, extract shapes and make images more suitable for further analysis (Zhou et al., 2017; Islam et al., 2020). Even though these techniques can emphasize features that may be obstructed, but they cannot recover information that has been lost due to poor conditions. One approach to address this issue is to use near infrared cameras and combine the information with optical images (Zhou et al., 2018).

Most of the studies applied image recognition in controlled environments, and a few have done in uncontrolled habitats. Among the latter ones a few have done tracking on individual fishes (Abe et al., 2021), and the rest detecting several fish species ranging from 1 to 10 most of the time. Villon et al. (2018) have detected 20 fish species in their studies.

Having recognized the importance of fish monitoring, many companies have decided to automate the process using artificial intelligence.

- **Cermaq:** The project aims to improve fish welfare and product quality with individual based filtering techniques. The process involves fishes in the cage going into a sensor, which determines their health and whether they are fit for consumption or should be kept alive. (cermaq.com)
- **Fishal.ai:** The goal of the 3-phase project is to create a user-friendly environment for fishers to upload images of caught fishes, use these images to create large datasets, and develop an artificial intelligence model to recognize fishes. (fish.ai)

1.4. Goal of this study

This study aims to improve biological monitoring of aquatic environments through the utilization of artificial intelligence. Camera footage of fish passing through the Kisköre fish pass are continually observed by convolutional neural networks. An artificial model using the YOLOv8 architecture has been implemented, with thousands of fish images used for training. The model has been applied to the camera footage. After analysing the footage, the model identifies the majority of fishes and produces statistics related to their population, size, and average swimming direction.

Implementing computer vision in fish monitoring offers numerous benefits. Automation makes the process much more efficient, reducing the need for biologists to evaluate fish populations. A system such as the one outlined in this study has the capacity for monitoring fish while reducing the workload on scientists. Additionally, this system can run continuously, producing ongoing data and statistics on fish. However, image recognition also has disadvantages and limitations. A camera must be set up in a way that the video quality and the visibility of the fishes are qualified for the use of the model. Additionally, addressing the challenge of small datasets is necessary.

2. Methods

2.1. Study site

Tisza is the longest river in Hungary, surpassing Danube by more than 150 km solely within the Hungarian section. It measures 962 km overall, of which 597 km flows through Hungary. Tisza is culturally and geographically important part of Hungary. Along with Danube, it splits Hungary into three parts, while its catchment area covers a significant area of Eastern Hungary (47,000 km^2). Tisza originates in the Eastern Carpathians in Ukraine, near a town called Rahó. Tisza river flows through Romania, Hungary, Slovakia, and Serbia before eventually flowing into the Danube in the middle of Vojvodina, near Titel. The river enters Hungary at the village of Tiszabecs and leaves near Gyálarét. In some sections in the north and south Tisza serves as a border between Ukraine and Hungary, and Serbia and Hungary. The Hungarian section of Tisza can be divided into three branches: Upper Tisza (northern border of Hungary – Tokaj), Middle Tisza (Tokaj – Tiszaug) and Lower Tisza (Tiszaug – southern border).

The Kisköre hydropower plant, located in the middle section of the Tisza in Hungary, is the highest preforming and largest hydropower plant of Hungary, generating 28 MW of electricity. Its construction began in 1967 and lasted until 1974, and it was inaugurated on the 16th of May 1973. The structure was built at the 403rd river mile with the purposes of providing irrigation water, producing electricity, provision of inland navigation and recreational uses. With the hydropower plant being in use, a forebay has been raised and the elevated section formed a man-made lake called Lake Tisza. Harnessing it for power was only additional benefit, it was not the main goal.

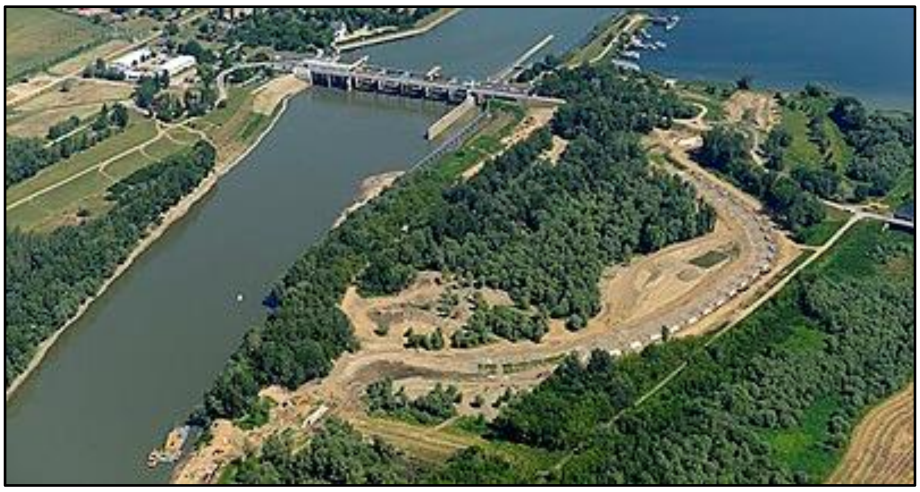


Figure 1: Kisköre hydropower plant and fish pass

Seeing that the flood gates blocked fishes from naturally traversing through the entirety of the Tisza a fish pass was integrated in the ship lock, however it was ineffective. Therefore in 2014 the Kisköre fish ladder was built, with the aim to help the fishes cross the dam. The new fish ladder consists of 37 stairs with a height difference of 20 cm, three small and one large resting lake, natural swimming pools, and a slotted fish passage. More than 40 fish species were documented to swim through the fish ladder regularly. The Kisköre fish ladder is also open for visitors, featuring multiple viewing places and a window to view the fishes passing by (Figure 2). One of the major Hungarian meteorology website has set up a webcam to record continuous footage of the fishes and to broadcast it on the internet (www.idokep.hu). (kotiweb.vizugy.hu)

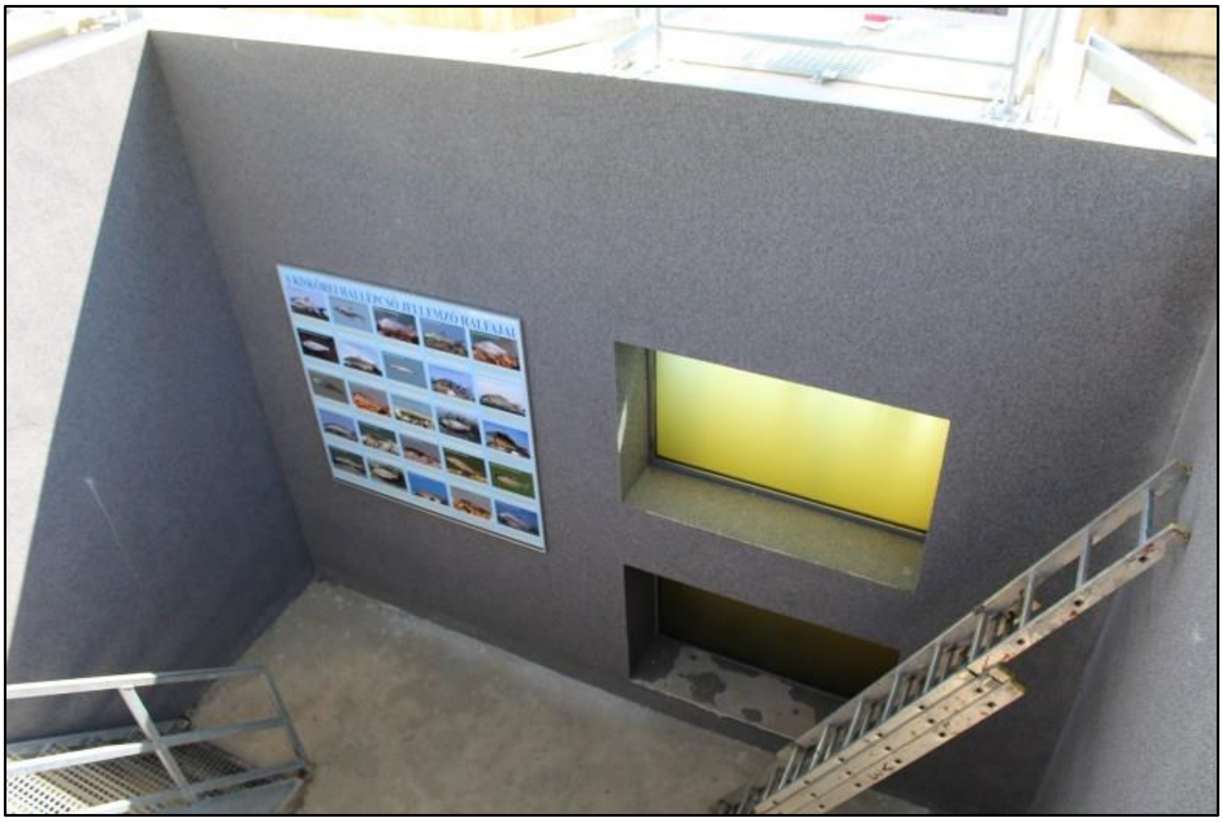


Figure 2: Kisköre fish ladder window

The fishes are sent through a canal, where a wall separates them for a while (Figure 3). On one side of the wall is a window seen in Figure 2. Because this site can be visited, many children place their hands on the window and leave hand marks, making the video feed somewhat blurred. The camera is also setup here and it records the water through the glass pane. Image recognition is aimed to be applied to the footage from the implemented camera. However, problems arise due to the several factors. Water turbidity and varying luminosity reduce the accuracy of the image recognition model. Because of the wall separating the canal, many fishes aren't taken into account, losing information. Further problems include the dirt on the window and significantly lower visibility at night. These problems cannot be solved within the limits of this study, so they must be worked around.

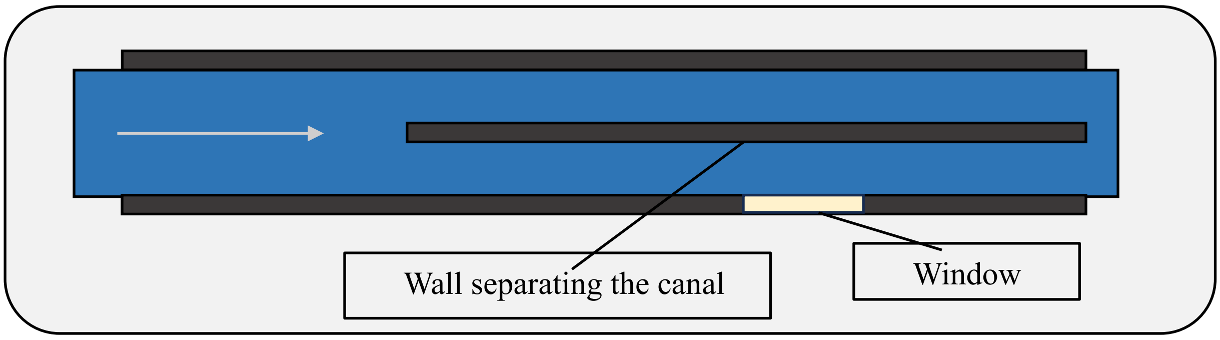


Figure 3: Kisköre fish ladder window structure

2.2. Object detection

2.2.1. Theory behind Convolutional Neural Networks

A popular technique for image recognition is employing convolutional neural network (CNN). As the name implies, CNNs are a distinct type of neural networks (NN). NNs are inspired by the biological nervous system. While their underlying theory is relatively straightforward, the mathematical concept is rather complex.

A NN model comprises an input layer, hidden layers, and an output layer. The input layer receives inputs, the hidden layers perform computations, and the output layer produces the output. Each of these layers contains several neurons that connect with every neuron in the subsequent layer. These neuron connections, known as weights, associate a value with each neuron, and this value is multiplied by the neuron's input. A neuron has a bias that is added to the weight multiplied by the input (Kinsley, 2020).

$$y = x * w + b$$

$$y: output$$

$$x: input$$

$$w: weight$$

$$b: bias$$

This forms the output of the neuron, which becomes the input of the next layer. Prior to sending its output to the next layer, each neuron activates. The activation is usually a function that limits the value of the output and enables the NN to be transformed to resemble any function. Once all the necessary calculations have been completed and the output layer has generated the results, the forward pass is concluded.

If we want to use the model in production only, we only need to go through the forward pass, however, during training the weights and biases, the parameters of the model are changed to make the model better. The training is a process, when the model goes through the forward pass, then based on the results the parameters are changed several times. The number of iterations is called the epoch. The epoch can be a small number, but it usually is a number in the hundreds or thousands, and training can last for weeks. Although we can change the parameters of the model randomly through each epoch, a faster way is to do backpropagation. Backpropagation is when we change the parameters consciously with respect to the inputs. After the forward pass we count loss based on the ground truth, (which we define with image annotation) which is a number that shows how bad was the performance of the model, with a loss function. Then all the parameters are derived with respect to the inputs.

After the derivation, the optimizer changes the parameters based on the derived values it got from the backpropagation multiplied by the learning rate. The learning rate is a hyperparameter (a parameter that the training person gives) that makes the learning process safer by making the learning steps smaller, so the loss doesn't fluctuate or diverge. There exist a variety of optimizers, each with different requirements for hyperparameters. Consequently, the training person must conduct hyperparameter tuning through trial-and-error.

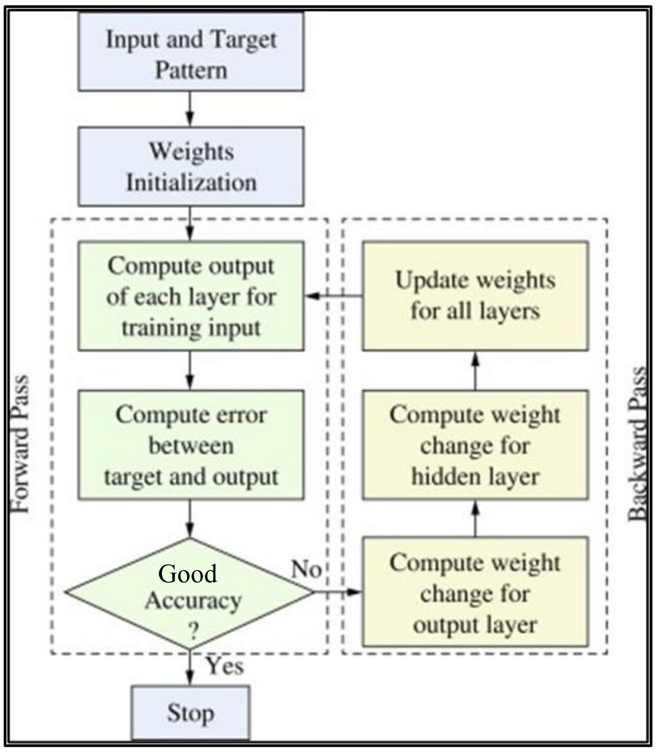

Input and Target Pattern
Weights Initialization
Forward Pass
Compute output of each layer for training input
Compute error between target and output
Good Accuracy ?
No
Yes
Stop
Backward Pass
Update weights for all layers
Compute weight change for hidden layer
Compute weight change for output layer

Figure 4: Epoch flow chart

One epoch consists of a forward pass, backpropagation and the optimizer changing the values. With large models even one epoch can take several minutes, because of the tons of calculations that must be done.

To make training faster and more efficient, the pictures are given to the model in batches. This means that during one epoch not one, but many (usually 16 or 32) images are given to the model.

The variation of neural network, convolutional neural networks fundamentally work the same way, however, there are some calculations before feeding the data into the network. CNNs contain one or more convolutional layers, an activation function, and a pooling layer. Also, CNNs usually take images as matrixes as inputs to preserve information of features that may span over multiple pixels. These matrixes are huge, as even a 640x640 colored image equals to 1,288,800 inputs for the AI. That's why NNs aren't used in image recognition, instead CNN and their feature-based predictions are preferred.

As the image is fed into the model, the convolutional layer amplifies specific features. Different kinds of filter layers extract different kind of features, these results called feature maps. Then the filtered image goes through an activation layer, in order to make the image less noisy and make the features stand out even more. The operational speed and efficiency of the model, even at the cost of information loss, is further elevated by the pooling layer. Thereafter, the processed images are fed to the fully connected neural network for recognition.

2.2.2. Accuracy measurements

Mean average precision (mAP) and intersection over union (IoU) are commonly used metrics to measure the accuracy of a model. mAP50 and mAP50-95 are even more widely used in CNN architectures (deci.ai).

Precision measures the ability of the model to make accurate predictions of the classes, while recall measures the proportion of the actual positive cases that the model correctly identifies (Figure 6). mAP averages the precisions over all the classes. IoU measures the accuracy of the bounding boxes by comparing them to the ground truth. Intersection is the overlap between the ground truth and the prediction, while union is the overall square both boxes can fit in (Figure 5).

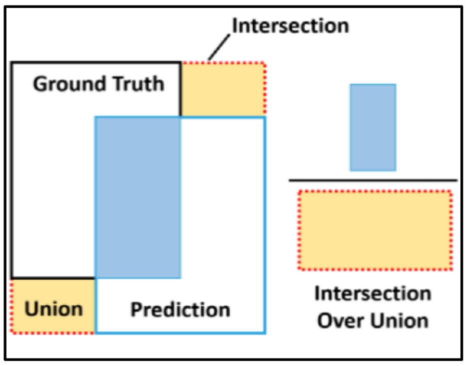


Figure 5: Intersection over Union

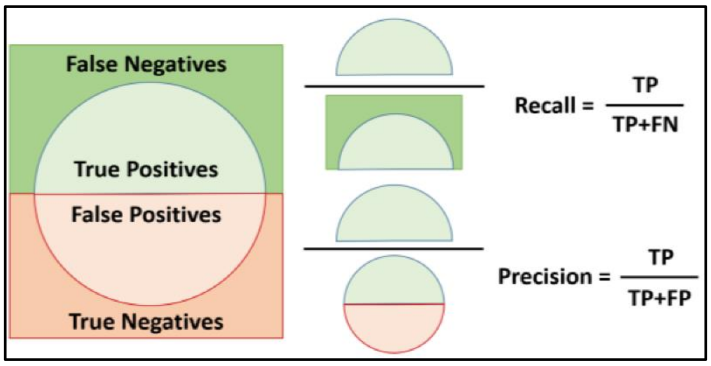


Figure 6: Precision and Recall

Precision for a prediction is usually only used for the mAP, if the IoU is more than a specified threshold. In mAP50, the prediction is considered successful, if the IoU is larger than 50%. When using mAP50-95, the precision is averaged for every 5% IoU from 50% to 95% (50%, 55%, 60%, …, 90%, 95%).

2.2.3. YOLO introduction

You Only Look Once (YOLO) is a vastly popular object detection architecture, that is utilized by many companies, projects, and individuals. YOLO is developed by Ultralitycs since the release of YOLOv5 in June 2020 (deci.ai).

The first version, YOLO was introduced to the artificial intelligence community in June 2016 by Joseph Redmon. YOLO's groundbreaking feature was the fact that only one network-pass was required instead of two or more. It also achieved an astonishing mAP of 63.4% on the PASCAL VOC2007 (a large dataset and a measuring base for image recognition models). Nevertheless, YOLO had still many limitations (i.e., not being able to predict more than two objects close to each other). YOLOv2 and YOLOv3 had improvements that led them to vastly outperform previous models. YOLO models were built to achieve real time speed with great accuracy. After YOLOv3, Redmon stopped working on YOLO, because he couldn't ignore the military applications and privacy concerns.

In 2020 others took up the project and created YOLOv4 and two months later Ultralytics published YOLOv5, which is still popular today. Fast forward to January 2023, Ultralytics released YOLOv8, a state-of-the-art object detection architecture, with astounding speed and great accuracy. YOLOv8 has 5 different scaled versions: YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), YOLOv8l (large), YOLOv8x (extra-large). The smaller versions are faster, but less accurate, while the larger versions are more accurate, but slower. Overall YOLOv8x, tested on the MS COCO dataset (a more recent larger dataset with millions of images and 9000 classes, used as a measurement base for image recognition models) achieved an mAP of 53.9% with an image size of 640 pixels with a speed of 280 FPS on a NVIDIA A100 GPU.

Because of its speed and high accuracy, in this project YOLOv8 was chosen to train and apply to automated fish monitoring. YOLOv8 has been trained on a large dataset of fishes and then tested on a day long sample of the footage of the camera at the Kisköre fish pass.

2.3. Annotation of fishes

Every neural network is trained on a dataset of carefully labelled images. The number of images is in correlation with the accuracy of the model most of the times, as deep learning models trained on vast datasets are superior in accuracy to the same models trained on fewer images (not accounting for overfitting). Annotating images is one of the most time consuming and tedious aspects of creating an artificial intelligence model. Annotating images is a manual process of labelling objects with classes and bounding boxes. Annotating can be an exasperatingly time-consuming exercise for one person in light of the considerable quantity of images involved. Therefore, it is common practice for annotation to be carried out by multiple individuals.

A sufficient dataset has the following traits not regarding the content of the images:

- a substantial amount of images,
- correct class and bounding box are applied,
- every instance of the objects is labelled,
- the instances of the classes are nearly equal.

This project required vast amounts of images of fish carefully labelled in the format of the input of YOLOv8. For every image a .txt file with the same name as the image file is needed, in which the data of the class and the bounding boxes are represented. The format of the label data is the following:

2	0.5	0.5	0.5	0.5
Class	Centre x	Centre y	Width	Height

The class is a number referencing to the class name in the classes file. The coordinate and size numbers are referring to the centre point and the size of the bounding box. It is important to note that all the coordinates are scaled to the width and height of the images, ensure that it stays relevant after resizing the image.

Calculating and writing down every number manually is a tedious task, hence there are many websites and applications giving a helping hand in annotating. Roboflow (https://roboflow.com/) is one of the most popular tools for image labelling. It is a user-friendly tool to not only annotate images fast but also train a model and deploy it right away. In this project Roboflow was used to annotate images and organize the files to fit the requirements of YOLOv8.

As a high school student, the author is not a fish biologist and cannot recognize fishes on a professional level. The annotation of the images, i.e., the manual recognition of fishes from video footages were performed with the help of a fish biologist of the Middle Tisza District Water Directorate (KÖTIVÍZIG), which is the responsible operator of the fish pass. The dataset had close to 800 images labelled in text files with 19 fish species such as carp and bream.

Nevertheless, the dataset had three major flaws:

- poor balance between the instances of fish species,
- plenty of fishes were not visible and only appeared as a blur,
- numerous images were inadequately annotated, and various fish were left unlabeled.

Seeing these flaws further work had to be done with the dataset. The fact that there were more smaller fish than larger fish caused the first issue. The first version of the dataset was imbalanced (Figure 7). Four classes were dominant, and the others didn't stand out much. This dataset caused the model to have great accuracy on the species that had more instances and low accuracy on the rest. This issue was solved by going over the dataset and removing the images that had many fishes of the classes with the most instances, especially common bleak (Szélhajtó Küsz). It was done this way and not by adding pictures, because the images from the Kisköre fish ladder could not be filtered to be taken of fishes with low instances.

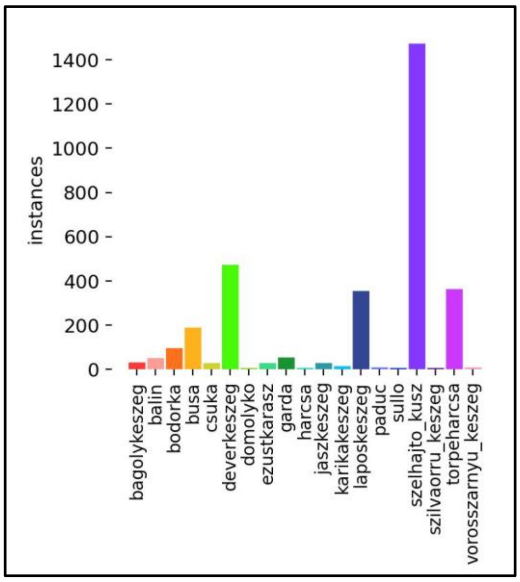


Figure 7: Instances of species

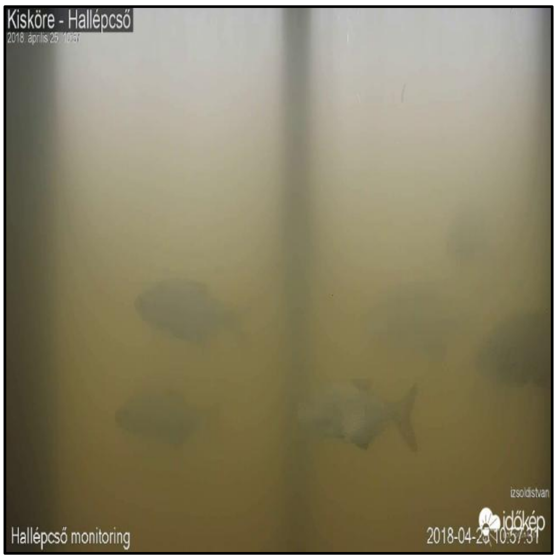


Figure 8: Image of Kisköre fish ladder

Due to water turbidity and fluctuating brightness under water, many fishes only appeared as a blur and weren't annotated (Figure 8). In some cases, not even the shape could be seen very well. The aim of the project requires this issue to be dealt with, since these fishes must be accounted for as well. A new classification, the "no class" class was created to solve the problem. Every fish that could not be seen were classified as "no class".

As stated in the third issue, many fishes weren't annotated either because they were a blur or because there were too many of them in one picture. This was solved by using the previously made "no class" class and by the deletion of images, where the fishes were perfectly visible, but they were unrecognisable for the author. These fishes weren't put in the "no class" class, since the "no class" is for blurs and not fishes that could be recognized but weren't annotated.

Although the final dataset after the annotation process was not ideal, it was more balanced than before (see Figure 9 – please disregard the discrepancy in numbers between Figure 7 and Figure 9, as it is due to the augmentation process explained in the subsequent chapter). The significant prevalence of common bleak does not impact the model's accuracy, as this species differs vastly from the other species.

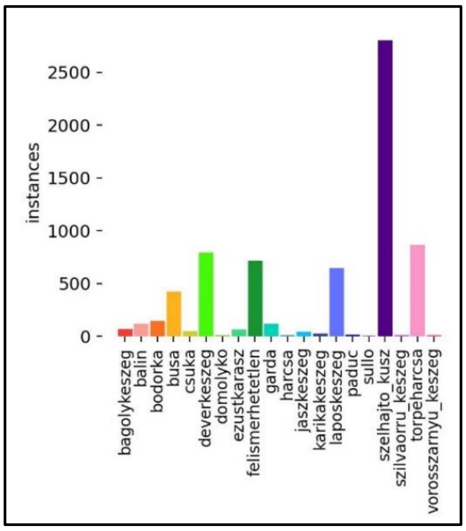


Figure 9: Instances of species enhanced

2.4. Pre-processing

2.4.1. Augmentation

Augmentation is a helpful process when making a dataset for a deep learning model. It is the process of adding changed images (Figure 10) to a dataset to multiply the number of pictures and avoid overfitting.

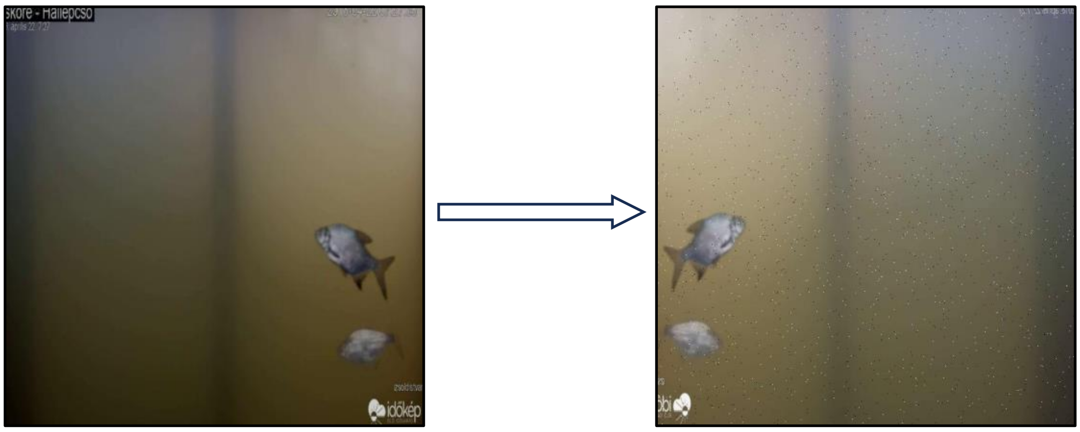


Figure 10: Augmentation

There are various augmentation steps, such as rotation, adding noise, adjusting brightness, blurring, cutout, etc. In the final dataset the following augmentation steps were used: horizontal and vertical flipping, grayscale – 25% of the images, brightness change – between -20% and +20%, crop – 15% maximum zoom, blur – up to 2 px, noise – up to 1% of pixels.

With the augmentation done, the number of images in the final dataset grew from around 800 to 2500. Although this is far from the astoundingly big datasets like MS COCO (with millions of pictures), it is sufficient enough to be used to train a model.

Additionally, the images were separated into training, validation, and testing sets. This separation is to test the model mid- and after training on untrained data. The separation ratio was 82% training, 12% validation, and 6% testing.

2.5. Training and testing

Training is the next step in creating an artificial intelligence model. The CNN model learns and adapts to the dataset by fine tuning itself based on the given dataset. As mentioned in 1.2., the training consists of many epochs, and an epoch consists of a forward pass, a backward pass and optimization. The image gets processed in the forward pass, and the CNN gives an output. This output is then compared to the ground truth (annotations given to the model in the dataset) to calculate how much information has the model lost with respect to the ground truth. This is loss, the lower its value, the better the model.

After calculating loss, the model performs a backward pass and calculates how much to change every weight and bias in the model to make the loss lower, thus its predictions similar to the ground truth. The optimizer uses these derived values to change the weights and biases of the layers. SGD (Stochastic Gradient Descent) is one of the if not the most basic optimizer. It changes the weights and biases by their derivatives multiplied by a learning rate. The learning rate is a hyperparameter and must be experimented upon by the training person to help the model learn. If the learning rate is too high, the loss will fluctuate through epochs, and if it's too low, the training will take a long time and the model might not reach the best performance in the give epochs. To make the optimizer more effective, methods can be used like momentum or learning rate decay. The most widespread optimizer is the Adam optimizer. The Adam optimizer calculates learning rate for every parameter (weight and bias) based on the changes done to them in the past epochs.

If done correctly, the model should pick up the features of the images and make a sufficient prediction. If, however, the model learns way too well, it might learn the features of the dataset and only the dataset. This is called overfitting, which is when the model can predict near perfectly on the images in the training dataset, but it cannot predict anything outside of the dataset. Several solutions exist to prevent overfitting, such as a dropout layer, which "turns off" and updating a percentage of the neurons in the hidden layers, thus making the learning even throughout the model.

Feeding images in batches makes training faster and more efficient. When a batch is given to the model, more images are given to the model at the same time as input to train on. Batch size is a hyperparameter and it means how many images are given to the model in one pass. Batch size can range from 1 to 128 or more. A common batch size is 16, 32 or 64. Nevertheless, batch size, being a hyperparameter is model specific and must be experimented with.

Fortunately, YOLOv8 makes training easier by being a premade model and having approximately good hyperparameters as default. Hyperparameters that might have big effect when changed is the optimizer, batch size, number of epochs and image size. Choosing the premade scaled YOLOv8 model is also a trial-and-error process. This research involved many attempts to train a fish recognizer model with the best accuracy.

Without testing and validation, measuring the accuracy of the model solely from the training data will provide unrealistic information, as the model is trying to make its predictions like the ground truth of the training data. Testing on unseen data will result in more realistic measurements, since when used in production it will only get unseen data. Measuring accuracy and loss on testing data (also called validation data in some literature) is used at the end of the training to see how the final model preforms on unseen data. Throughout training the validation data is fitted to the model to see how much it improved in every epoch.

After augmentation, the dataset is split into training, testing and validation datasets. Usually, the split ratio is 70% : 20% : 10% (train : validation : test). However, it is also up to the training personnel to choose the ratio, for example if the dataset were to be too small. (en.wikipedia.org)

2.6. Postprocessing

After training the model and it is ready to be used, displaying and gaining statistical data might be challenging. The goal of this project is to count fishes, obtain their trajectories and their sizes. To achieve these goals, detecting and classifying fishes would not be enough, they must be tracked and given an id. Fortunately, YOLOv8 has a built-in tracking system. On the contrary, it only accepts video as an input. The results of the tracking system consist of the following:

ID	Class	Bounding Box (top left, bottom right coordinates)	Confidence

These results are processed and used to draw the bounding boxes on the video with the id, class, and confidence usually above it. Classifying a fish into one class proved difficult, because of the fluctuating brightness and blurriness of the fish. Some appeared as a blur but became perfectly visible as they came closer, resulting in a change in the classification of the fish. It is important to decide what method to use for choosing the classification of the fish. The following method was used to choose:

$$Maximum(average\ confidence\ per\ class * length\ of\ appearance\ of\ the\ class)$$

After the classification, the direction of the fish is calculated. Next the trajectory and the size of the fishes are saved for statistics. At the end of the postprocessing, three major statistics are obtained: sizes of species, fish trajectories and the number of fishes per class.

Except for displaying the statistics this process will be performed on every frame of the video, meaning that if the aim is to recognize real time, the predicting and postprocessing process has to be fast. Since both of these require vast computational space and speed, real time recognition can only be achieved on GPUs.

2.7. Programming

For the research a fair bit of programming had to be done. The programming was carried out in the Python programming language and was ran on the GPUs of Google Colab and Kaggle. Fortunately, the training didn't require a lot of code, since YOLOv8 makes it relatively easy. The next programming segment was the video processing part. YOLOv8 supports tracking functions, and only a few lines of code are needed to make it work. Obtaining the data and drawing the results, however, was by far the largest segments of the programming. Drawing statistics was also a significant part of the coding efforts.

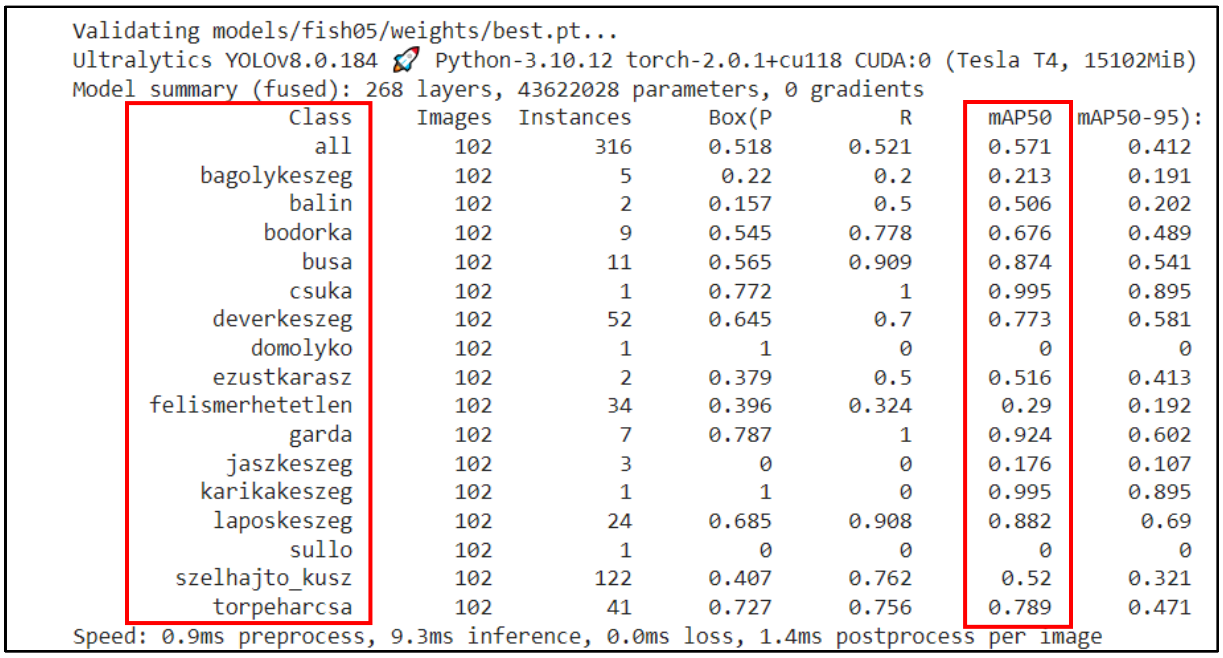
A class/object was made from the code for easier use and to potentially make a Python package (similar to Numpy, Matplotlib, etc) out of it in further development. The code that downloads videos from www.idokep.hu was made by Gergely Tikász, but was fitted to the fish recognizer tool by the author. Through this code, the model recognizes the 3-4-second-long videos, then the code draws a video and saves the data for further statistics frame by frame. All in all the code is close to 600 lines.

3. Results

3.1. Developed models

Many models have been created throughout the training phase, as training AI models is a trial-and-error method. The first training on the original dataset was done to test the training time, and capability of the model. The training had the following parameters: model: YOLOv8 nano – the smallest and fastest YOLOv8 model, epochs: 10, batch size: 16, image size: 640x640. The test was a failure, seeing that the training time was 4 hours and the mAP50 of the model was 0.35. It took 4 hours for the reason that it was done on the CPU (central processing unit) of a ThinkPad and far too much computation had to be done. Although 4 hours isn't long in contrast of AI training time, it is long considering that 10 epochs took this long, when the model is supposed to be trained on hundreds of epochs. The accuracy of the model was low, because the number of epochs and batch size was low. From this point onwards, a free Tesla T4 GPU (graphics processing unit) in Google Colab was used to make training faster.

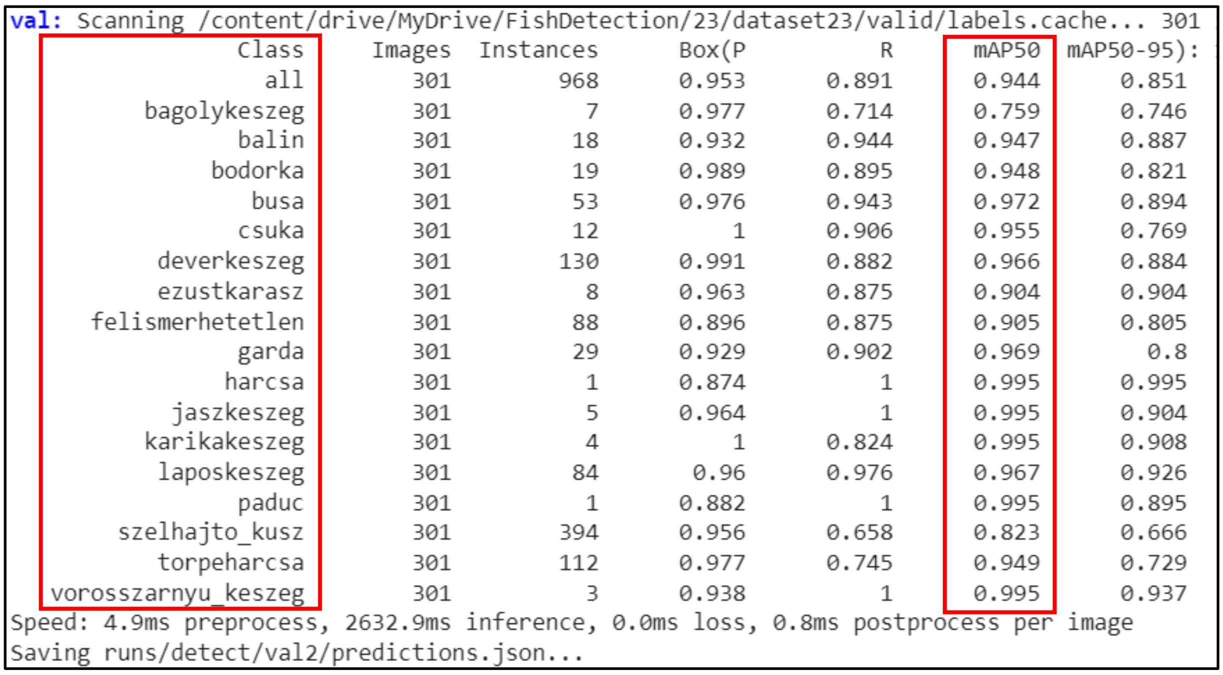
The next training was done on a better dataset (still not the final), on YOLOv8 large, the 2nd largest out of the five models, 200 epochs, 32 batch size and 480x480 image size. This model did a lot better than the previous with a final mAP50 of 0.57. Although this number may seem low, the imbalance of the dataset must be accounted for. The model achieved lower accuracies on classes with few instances and high accuracies on classes with more instances (Figure 11). In spite of this fact the model achieved 0.955 mAP50 on pikes (csuka), a species with low appearance in the dataset. The reason for this is that pikes differ greatly from other fishes, and they are perfectly seen on the images. This dataset also had the imperfections mentioned in 1.3. but on a lower scale.



```
    Validating models/fish05/weights/best.pt...
    Ultralytics YOLOv8.0.184 🚀 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)
    Model summary (fused): 268 layers, 43622028 parameters, 0 gradients
                     Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
                       all        102        316      0.518      0.521      0.571      0.412
              bagolykeszeg        102          5       0.22        0.2      0.213      0.191
                     balin        102          2      0.157        0.5      0.506      0.202
                   bodorka        102          9      0.545      0.778      0.676      0.489
                      busa        102         11      0.565      0.909      0.874      0.541
                     csuka        102          1      0.772          1      0.995      0.895
               deverkeszeg        102         52      0.645        0.7      0.773      0.581
                  domolyko        102          1          1          0          0          0
               ezustkarasz        102          2      0.379        0.5      0.516      0.413
          felismerhetetlen        102         34      0.396      0.324       0.29      0.192
                     garda        102          7      0.787          1      0.924      0.602
                jaszkeszeg        102          3          0          0      0.176      0.107
              karikakeszeg        102          1          1          0      0.995      0.895
               laposkeszeg        102         24      0.685      0.908      0.882       0.69
                     sullo        102          1          0          0          0          0
             szelhajto_kusz        102        122      0.407      0.762       0.52      0.321
               torpeharcsa        102         41      0.727      0.756      0.789      0.471
    Speed: 0.9ms preprocess, 9.3ms inference, 0.0ms loss, 1.4ms postprocess per image
```

Figure 11: 2nd training accuracy per class

After this model, the dataset has been corrected (at least as far as the author could – mentioned in 2.3.). It had problems, that could not be solved within the limits of this research, as some fishes were incorrectly labelled, which consequently impacted the model's accuracy. Nevertheless, the new dataset consisted of close to 800 pictures, which is after augmentation, raised to 2556 images with 19 fish species not including the “no class” class. The train, validation, test split ratio was set to 82%, 12%, 6%. This ratio was deemed good, since the whole dataset is still small. Using this dataset a YOLOv8 large model was trained on 200 epochs, with a batch size of 32, image size of 480x480 pixels with the Adam optimizer. The model achieved an astonishing mAP50 of 0.94. This is an average across all classes, meaning that most of the model predicted most of the classes with an mAP50 of over 0.9 (Figure 12). The weakest preforming classes were the white bream (bagolykeszeg) and the common bleak (szélhajtó küsz), since even though most issues of the dataset were corrected, false labelling was not. Nevertheless, the model performed near perfectly on the validation test, which meant, that it performed according to the ground truth. This means, if the fish were labelled more accurately, the predictions would be more precise as well.



```
val: Scanning /content/drive/MyDrive/FishDetection/23/dataset23/valid/labels.cache... 301
                   Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
                     all        301        968      0.953      0.891      0.944      0.851
             bagolykeszeg        301          7      0.977      0.714      0.759      0.746
                    balin        301         18      0.932      0.944      0.947      0.887
                  bodorka        301         19      0.989      0.895      0.948      0.821
                     busa        301         53      0.976      0.943      0.972      0.894
                    csuka        301         12          1      0.906      0.955      0.769
              deverkeszeg        301        130      0.991      0.882      0.966      0.884
              ezustkarasz        301          8      0.963      0.875      0.904      0.904
          felismerhetetlen        301         88      0.896      0.875      0.905      0.805
                    garda        301         29      0.929      0.902      0.969        0.8
                   harcsa        301          1      0.874          1      0.995      0.995
               jaszkeszeg        301          5      0.964          1      0.995      0.904
             karikakeszeg        301          4          1      0.824      0.995      0.908
              laposkeszeg        301         84       0.96      0.976      0.967      0.926
                    paduc        301          1      0.882          1      0.995      0.895
            szelhajto_kusz        301        394      0.956      0.658      0.823      0.666
              torpeharcsa        301        112      0.977      0.745      0.949      0.729
        vorosszarnyu_keszeg        301          3      0.938          1      0.995      0.937
Speed: 4.9ms preprocess, 2632.9ms inference, 0.0ms loss, 0.8ms postprocess per image
Saving runs/detect/val2/predictions.json...
```

Figure 12: Final model accuracy per class

With this high accuracy, the model was decided to be the final model, and it is ready to be used on the footage of the Kisköre fish ladder.

3.2. Sample application

After training the artificial intelligence model and programming postprocessing, the next step was to apply the model to the footage of the Kisköre fish ladder streamed on www.idokep.hu. The length of the analysis was decided to be one day long, to get realistic but overwhelming amount of data. A short video (0.5-1 hour long) wouldn't provide with enough data, meaning that tangible conclusions won't be able to be drawn. A longer video (e.g., multiple days or a week) would provide excessive amount of data that makes conclusions hard to be drawn, as many meaningful information would be overshadowed by the sheer amount of data.

The 3-4-second-long videos were continuously downloaded from the internet and analysed real time for one day. Even though the analysis was done in two parts, because of the large amount of analysed videos filled up the online HDD (hard disc drive). Trough the analysis the number of fishes were counted based on swimming direction, their sizes were measured, and their trajectories were obtained from the video. The analysis was carried out on October 25, 2023.

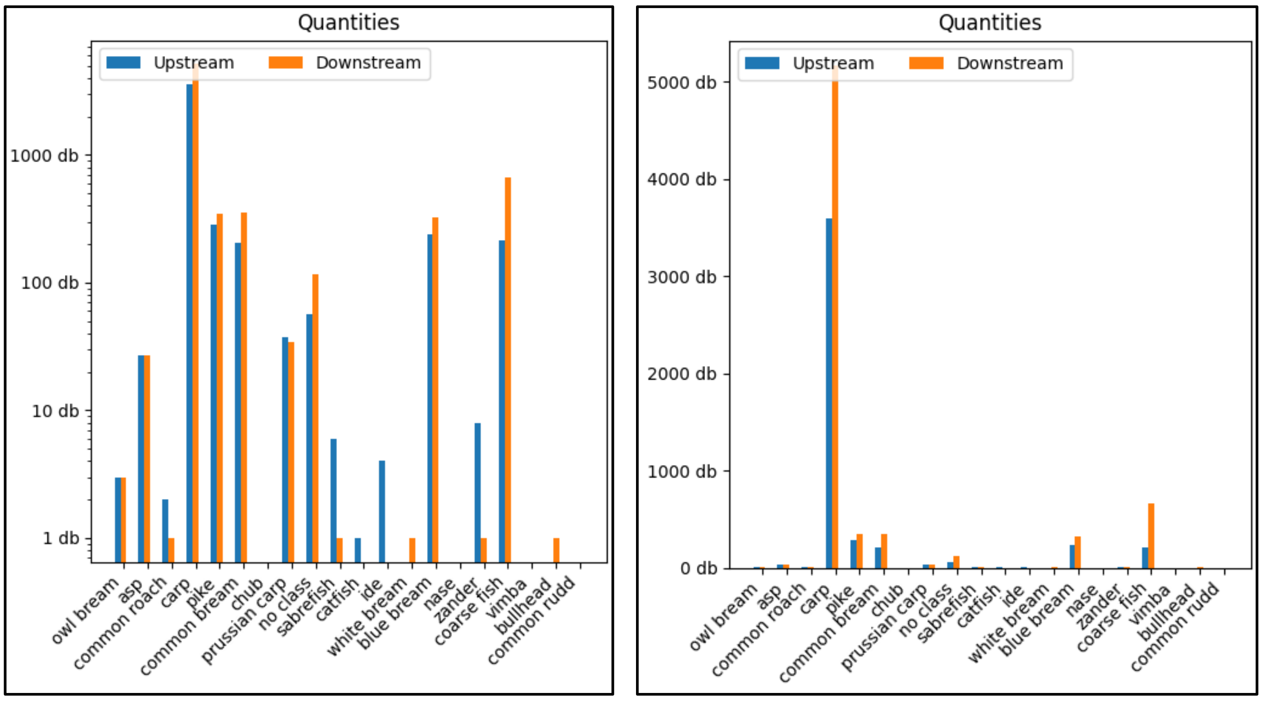


Figure 13: Quantities with logarithmic scale (left) and with non-logarithmic scale (right)

Figure 13 depicts the quantities of the fishes based on their swimming direction. Left means upstream, and right is downstream direction of swimming, respectively. On the horizontal axis are the classes of the fishes, and the vertical axis is the quantity of the fishes with logarithmic scale. The figure depicts over 8500 carp swimming in front of the observer window on that day. The following species were significantly less common, with the coarse fish/common bleak presenting the next highest count of approximately 1000 individuals. The third most common species was a three-way tie between pikes, common breams and blue breams, appearing around 900 times each. The direction of the swimming of fish can also be seen in the diagram of Figure 13. The directions were calculated by subtracting the coordinates of starting point from the ending point.

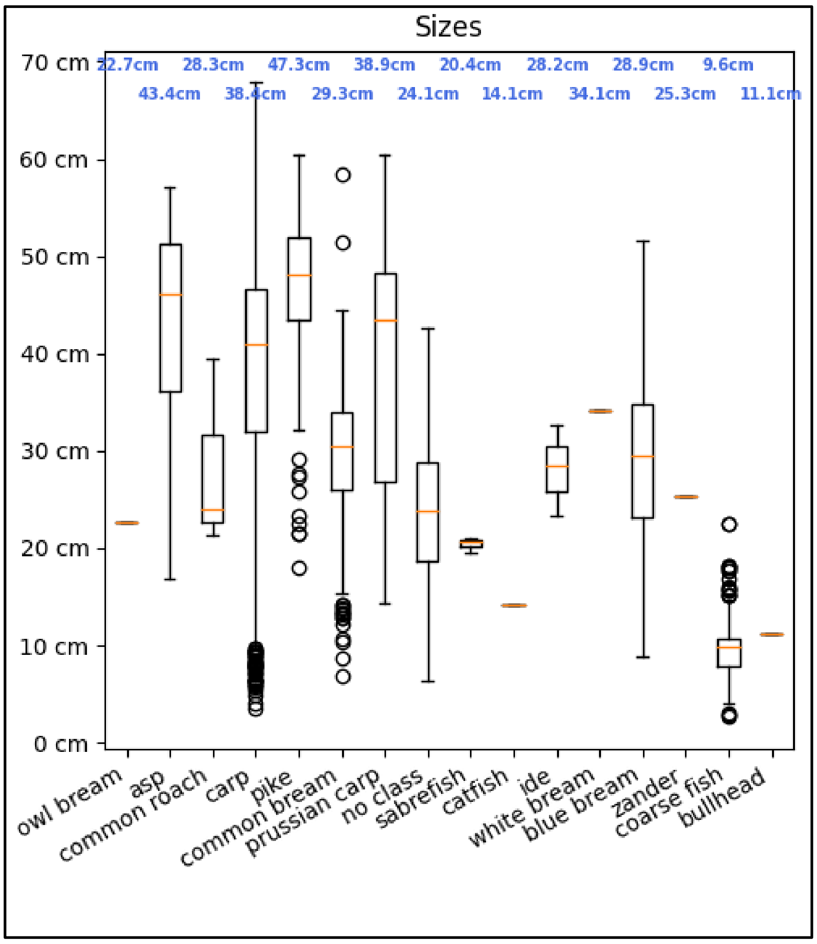


Figure 14: Typical sizes of fish species

Figure 14 illustrates the typical sizes of fish species. The box diagram indicates the average and outliers of sizes. As seen in the figure, the largest recorded fish species on average was the pike with an average fork length of 47.3 cm. The diagram also shows that there were some pikes, that were extraordinarily smaller than their peers. This could also be said of the carps, a species with a wide range of sizes. Although on average carps were the fourth largest with a fork length of 38.4 cm, the biggest fish (~ 67 cm) and one of the smallest fishes (~ 5 cm) were recorded in that species. Although shown, some species like the catfish and owl bream did not appear many times throughout the analysis, consequently meaningful size data could not be obtained. It should be mentioned that the size measurements are relative to the size of the area of the footage, which was only approximated. The size of the footage is approximately 96 x 54 cm based on appearing objects in various photos taken from multiple angle.

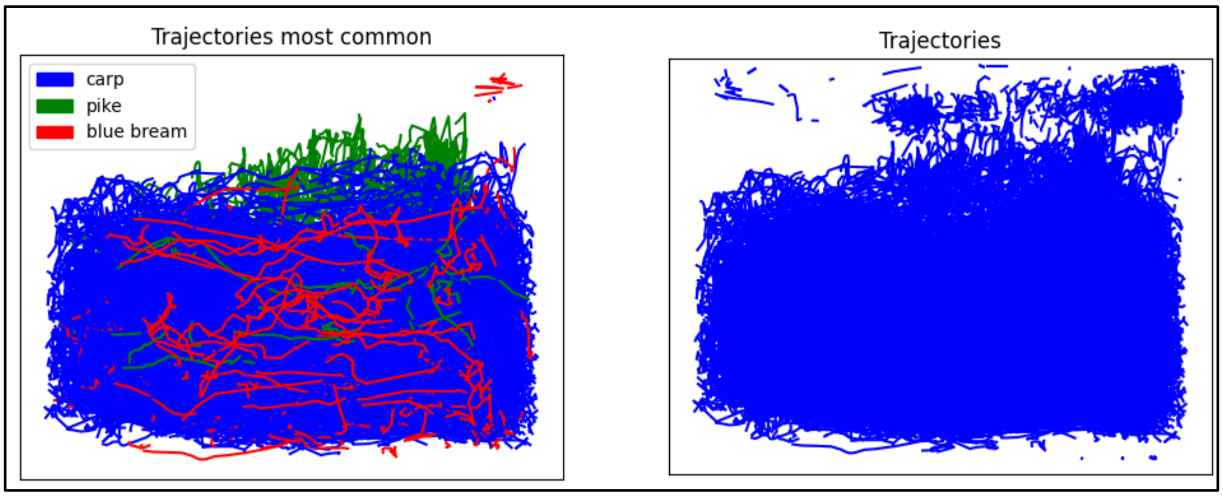


Figure 15: Trajectories

The figures above depict the trajectories of recognized fish. The trajectory data was calculated from the bounding boxes of the fish. As seen in the right part of the figure, most of the fish were swimming on the lower part of the video. On the left side of the figure the trajectories of the three most common fish species are shown.

Although these diagrams show several meaningful information, it is not the aim of the research to draw conclusions from the collected data. The main objective was to determine the capability of artificial intelligence to recognize fishes in unconstrained underwater environments. In light of that, the accomplishments of the fish recognizer model are the following:

- It could recognize most fishes and classify them with a high confidence score.
- Even though not all fishes were classified, they were taken into account by sorting them into the "no class" category.
- The program could obtain the directions, trajectories, and characteristic sizes of the fishes.

As stated in Chapter 1.1., fish monitoring is done by classifying fishes and measuring their fork length. The program could extract both of these information, and in addition the movement was recorded as well. Furthermore, the model could produce a video that showed each fish classified (Figure 16).

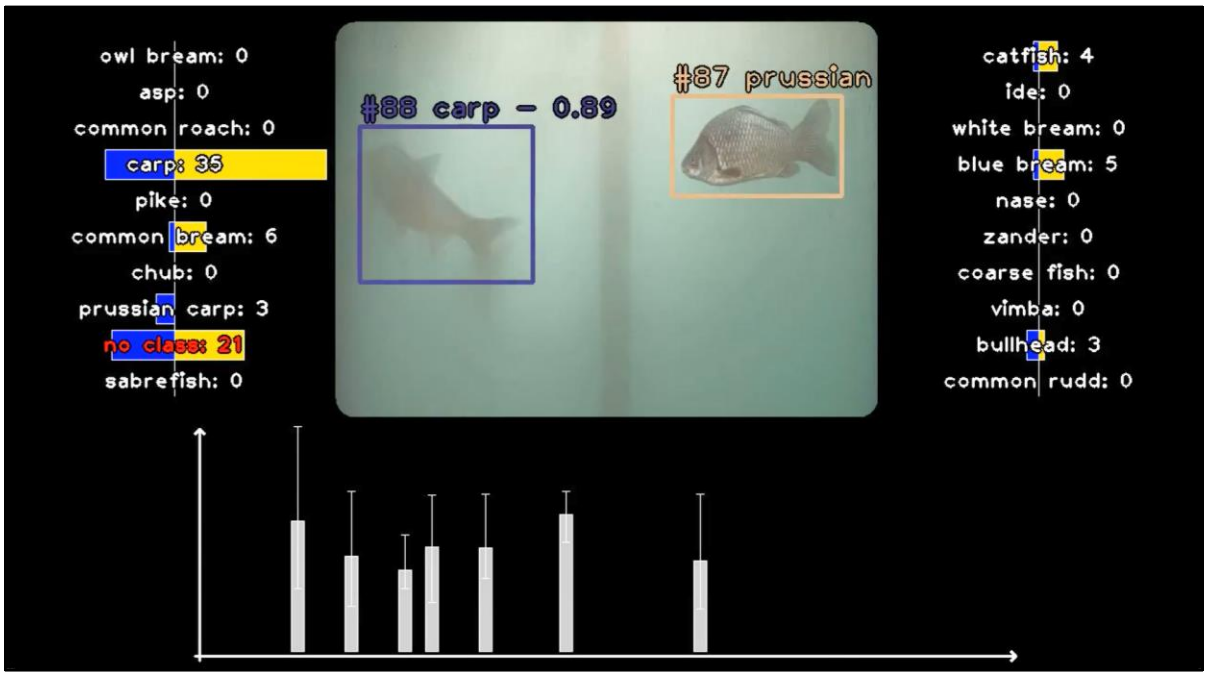


Figure 16: The processed video by the code

In conclusion the model accomplished most of the objectives that were set. If trained on a larger and more accurate dataset, most fish would be recognizable by the model without issue, despite changes in uncontrolled environments. Addressing the concerns mentioned in section 2.1, artificial intelligence and deep learning has a promising potential to assist scientists in monitoring fish populations.

4. Discussion

4.1. Main novelty of the study

Many studies have conducted research involving the application of artificial intelligence on underwater footage. In recent years the number of studies has continuously risen, with most studies using the methods detecting, measuring, tracking, and classification (Barbedo, et al., 2020).

Detection and quantification can be done underwater to assess fish populations, as one of the most basic, but important parts of ecological monitoring. Many studies have applied artificial intelligence in their studies in order to detect and count fishes. Despite the fact that many research have applied AI in unrestrained underwater environments (Labao & Naval, 2019; Laradji, et al., 2021), most of them proposed a framework (Liu, et al., 2021), or detected only a few species (Aliyu, et al., 2020).

Measuring fish properties non-invasively proved difficult in uncontrolled underwater habitats, as most of the studies have only studied it out of water or in controlled environments. The ones, that tried uncontrolled places have only measured the properties of a few species (Díaz, et al., 2020).

Tracking fishes to observe their movements is also an important role in ecological monitoring. Determining the trajectories of the fishes can help determine diseases among the populations. Almost all the studies have been studying fishes in controlled environments, and only a handful have examined them in uncontrolled settings. However, these studies were also limited to tracking just one or two fish species. (Abe, et al. 2021).

When multiple species are present, simply counting the fishes might not be enough to draw meaningful conclusions. That is why determining the species of the fishes is important in ecological monitoring. Although an increasing number of studies have explored underwater environments, the majority have not. The number of species classified ranged from 3 (Coro & Walsh, 2021) to 20 (Villon, et al., 2018).

The main novelty of this study compared to previous ones is the fact that the model presented in this paper has been used on images of uncontrolled underwater setting and the fact that it does every methods mentioned above in near real time. The model recognizes a fish, identifies its species, measures its size, tracks its movements, and counts it. Moreover, the model can recognize 19 fish species common in the study area such as pike, carp, or common bleak.

4.2 Evaluation of results

The results, seen in chapter 3.2., show that close to 12,000 fishes swam through the fish ladder on the day of the sample application. The species with the most fishes was the carp, outstanding by far with close to 8,000 individuals. The next biggest species in population size was the common bleak / coarse fish with close to 1,000 individuals. However, some issues make the results uncertain.

As seen in Figure 3, the canal splits into two before the fishes can be seen through the window. Also seen in Figure 2, there are two windows, on at the top and one at the bottom. The camera only watches the top window. For these two reasons it is safe to say, that only a portion of the fishes are seen on the footage. Accurate approximations cannot be made to account for these issues, but the goal of the study does not require them, as the goal is to see how well deep learning can perform underwater in uncontrolled environments. Moreover, the physical size of the footage was approximated based on footages and pictures. The size of the footage was used to measure the typical sizes of the fishes trough analysis. It is important to note that this is only an approximation, not measured.

As mentioned in 2.6., the method for determining the class of a fish is a challenging task, as the tracking method changes the classes of the fishes as they get closer or farther from the camera. For statistics a true class must be determined to not count the fishes twice or more times. The current method of calculating true class is seen in Postprocessing – 2.6. While this method can eliminate false classes caused by water turbidity, it may also mistakenly remove correct detections. For instance, a fish may initially be classified as no class because it is far from the camera, but later be correctly identified as it gets closer; the filter would falsely eliminate the correct detection if it only lasted for a short time.

Another issue came up as the one-day-long sample application was performed. The footage of the fish ladder is uploaded to www.idokep.hu in a continuous stream consisting of 3-4-second-long videos. The program gets these videos and feeds it to the tracking method. This is causing a problem since the tracking function can only track in one video, hence, it resets the ID counter to 0 for each video. Which means that fishes that took longer than 4 seconds to cross the window were assessed twice or more times. This issue requires further assessment in order to enhance the precision of quantification predictions. Nonetheless, the conclusion that way more carps swam through the fish ladder can't be drawn as the issue with the tracking method proposes that carps may have been slower, than other fishes, hence one fish appeared on more than one 3-4-second-long video, thus counted more than once. But even taking this issue into account, approximately 1500-2500 carps swam through the fish ladder throughout the day.

In conclusion, four major aspects of the model make the results uncertain:

- fishes aren't labelled completely correctly,
- only a portion of the fish pass can be seen on the video footages,
- the true class calculation filters out scenarios that shouldn't be,
- some fishes are assessed more times because of the tracking method.

4.3. Applicability limitations

While the model proves proficient in extracting various fish statistics, its application is currently restricted to footage exclusively obtained from the Kisköre fish ladder. This confinement arises because the model was trained only on images obtained from this particular setting.

Resolving this challenge requires training the model on a more diverse and inclusive dataset containing a broader spectrum of environments.

Furthermore, incapacity of the model to perform real-time analyses without a high-powered GPU is evident. When tested on a CPU of a ThinkPad device, a 5-minute video required an extensive 3-hour analysis time. In contrast, the analysis on a Tesla T4 GPU took less than 8 minutes. However, depending on the research team's budget, an investment in high-performing GPUs or renting online GPUs may significantly raise the cost of the examination. One potential solution to this challenge involves exploring faster CNN models capable of preserving accuracy while reducing processing time.

Another constraint stems from the substantial light obstruction in water, which limits the visibility range to only nearby fish. This phenomenon leads to inability of the footage to capture fish in more extensive water bodies, such as lakes, depicting merely a fraction of the fish population. As a result, the model performs optimally within constrained water bodies, particularly narrow canals or fish ladders.

4.4. Future development directions

The current phase of the fish recognition system, while marking progress in understanding fish behaviors within the Kisköre fish ladder, encounters various limitations. To refine and broaden its application, several avenues of development are essential.

The existing challenge primarily revolves around the limitation and inadequacy of the dataset. To make the model richer of diverse fish species and behaviors, there is a critical need to expand the dataset. Collaboration with environmental agencies, scientific departments, and fisheries would play an important role in evolving the dataset. This expanded data will be important for the AI to recognize and interpret a wider spectrum of fish behavior patterns.

The current reliance on high-performance GPUs for near real-time functioning poses a significant constraint. The forthcoming focus aims to reconfigure the model architecture for compatibility across a broader range of hardware configurations. This optimization seeks to enable the AI to operate efficiently across various computing setups, ensuring its applicability on diverse computational systems.

While the current application can only be used for the specific conditions of the Kisköre fish ladder, future enhancements seek to transcend these limitations. Efforts will concentrate on enhancing adaptability and versatility of the model, enabling it to function effectively in various aquatic ecosystems and fish migration systems.

The tracking methodology sometimes leads to multiple assessments of the same fish, potentially compromising the precision of behavioral analysis. Future initiatives will revolve around fine-tuning the tracking algorithms to mitigate such occurrences and enhance tracking accuracy. Advanced tracking methodologies, including sophisticated object recognition and refined motion prediction, will be explored to ensure precise fish tracking without redundancy.

In conclusion, solving these challenges through dedicated research and development will not only advance the capabilities of the fish recognition AI but also expand its potential applications in ecological monitoring and comprehensive analysis of fish behavior. These future developments aim to propel the model toward greater adaptability, accuracy, and wider applicability in diverse aquatic settings.

4.5. My contribution

As a 16-year-old high school student I enjoyed taking part in a university research. My contributions to the study are the following:

- researched past literature,
- corrected the annotation of nearly 1000 labelled images for the AI model,
- trained models and picked the best one to work with,
- wrote a program to analyze and process videos from the Kisköre fish ladder and draw statistics,
- implemented the model into the code that downloads the videos from the internet.

Throughout this project I learned Hungarian freshwater fishes, I discovered the Kisköre fish ladder and hydropower plant, and I studied artificial intelligence. After reviewing and correcting the annotated images I picked up on recognizing fishes, obviously not at a professional level yet. I began to like fishes and working on recognizing them. I also studied a lot about artificial intelligence, deep learning and CNNs. Although I had prior knowledge about neural networks, I learned so much more, that I realized that I still know less than 1% of the subject, which I believe is the beginning of real knowledge. I also learned how to write a scientific paper like this, and how to manage project time. I enjoyed making this research and gained a lot of experience.

Acknowledgements

Above anything I owe my thanks to my supervisors and consultants, Dr. Sándor Baranya and Gergely Tikász for their help and advice during the preparation of my thesis. I want to give my thanks to Middle Tisza District Water Directorate (KÖTIVÍZIG) for the annotated images, and to Gergely Tikász for the code that downloads the videos from the internet.

Figures

Bibliography

Abe, S.; Takagi, T.; Torisawa, S.; Abe, K.; Habe, H.; Iguchi, N.; Takehara, K.; Masuma, S.; Yagi, H.; Yamaguchi, T.; et al. Development of fish spatio-temporal identifying technology using SegNet in aquaculture net cages. Aquac. Eng. 2021, 93, 102146.

Aliyu, I.; Gana, K.J.; Musa, A.A.; Adegboye, M.A.; Lim, C.G. Incorporating Recognition in Catfish Counting Algorithm Using Artificial Neural Network and Geometry. KSII Trans. Internet Inf. Syst. 2020, 14, 4866–4888.

Barbedo, J.G.A. A Review on the Use of Computer Vision and Artificial Intelligence for Fish Recognition, Monitoring, and Management. Fishes 2022, 7, 335.

CA. Fausch, K.D., Lyons, J., Karr, J.R., Angermeier, P.L., 1990. Fish communities as indicators of environmental degradation. Am. Fish. Soc. Symp. 8, 123–144

Coro, G.; Walsh, M.B. An intelligent and cost-effective remote underwater video device for fish size monitoring. Ecol. Inform. 2021, 63, 101311.

Islam, M.J.; Xia, Y.; Sattar, J. Fast Underwater Image Enhancement for Improved Visual Perception. IEEE Robot. Autom. Lett. 2020, 5, 3227–3234.Zhou, C.; Lin, K.; Xu, D.; Chen, L.; Guo, Q.; Sun, C.; Yang, X. Near infrared computer vision and neuro-fuzzy model-based feeding decision system for fish in aquaculture. Comput. Electron. Agric. 2018, 146, 114–124.

J.C. Kurtz, L.E. Jackson, W.S. Fisher, Strategies for evaluating indicators based on guidelines from the Environmental Protection Agency's Office of Research and Development., Ecol. Indic., 1 (2001), pp. 49-60

Kinsley, H., Kukieła, D., Neural Networks from Scratch in Python. 2020

Labao, A.B.; Naval, P.C. Cascaded deep network systems with linked ensemble components for underwater fish detection in the wild. Ecol. Inform. 2019, 52, 103–121.

Laradji, I.H.; Saleh, A.; Rodriguez, P.; Nowrouzezahrai, D.; Azghadi, M.R.; Vazquez, D. Weakly supervised underwater fish segmentation using affinity LCFCN. Sci. Rep. 2021, 11, 17379.

Liu, H.; Liu, T.; Gu, Y.; Li, P.; Zhai, F.; Huang, H.; He, S. A high-density fish school segmentation framework for biomass statistics in a deep-sea cage. Ecol. Inform. 2021, 64, 101367.

Oberdorff, T., & Hughes, R. M. (1992). Modification of an index of biotic integrity based on fish assemblages to characterize rivers of the Seine Basin, France. Hydrobio-logia, 228, 117–130.

Rico-Díaz, J.; Rabuñal, J.R.; Gestal, M.; Mures, O.A.; Puertas, J. An Application of Fish Detection Based on Eye Search with Artificial Vision and Artificial Neural Networks. Water 2020, 12, 3013.

Saleh, A.; Sheaves, M.; Jerry, D.; Azghadi, M.R. Applications of Deep Learning in Fish Habitat Monitoring: A Tutorial and Survey. arXiv 2022, arXiv:2206.05394.

Seymour A. Papert. The summer vision project. AIM-100, 1966.

Villon, S.; Mouillot, D.; Chaumont, M.; Darling, E.S.; Subsol, G.; Claverie, T.; Villéger, S. A Deep learning method for accurate and fast identification of coral reef fishes in underwater images. Ecol. Inform. 2018, 48, 238–244.

Water Framework Directive, 2000/60/EC

Zhou, C.; Yang, X.; Zhang, B.; Lin, K.; Xu, D.; Guo, Q.; Sun, C. An adaptive image enhancement method for a recirculating aquaculture system. Sci. Rep. 2017, 7, 6243.

Internet references

cermaq.com
https://www.cermaq.com/news/ifarm---cermaq-towards-individual-based-farming
[viewed at 2023.10.28.]

deci.ai
https://deci.ai/blog/history-yolo-object-detection-models-from-yolov1-yolov8/
[viewed at 2023.10.15]

fao.org
https://www.fao.org/fishery/static/eifaac/wpfmfw/DraftGuidelinesMonitoringFishFreshwaters.pdf
[viewed at 2023.10.19.]

fish.ai
https://fishial.ai/
[viewed at 2023.10.28.]

kotiweb.vizugy.hu
https://kotiweb.vizugy.hu/doksik/hallepcso.pdf
[viewed at 2023.10.05.]

wikipedia.org

https://en.wikipedia.org/wiki/Training,_validation,_and_test_data_sets

[viewed at 2023.10.16.]

Appendix

```python
from ultralytics import YOLO
import cv2
import supervision as sv
from collections import Counter
from statistics import mean
from PIL import Image as Img
import matplotlib.colors as mcolors
from matplotlib import pyplot as plt
import numpy as np
import pickle
from matplotlib.ticker import FormatStrFormatter

class FishDetectorFunctions:
  def __init__(self):
    pass

  def roundImg(self, img, ratio):
    img = np.array(img).astype(np.uint8)
    imgw = img.shape[1]
    imgh = img.shape[0]

    # mask data
    radius = int(imgh * ratio / 2)
    w = int(imgh * ratio)
    h = int(imgh * ratio)
    mask_base = np.zeros_like(img[0:h, 0:w])
    img_mask = np.ones_like(img)
    img_mask *= 255

    # mask1
    mask1 = cv2.circle(mask_base.copy(), (w//2,h//2), radius, (255,255,255), -1)
    cv2.rectangle(mask1, (0,h), (w, h//2), (255,255,255), -1)
    cv2.rectangle(mask1, (w//2,0), (w,h), (255,255,255), -1)

    # mask2
    mask2 = cv2.circle(mask_base.copy(), (w//2,h//2), radius, (255,255,255), -1)
    cv2.rectangle(mask2, (0,h), (w, h//2), (255,255,255), -1)
    cv2.rectangle(mask2, (0,0), (w//2,h), (255,255,255), -1)

    # mask3
    mask3 = cv2.circle(mask_base.copy(), (w//2,h//2), radius, (255,255,255), -1)
    cv2.rectangle(mask3, (0,0), (w, h//2), (255,255,255), -1)
    cv2.rectangle(mask3, (w//2,0), (w,h), (255,255,255), -1)

    # mask4
    mask4 = cv2.circle(mask_base.copy(), (w//2,h//2), radius, (255,255,255), -1)
    cv2.rectangle(mask4, (0,0), (w, h//2), (255,255,255), -1)
    cv2.rectangle(mask4, (0,0), (w//2,h), (255,255,255), -1)

    # add masks
    img_mask[0:h, 0:w] = mask1
    img_mask[0:h, imgw-w:imgw] = mask2
    img_mask[imgh-h:imgh, 0:w] = mask3
    img_mask[imgh-h:imgh, imgw-w:imgw] = mask4

    # put alpha channel on img
    #result = cv2.cvtColor(img.copy(), cv2.COLOR_BGR2BGRA)
    #result[:,:,3] = img_mask[:,:,0]
    #result = cv2.bitwise_and(img, img_mask)
    result = img * (img_mask / 255)

    return result.astype(np.uint8)

  def placeStatLabel(self, frame, label, xy, ysize):
    x, y = xy

    text_size = cv2.getTextSize(text=label, fontFace=cv2.FONT_HERSHEY_PLAIN, fontScale=1.5, thickness=2)
    ty = y + text_size[0][1] // 2
    tx = x - text_size[0][0] // 2
    cv2.putText(frame, label, (tx, ty), cv2.FONT_HERSHEY_PLAIN, 1.5, (0,0,0), 5)
    if 'no class' in label:
      cv2.putText(frame, label, (tx, ty), cv2.FONT_HERSHEY_PLAIN, 1.5, (255,0,0), 2)
    else:
      cv2.putText(frame, label, (tx, ty), cv2.FONT_HERSHEY_PLAIN, 1.5, (255,255,255), 2)

    return frame

  def calculate_box_center(self, xyxy):
    centerx = xyxy[0] + (xyxy[2] - xyxy[0]) // 2
    centery = xyxy[1] + (xyxy[3] - xyxy[1]) // 2
    return (centerx, centery)

  def add_to_tuple(self, tup, index, num):
    tup_list = list(tup)
    tup_list[index] += num
    return tuple(tup_list)

  def draw_graph(self, frame, size, video_bottom):
    width, height = size

    x1 = (len(frame[0]) - width) // 2
    y_sep = (len(frame) - video_bottom - height) // 2
    y1 = video_bottom + y_sep
    x2 = x1 + width
    y2 = y1 + height

    # draw lines
    cv2.rectangle(frame, (x1,y1), (x1,y2 + 5), (255,255,255), 2)
    cv2.rectangle(frame, (x1 - 5,y2), (x2,y2), (255,255,255), 2)

    # draw triangles
    pts = np.array([(x1 - 5, y1 + 3), (x1, y1-2)])
    cv2.drawContours(frame, [pts], 0, (255,255,255), 2)
    pts2 = np.array([(x1 + 5, y1 + 3), (x1, y1 - 2)])
    cv2.drawContours(frame, [pts2], 0, (255,255,255), 2)
    pts3 = np.array([(x2 + 2, y2), (x2 - 3, y2 - 5)])
    cv2.drawContours(frame, [pts3], 0, (255,255,255), 2)
    pts4 = np.array([(x2 + 2, y2), (x2 - 3, y2 + 5)])
    cv2.drawContours(frame, [pts4], 0, (255,255,255), 2)

    return frame

  def get_sizes(self, boxes):
    return [np.mean([boxes[id][i][2] - boxes[id][i][0] for i in range(len(boxes[id]))]) for id in boxes.keys()]

  def draw_size_bars(self, frame,xyx,bar_height_scaled,sizes):
    x1,y,x2 = xyx
    avg = np.mean(sizes)
    maxim = max(sizes)
    minim = min(sizes)

    # calculate min,max place
    mx1 = x1 + (x2 - x1) // 4
    mx2 = x2 - (x2 - x1) // 4
    my1 = int(y - bar_height_scaled * minim)
    my2 = int(y - bar_height_scaled * maxim)
    lx = x1 + (x2 - x1) // 2

    # draw rectangle
    cv2.rectangle(frame, (x1,y), (x2,int(y - bar_height_scaled * avg)), (200,200,200), -1)
    cv2.rectangle(frame, (x1,y), (x2,int(y - bar_height_scaled * avg)), (255,255,255), 1)

    # draw min,max
    cv2.rectangle(frame,(lx,my1),(lx,my2), (255,255,255), 1)
    cv2.rectangle(frame,(mx1,my1),(mx2,my1),(255,255,255), 1)
    cv2.rectangle(frame,(mx1,my2),(mx2,my2),(255,255,255), 1)

    return frame

  def rand(self, n):
    b = []
    for i in range(n):
      a = []
      a.append(np.random.randint(25))
      a.append(np.random.randint(10))
      a.append(np.random.randint(a[0],200))
      a.append(np.random.randint(a[1],100))
      b.append(a)
    return b



class FishDetector(FishDetectorFunctions):
  def __init__(self):
    super().__init__()

    self.HEIGHT = 720
    self.WIDTH = 1280
    self.BAR_SEPARATION_RATIO = 0.5
    self.bar_width_scaled = 10
    self.SIZE_BAR_SEP_RATIO = 0.5
    self.bar_height_scaled = 5

    self.id_stat = {}
    self.cls_stat = {}

  def detect_init(self, model_path, outpath, id_stat={}, cls_stat={}):

    self.out = cv2.VideoWriter(outpath,cv2.VideoWriter_fourcc(*'mp4v'),30,(self.WIDTH,self.HEIGHT))

    self.model = YOLO(model_path)

    self.id_stat = id_stat
    id_stat_init = {'class': -1, 'conf': -1, 'trajectory': -1, 'boxes': [], 'classes': [], 'confs': []}
    self.cls_stat = cls_stat
    cls_stat_init = {'num':(0,0), 'boxes':{}} # (left, right)
    self.names = []
    self.names = {0: 'owl bream', 1: 'asp', 2: 'common roach', 3: 'carp', 4: 'pike', 5: 'common bream', 6: 'chub', 7: 'prussian carp', 8: 'no class',
9: 'sabrefish', 10: 'catfish', 11: 'ide', 12: 'white bream', 13: 'blue bream', 14: 'nase', 15: 'zander', 16: 'coarse fish', 17: 'vimba', 18: 'bull-
head', 19: 'common rudd'}
    self.colors = np.random.choice(list(mcolors.CSS4_COLORS.values()), size=len(self.names), replace=False)
    self.colors = [tuple(np.array(mcolors.to_rgb(self.colors[i]))*255) for i in range(self.colors.shape[0])]

    self.start_save = 0

  def process_video(self, video, save_time=-1, save_path=''):
    if len(list(self.id_stat.keys())) > 0:
        start_id = list(self.id_stat.keys())[-1]
    else:
        start_id = 0
    self.frame_width = 0
    self.frame_height = 0

    # processing video
    for i, result in enumerate(self.model.track(source=video, stream=True)):
      video_frame = result.orig_img
      frame = np.zeros((self.HEIGHT,self.WIDTH,3)).astype(np.uint8)

      if self.frame_width == 0:
        self.frame_width = result.orig_img.shape[1]
        self.frame_height = result.orig_img.shape[0]

      # get results
      if len(self.names) == 0:
        self.names = result.names
        self.colors = np.random.choice(list(mcolors.CSS4_COLORS.values()), size=len(self.names), replace=False)
        self.colors = [tuple(np.array(mcolors.to_rgb(self.colors[i]))*255) for i in range(self.colors.shape[0])]

      detections = sv.Detections.from_ultralytics(result)

      if result.boxes.id is not None:
          detections.tracker_id = result.boxes.id.cpu().numpy().astype(int)

      # calculate within results
      for xyxy, _, confidence, class_id, tracker_id in detections:
        if tracker_id:
          # initializing
          tracker_id += start_id
          if tracker_id not in self.id_stat.keys():
            self.id_stat[tracker_id] = {'class': -1, 'conf': -1, 'boxes': [], 'classes': [], 'confs': []}

          # feeding data
          self.id_stat[tracker_id]['boxes'].append(xyxy)
          self.id_stat[tracker_id]['classes'].append(class_id)
          self.id_stat[tracker_id]['confs'].append(confidence)
          # calculating true class
          classes = self.id_stat[tracker_id]['classes']
          confs = self.id_stat[tracker_id]['confs']
          true_class = -1
          true_conf = -1
          cls_num = None
          conf_cls = {}
          max_conf_cls = {}
          cls_scores = {} # true_class is the maximum scored class, score = num_of_cls_appearance * avg_conf**2

          cls_num = dict(Counter(classes))
          if len(cls_num.keys()) == 1 and 8 in cls_num.keys():
            true_class = 8
            true_conf = max(confs)
          else:
            for j in range(len(classes)):
              if classes[j] != 8:
                if classes[j] not in conf_cls.keys():
                  conf_cls[classes[j]] = []
                conf_cls[classes[j]].append(confs[j])
            if 8 in cls_num.keys(): cls_num.pop(8)
            for cls in cls_num.keys():
              cls_scores[cls] = cls_num[cls] * mean(conf_cls[cls])**2
              max_conf_cls[cls] = max(conf_cls[cls])
            true_class = max(cls_scores, key=cls_scores.get)
            true_conf = max_conf_cls[true_class]

          # calculating trajectory
          first_box_center = self.calculate_box_center(self.id_stat[tracker_id]['boxes'][0])
          last_box_center = self.calculate_box_center(self.id_stat[tracker_id]['boxes'][-1])
          trajectory = (last_box_center[0] >= first_box_center[0]) * 1 # False -> 0 -> left; True -> 1 -> right

          # calculating self.cls_stat
          if self.id_stat[tracker_id]['class'] != true_class or self.id_stat[tracker_id]['trajectory'] != trajectory:
            boxes = []
            if self.id_stat[tracker_id]['class'] >= 0:
              self.cls_stat[self.id_stat[tracker_id]['class']]['num'] = self.add_to_tuple(self.cls_stat[self.id_stat[tracker_id]['class']]['num'],
self.id_stat[tracker_id]['trajectory'], -1)
              boxes = self.cls_stat[self.id_stat[tracker_id]['class']]['boxes'].pop(tracker_id, None)
            if true_class not in self.cls_stat.keys():
              self.cls_stat[true_class] = {'num':(0,0), 'boxes':{}}
            self.cls_stat[true_class]['num'] = self.add_to_tuple(self.cls_stat[true_class]['num'], trajectory, 1)
            if len(boxes) > 0:
              try:
                [self.cls_stat[true_class]['boxes'][tracker_id].append(box) for box in boxes]
              except:
                self.cls_stat[true_class]['boxes'][tracker_id] = boxes
            else:
              self.cls_stat[true_class]['boxes'][tracker_id] = [self.id_stat[tracker_id]['boxes'][-1]]

          # feeding data into self.id_stat
          self.id_stat[tracker_id]['class'] = true_class
          self.id_stat[tracker_id]['conf'] = true_conf
          self.id_stat[tracker_id]['trajectory'] = trajectory

          # draw on video_frame
          # draw box
          rgb = tuple(self.colors[self.id_stat[tracker_id]['class']])
          color = (int(rgb[0]), int(rgb[1]), int(rgb[2]))
          cv2.rectangle(video_frame, (int(xyxy[0]),int(xyxy[1])), (int(xyxy[2]),int(xyxy[3])), color, 3)

          # draw text
          nameID = self.id_stat[tracker_id]['class']
          name = self.names[nameID]
          conf = round(self.id_stat[tracker_id]['conf'], 2)
          text = '#' + str(tracker_id) + ' ' + name + ' - ' + str(conf)
          cv2.putText(video_frame, text, (int(xyxy[0]), int(xyxy[1] - 10)), cv2.FONT_HERSHEY_PLAIN, 2, (0,0,0), 5)
          cv2.putText(video_frame, text, (int(xyxy[0]), int(xyxy[1]) - 10), cv2.FONT_HERSHEY_PLAIN, 2, color, 2)

      # round video_frame
      video_frame = video_frame[int(self.frame_height*0.2):, :]
      vframe_height = len(video_frame)
      vframe_width = len(video_frame[0])
      video_frame = self.roundImg(video_frame, 0.1)

      # place video_frame on frame
      video_left = (self.WIDTH - vframe_width) // 2
      video_right = video_left + vframe_width
      frame[20:20+vframe_height, video_left:video_right] = video_frame

      # draw statistics
      bar_sep_height = int((vframe_height - 30) // (len(self.names) // 2))
      bar_height = int(bar_sep_height // (1 + self.BAR_SEPARATION_RATIO))
      sep_height = int(bar_height * self.BAR_SEPARATION_RATIO)
      middle_lines = (10 + (video_left - 10 - 10) // 2, self.WIDTH - 10 - (video_left - 10 - 10) // 2)

      y1 = 35 + sep_height // 2
      y2 = (35 + (len(self.names) // 2 - 1) * (bar_height + sep_height) + sep_height // 2) + bar_height + sep_height // 2
      cv2.rectangle(frame, (middle_lines[0], y1), (middle_lines[0], y2), (255,255,255), 1)
      y3 = (35 + (len(self.names) // 2 - 1) * (bar_height + sep_height) + sep_height // 2) + bar_height + sep_height // 2
      cv2.rectangle(frame, (middle_lines[1], y1), (middle_lines[1], y3), (255,255,255), 1)

      # adjust bar_width_scaled
      if len(self.cls_stat.keys()) > 0:
        nums = [self.cls_stat[cls]['num'] for cls in self.cls_stat.keys()]
        max_x = max(nums, key=lambda tup: tup[0])[0]
        max_y = max(nums, key=lambda tup: tup[1])[1]
        if max_x > max_y and int(middle_lines[0] - 10 - self.bar_width_scaled * max_x) < 10:
          self.bar_width_scaled = (middle_lines[0] - 10) / max_x
        elif max_x < max_y and int(middle_lines[0] + 10 + self.bar_width_scaled * max_y) > video_left - 10:
          self.bar_width_scaled = (video_left - 10 - middle_lines[0]) / max_y

      # place bars
      for j in range(len(self.names)):
        color = ((2, 10, 230), (250, 226, 7))
        if j < len(self.names) // 2:
          # place left bars
          x2 = middle_lines[0]
          if j in self.cls_stat.keys():
            x1 = x2 - int(self.bar_width_scaled * self.cls_stat[j]['num'][0])
          else:
            x1 = x2
          y1 = 35 + j * (bar_height + sep_height) + sep_height // 2
          y2 = y1 + bar_height + sep_height // 2
          cv2.rectangle(frame, (x1,y1), (x2,y2), color[0], -1)
          cv2.rectangle(frame, (x1,y1), (x2,y2), (255,255,255), 1)

          # place right bars
          x1 = middle_lines[0]
          if j in self.cls_stat.keys():
            x2 = x1 + int(self.bar_width_scaled * self.cls_stat[j]['num'][1])
          else:
            x2 = x1
          y1 = 35 + j * (bar_height + sep_height) + sep_height // 2
          y2 = y1 + bar_height + sep_height // 2
          cv2.rectangle(frame, (x1,y1), (x2,y2), color[1], -1)
          cv2.rectangle(frame, (x1,y1), (x2,y2), (255,255,255), 1)

          # place label
          if '_' in self.names[j]:
            label = ' '.join(self.names[j].split('_'))
          else:
            label = self.names[j]
          if j in self.cls_stat.keys():
            label += ': ' + str(self.cls_stat[j]['num'][0] + self.cls_stat[j]['num'][1])
          else:
            label += ': 0'
          y = y1 + (y2 - y1) // 2
          frame = self.placeStatLabel(frame, label, (middle_lines[0],y), bar_height)
        else:
          # place left bars
          x2 = middle_lines[1]
          if j in self.cls_stat.keys():
            x1 = x2 - int(self.bar_width_scaled * self.cls_stat[j]['num'][0])
          else:
            x1 = x2
          y1 = 35 + (j - len(self.names) // 2) * (bar_height + sep_height) + sep_height // 2
          y2 = y1 + bar_height + sep_height // 2
          cv2.rectangle(frame, (x1,y1), (x2,y2), color[0], -1)
          cv2.rectangle(frame, (x1,y1), (x2,y2), (255,255,255), 1)

          # place right bars
          x1 = middle_lines[1]
          if j in self.cls_stat.keys():
            x2 = x1 + int(self.bar_width_scaled * self.cls_stat[j]['num'][1])
          else:
            x2 = x1
          y1 = 35 + (j - len(self.names) // 2) * (bar_height + sep_height) + sep_height // 2
          y2 = y1 + bar_height + sep_height // 2
          cv2.rectangle(frame, (x1,y1), (x2,y2), color[1], -1)
          cv2.rectangle(frame, (x1,y1), (x2,y2), (255,255,255), 1)

          # place label
          if '_' in self.names[j]:
            label = ' '.join(self.names[j].split('_'))
          else:
            label = self.names[j]
          if j in self.cls_stat.keys():
            label += ': ' + str(self.cls_stat[j]['num'][0] + self.cls_stat[j]['num'][1])
          else:
            label += ': 0'
          y = y1 + (y2 - y1) // 2
          frame = self.placeStatLabel(frame, label, (middle_lines[1],y), bar_height)

      # place size graph
      frame = self.draw_graph(frame, (self.WIDTH - 400, self.HEIGHT - 20 - vframe_height - 30), 20+vframe_height)
      bar_sep_width = (self.WIDTH - 410) // ((1 + self.SIZE_BAR_SEP_RATIO) * len(self.names))
      sep_width = bar_sep_width * self.SIZE_BAR_SEP_RATIO
      bar_width = bar_sep_width - sep_width
      all_sizes = []
      for cls in self.cls_stat.keys():
        sizes = self.get_sizes(self.cls_stat[cls]['boxes'])
        all_sizes.append(sizes)
        if len(sizes) > 0:
          if max(sizes) * self.bar_height_scaled > self.HEIGHT - 20 - vframe_height - 30:
            self.bar_height_scaled = (self.HEIGHT - 20 - vframe_height - 30) / max(sizes)

      for j, cls in enumerate(self.cls_stat.keys()):
        if len(all_sizes[j]) > 0:
          x1 = int(205 + cls * (bar_sep_width) + sep_width // 2)
          x2 = int(x1 + bar_width)
          y = int(self.HEIGHT - 20)
          self.draw_size_bars(frame,(x1,y,x2),self.bar_height_scaled,all_sizes[j])

      frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
      self.out.write(frame)

      if save_time >= 0 and self.start_save % save_time == 0:
        self.save_results(save_path)
      self.start_save += 1

      if i == 9465:
        print(len(self.names))
        print(self.names)
        print(len(self.id_stat.keys()))

  def save_results(self, save_path=''):
    # save dictionaries for statistics
    fcls = open('/kaggle/working/cls_stat.pkl','wb')
    fid = open('/kaggle/working/id_stat.pkl','wb')
    pickle.dump(self.cls_stat,fcls)
    pickle.dump(self.id_stat,fid)
    fcls.close()
    fid.close()

  def detect_finish(self):
    self.out.release()

  def statistics(self, stat_path='', show=False):
    # Statistics

    if os.path.getsize(stat_path + 'id_stat.pkl') > 0:
      self.id_stat = pickle.load(open(stat_path + 'id_stat.pkl','rb'))
      self.cls_stat = pickle.load(open(stat_path + 'cls_stat.pkl','rb'))


names = {0: 'owl bream', 1: 'asp', 2: 'common roach', 3: 'carp', 4: 'pike', 5: 'common bream', 6: 'chub', 7: 'prussian carp', 8: 'no class', 9: 'sabre-
fish', 10: 'catfish', 11: 'ide', 12: 'white bream', 13: 'blue bream', 14: 'nase', 15: 'zander', 16: 'coarse fish', 17: 'vimba', 18: 'bullhead', 19:
'common rudd'}
colors = np.random.choice(list(mcolors.CSS4_COLORS.values()), size=len(names), replace=False)
colors = [tuple(np.array(mcolors.to_rgb(colors[i]))*255) for i in range(colors.shape[0])]

all_sizes = []
size_names = []
nums = {'Upstream':[],'Downstream':[]}

# get num and size
for cls in names.keys():
  if cls in cls_stat.keys():
    if len(cls_stat[cls]['boxes']) > 0:
      all_sizes.append(get_sizes(cls_stat[cls]['boxes']))
      size_names.append(names[cls])
    nums['Upstream'].append(cls_stat[cls]['num'][0])
    nums['Downstream'].append(cls_stat[cls]['num'][1])
  else:
    nums['Upstream'].append(0)
    nums['Downstream'].append(0)

# plot
fig = plt.figure(constrained_layout=True, figsize=(10,10))
ax = fig.add_gridspec(2, 2)
ax1 = fig.add_subplot(ax[0, 0])
ax1.set_title('Sizes')
ax2 = fig.add_subplot(ax[0, 1])
ax2.set_title('Quantities')
ax3 = fig.add_subplot(ax[1, 1])
ax3.set_title('Trajectories')
ax4 = fig.add_subplot(ax[1, 0])
ax4.set_title('Trajectories most common')

asizes = [[round(size * 96 / 580, 1) for size in sizes] for sizes in all_sizes]
ax1.boxplot(asizes, vert=True)
ax1.set_xticklabels(size_names, rotation=30,ha='right')
ax1.yaxis.set_major_formatter(FormatStrFormatter('%d cm'))

pos = np.arange(len(all_sizes)) + 1
medians = [np.mean(size) for size in all_sizes]
upper_labels = [str(round(s * 96 / 580, 1)) + 'cm' for s in medians]
weights = ['bold', 'semibold']
box_colors = ['darkkhaki', 'royalblue']
yplace = [0.97, 0.93]
for tick, label in zip(range(len(all_sizes)), ax1.get_xticklabels()):
  k = tick % 2
  ax1.text(pos[tick], yplace[k], upper_labels[tick],
            transform=ax1.get_xaxis_transform(),
            horizontalalignment='center', size='x-small',
            weight=weights[k], color='royalblue')

ax = ax2
x = np.arange(len(names.values()))  # the label locations
width = 0.25  # the width of the bars
multiplier = 0
for attribute, measurement in nums.items():
    offset = width * multiplier
    rects = ax.bar(x + offset, measurement, width, label=attribute)
    multiplier += 1

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_xticks(x + width, names.values(), rotation=45, ha='right')
ax.legend(loc='upper left', ncols=2)
ax.set_yscale('log')
ax.yaxis.set_major_formatter(FormatStrFormatter('%d db'))

# plot trajectories
for id in id_stat.keys():
  boxes = id_stat[id]['boxes']
  if id_stat[id]['class'] > -2:
    trajx = [boxes[i][0] + (boxes[i][2] - boxes[i][0]) // 2 for i in range(len(boxes))]
    trajy = [580 - boxes[i][1] - (boxes[i][3] - boxes[i][1]) // 2 for i in range(len(boxes))]
    ax3.plot(trajx,trajy,color='b')
#ax3.set_xlim(0,752)
ax3.set_ylim(0,580)
ax3.set_aspect(1)
ax3.set_xticks([])
ax3.set_yticks([])

colors = ['b','g','r']
dictionary = {cls:cls_stat[cls]['num'][0] + cls_stat[cls]['num'][0] for cls in cls_stat.keys()}
k = Counter(dictionary)
high = k.most_common(3)
high = [tup[0] for tup in high]
namesax4 = [names[cls] for cls in high]

# plot trajectories
for id in id_stat.keys():
  boxes = id_stat[id]['boxes']
  if id_stat[id]['class'] in high:
    trajx = [boxes[i][0] + (boxes[i][2] - boxes[i][0]) // 2 for i in range(len(boxes))]
    trajy = [580 - boxes[i][1] - (boxes[i][3] - boxes[i][1]) // 2 for i in range(len(boxes))]
    ax4.plot(trajx,trajy,color=colors[high.index(id_stat[id]['class'])])
#ax4.set_xlim(0,752)
#ax4.set_ylim(0,580)
ax4.set_aspect(1)
ax4.set_xticks([])
ax4.set_yticks([])
patches = []
for i in range(len(colors)):
  patches.append(mpatches.Patch(color=colors[i], label=namesax4[i]))
ax4.legend(handles=patches)

plt.savefig("statistics.png")
    if show:
      plt.show()
```