Alex LO

**Report : Object Detection - Alert owner when their goldfish is dying**

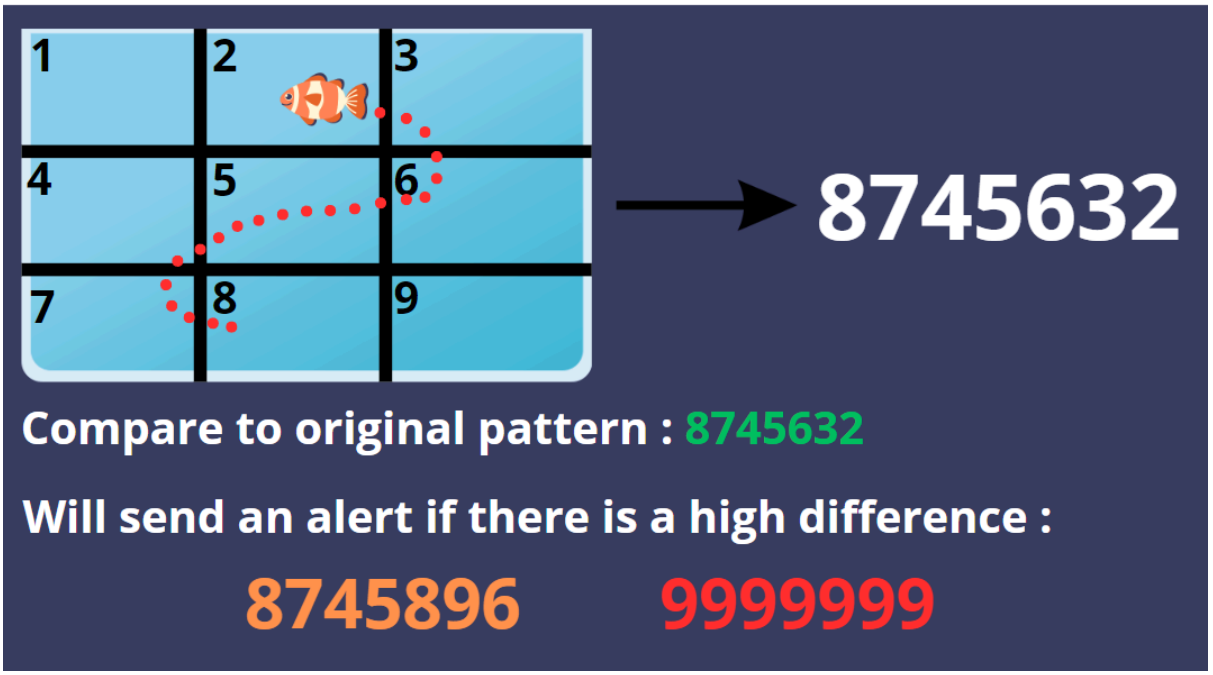
Fishkeeping is an attractive passion that transports enthusiasts to the heart of an aquatic world full of diversity, and constitutes a growing global market. There are thousands of freshwater and marine species, some of which enjoy exceptional popularity due to their bright coloration, unique morphological features or fascinating behaviors.

However, behind the splendor of aquariums lies a persistent and worrying problem: the high frequency of premature deaths among aquarium fish. More than twenty million households around the world are home to these miniature ecosystems, witnesses of the growing popularity of this activity. Unfortunately, this passion has sometimes turned into a common consumption, where the simple substitution of a dead fish becomes commonplace, without careful consideration of the underlying causes of this early mortality.

In order to address this crucial issue, our project focuses on the following essential question: How to identify the signs of premature death of aquarium fish?

By looking at the behavior of a fish in an aquarium, we can identify a swimming pattern for each fish. This pattern will be the key to our detection of fish health. Diseases that usually affect fish in aquariums will tend to make him swim on his side, keep his fins tight, float, sink, hide, stay close to the surface or at the bottom of the aquarium. All these elements will change its original pattern and this is precisely what will allow us to alert the owner as quickly as possible.

The main objective of the project is to create an AI capable of detecting the movements of a fish in real time and comparing its swimming pattern with the one saved. If the AI observes a high difference between the patterns it will send an alert to the owner.



However, my role in this project is only to create an AI capable of detecting the movements of a fish. To do this, you must first choose which type of AI with object detection to use.

Object detection is a computer vision task that involves identifying and locating objects in images or videos. It can be divided into two main categories: single-shot detectors and two-stage detectors.

Single-shot object detection uses a single pass of the input image to make predictions about the presence and location of objects in the image. It processes an entire image in a single pass, making them computationally efficient. Such algorithms can be used to detect objects in real time in resource-constrained environments.

Two-shot object detection uses two passes of the input image to make predictions about the presence and location of objects. The first pass is used to generate a set of proposals or potential object locations, and the second pass is used to refine these proposals and make final predictions. This approach is more accurate than single-shot object detection but is also more computationally expensive.

Generally, single-shot object detection is better suited for real-time applications, while two-shot object detection is better for applications where accuracy is more important.

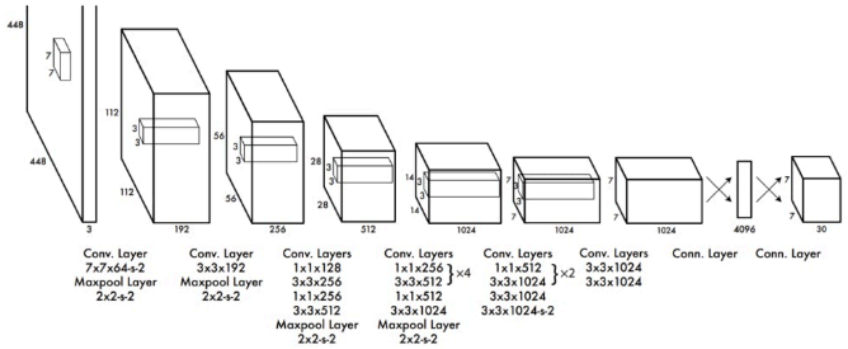
In this case, I choose the single-shot detector YOLO for the project. You Only Look Once (YOLO) proposes using an end-to-end neural network that makes predictions of bounding boxes and class probabilities all at once. It differs from the approach taken by previous object detection algorithms, which repurposed classifiers to perform detection.

Following a fundamentally different approach to object detection, YOLO achieved state-of-the-art results, beating other real-time object detection algorithms by a large margin.

While algorithms like Faster RCNN work by detecting possible regions of interest using the Region Proposal Network and then performing recognition on those regions separately, YOLO performs all of its predictions with the help of a single fully connected layer.

Methods that use Region Proposal Networks perform multiple iterations for the same image, while YOLO gets away with a single iteration.

The YOLO algorithm takes an image as input and then uses a simple deep convolutional neural network to detect objects in the image. The architecture of the CNN model that forms the backbone of YOLO is shown below.



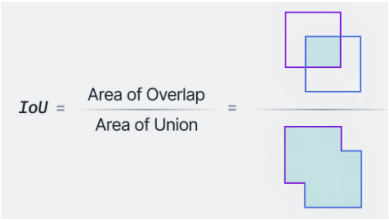

The first 20 convolution layers of the model are pre-trained using ImageNet by plugging in a temporary average pooling and fully connected layer. Then, this pre-trained model is converted to perform detection since previous research showcased that adding convolution and connected layers to a pre-trained network improves performance. YOLO's final fully connected layer predicts both class probabilities and bounding box coordinates.

YOLO divides an input image into an S × S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and how accurate it thinks the predicted box is.

YOLO predicts multiple bounding boxes per grid cell. At training time, we only want one bounding box predictor to be responsible for each object. YOLO assigns one predictor to be "responsible" for predicting an object based on which prediction has the highest current IOU with the ground truth.

**Intersection over Union (IoU)**

Intersection over Union is a popular metric to measure localization accuracy and calculate localization errors in object detection models.



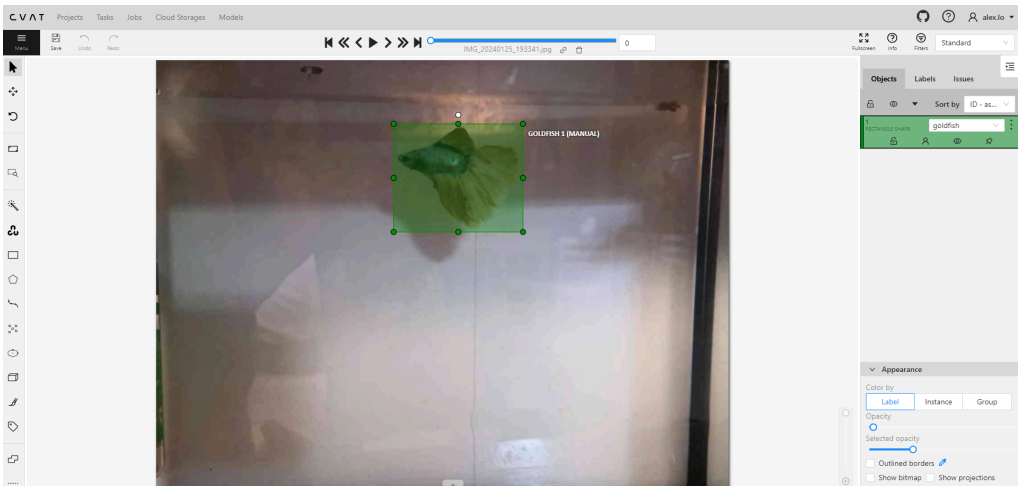

One key technique used in the YOLO models is non-maximum suppression (NMS). NMS is a post-processing step that is used to improve the accuracy and efficiency of object detection. NMS is used to identify and remove redundant or incorrect bounding boxes and to output a single bounding box for each object in the image.

Now that we chose the model, we need a relevant dataset. To be as accurate as possible, we will use a dataset based on real images of my goldfish in my aquarium. I used 150 images of my goldfish from multiple angles :

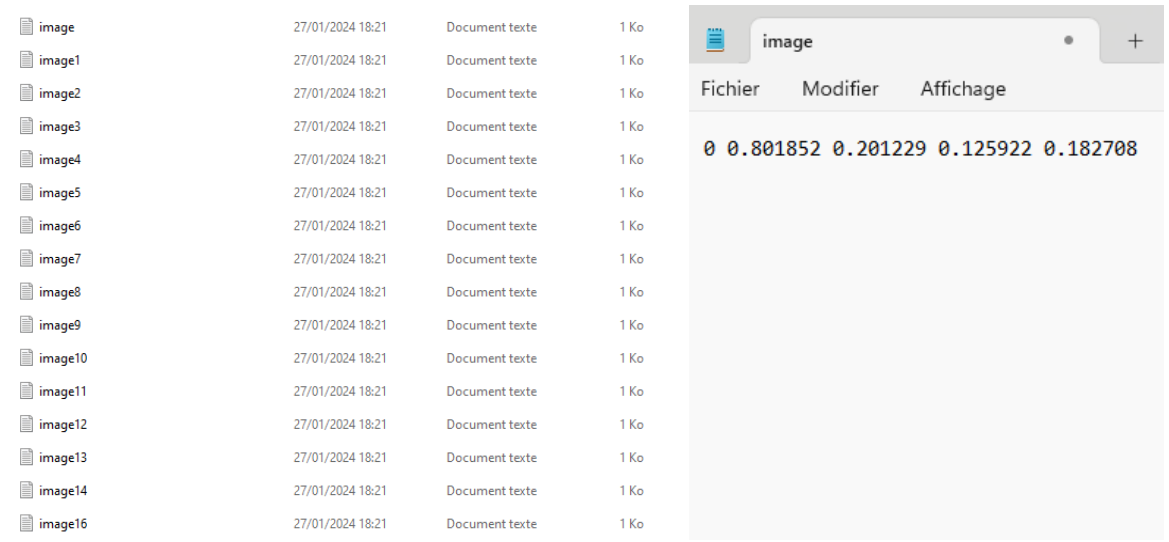

After taking the photos, we have to annotate the dataset. Annotating images is a crucial step in the data preparation process. It involves labeling objects of interest in the images, specifying their locations and classes.

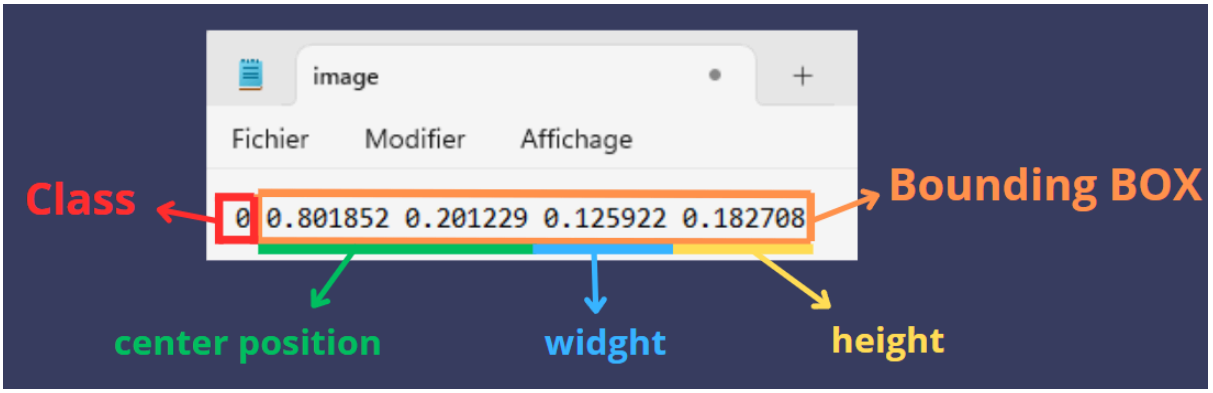
YOLO is a supervised learning algorithm, which means it needs labeled training data to learn patterns and relationships between input images and corresponding object locations. Annotations serve as ground truth labels for training the YOLO model.

To annotate the dataset, I choose CVAT :

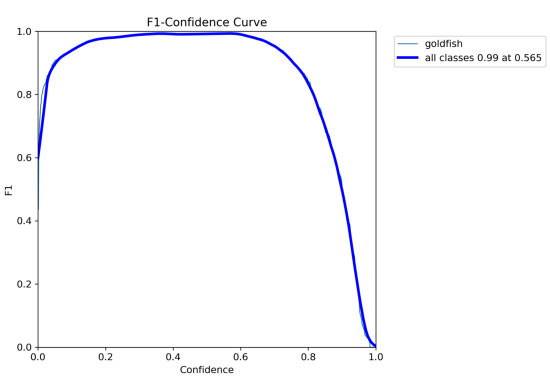


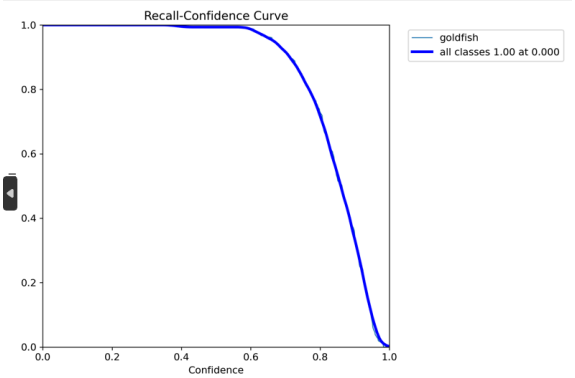
After annotating the dataset, we can export annotations with the YOLO format. we end up with a directory composed with multiples .txt files with image's matching names :

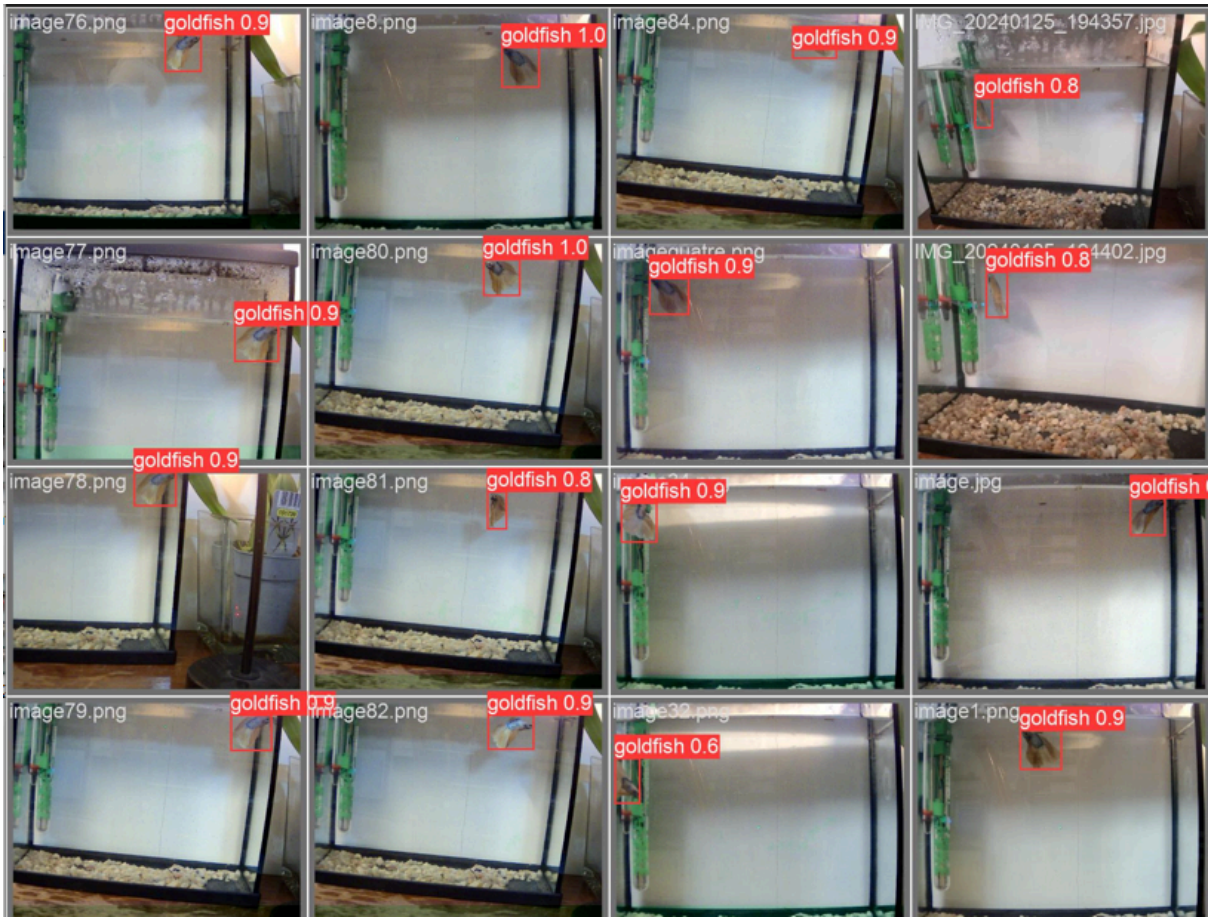



We built a model from scratch with YOLOv8. To perform a deeper train I choose to train the model with 1 epoch( (file train), 10 epochs (file train3) and 100 epochs(file train2). The training returns a lot of information in the file **"runs>detect>train2"** :






The model shows a good performance on the dataset that we give it to for training. Let's try on data that has never been seen. To perform this test, I use a 30sec video and the model trained on 100 epochs.

After running **predict.py** we got the **video1.mp4_out** (the video is in the "**videos**" directory) which shows how the model reacts on unseen data.

The result is positive, we can see that it performs well when the goldfish is swimming face to the camera. However, it cannot find it when the goldfish is turning around or going up.