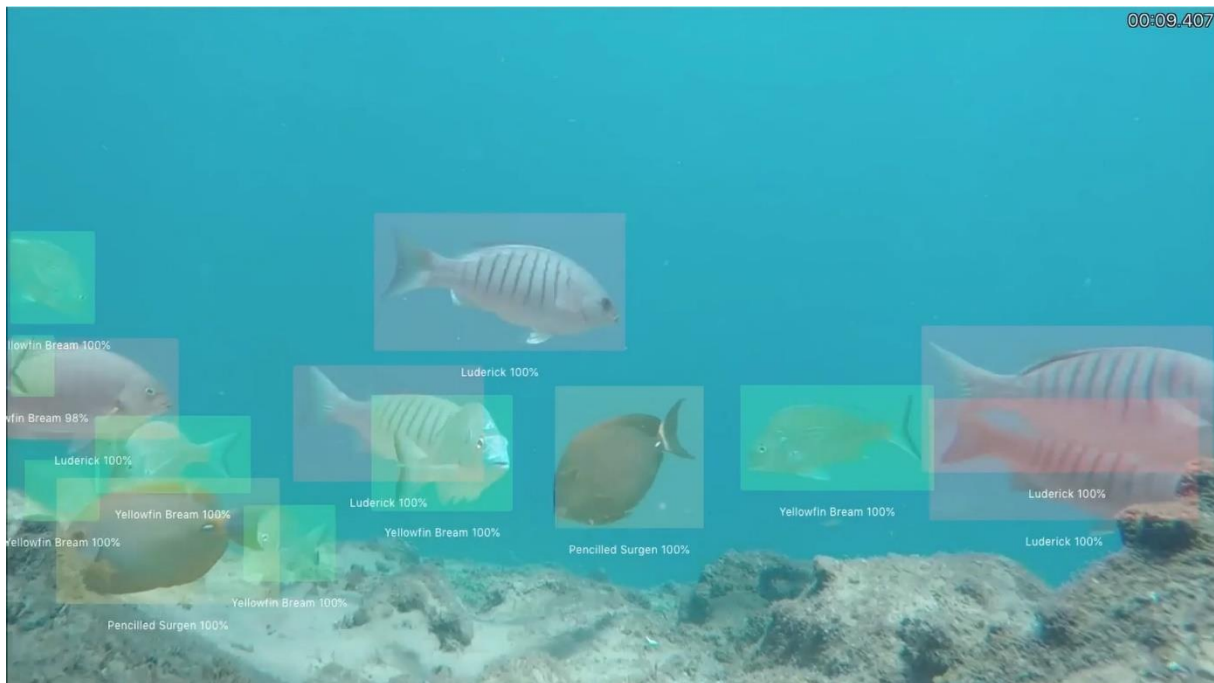


TER 2021-071 – Development

Title:

Deep Learning System for animal recognition in underwater images and videos from Mediterranean Sea



Students:

Giuseppe Alessio Dimonte	Master 2 EIT Digital (DSC)
Arturo Adan Pinar	Master 2 EIT Digital (DSC)
Yueqiao Huang	MAM5

Supervisors:

Frederic Precioso
Diane Lingrand

Date: 14/02/2022

Index of contents

1. PRESENTATION OF THE SUBJECT	2
1.1 PROBLEM	2
1.2 CONTEXT AND USERS	2
1.3 SCOPE QUALIFICATION.....	3
2. PRESENTATIONS OF THE SOLUTIONS	4
2.1. OBJECT DETECTION	4
2.1.1. YOLO.....	4
2.2. INSTANCE SEGMENTATION	5
2.2.1. Mask R-CNN	7
2.2.2. Cascade	7
2.2.3. HTC (Hybrid Task Cascade).....	7
2.2.4. PANet	8
2.2.5. Mask Scoring R-CNN.....	8
2.2.6. MPN (Multi-Path Network)	9
2.2.7. YOLACT	10
3. POSITIONING OF THE SOLUTION IN RELATION TO THE EXISTING ONES	11
3.1. MASK SCORING R-CNN VS MASK R-CNN.....	11
3.2. PANET VS MASK R-CNN	12
4. WORK DONE	13
4.1 IDENTIFYING THE PROBLEM AND THE PURPOSE	13
4.1 DATASET ACQUISITION	14
4.3 MEGADETECTOR ADAPTATION	14
4.4 USER INTERFACE FOR ANNOTATION.....	15
4.5 MODEL RESEARCH	17
4.5.1 Mask R-CNN on a custom dataset.....	17
4.5.2 Object tracking with OpenCV	18
4.5.3 Object tracking with DeepSORT	19
4.6 PROJECT REFINEMENT	19
4.7 USER INTERFACE MODIFICATION.....	20
4.7.1 "Start training" button	20
4.7.2 Flask	20
5. CONCLUSIONS	22
6. REFERENCES.....	24
7. ANNEX	26
7.1 USE CASE PRESENTED IN THE DOW	26
7.2 USER MANUAL.....	27

1. Presentation of the subject

1.1 Problem

One of the greatest humans' abilities is the detection and classification of objects in visual scenarios (real life, images, and videos) with a natural accuracy, simplicity, and velocity. Humans preserve this ability also for underwater ecosystems, being able to easily discriminate the position of fishes, corals, and marine-related objects. Beyond that, humans with a certain expertise in the marine field are capable to distinguish different species, classifying each object in the correct way.

What is required in the development of the project is to automatically detect and recognize fishes in the marine ecosystem without passing through experts each time it is necessary to perform this kind of task. In fact, in the case the process needs to be done at a high scale and considering many species, it might become a process requiring a notable amount of time. Fish experts should spend a substantial amount of their time drawing the shape and assigning the label for each fish. On a practical side, it is clearly an ineffective use of time, energy, and money. Considering that on average 2 minutes are required for annotating a single image by an expert, that means that the annotation of 1000 images would require more than 33 hours. This amount of time turns out to be a waste of the costly experts' working hours.

To tackle this problem a machine learning solution can be deployed to accomplish the same duty in an automatic way, saving in this way time for experts who can dedicate themselves to other activities.

1.2 Context and users

The users involved are the experts from the Computer Science, Signals and Systems laboratory of Sophia Antipolis (I3S) and the Ecology and Conservation Science for Sustainable Seas (ECOSEAS), whose research interest is biology and who are focusing on the monitoring and safeguarding of the Mediterranean area, with the aim of protecting the biodiversity of the sea [1][2].

The main outcome of the project is to support the biologists with an efficient tool they can exploit for fish location and class extraction, with the ability of potentially perform large-scale operations on many images.

This is the reason why a tool for the analysis and monitoring of the Mediterranean Sea area is considered as a solid starting point for the above-mentioned players.

1.3 Scope qualification

Considering the dataset dimension, the project aims to use images representing the underwater ecosystem within the Mediterranean Sea by using the SeaCLEF dataset [3]. However, due to its unavailability, the DeepFish dataset was used instead. This dataset contains approximately 40 thousand images of fishes, and because of that it enables the recognition of fishes from different scenarios, allowing its use in more general contexts [4].

For the technological aspect, the software created can identify, locate, and create a mask for fishes in different images classifying them in a unique “Fish” class. Regarding the accuracy of the model, it relies on the number of images annotated in the dataset, which can be improved by manual annotation from the experts through the User Interface.

2. Presentations of the solutions

The first request for the fish model is to infer the position of each fish in each image. This task is harder in the case of underwater images, where fishes are difficult to distinguish due to the blurriness of the images and the numerous overlapping and occlusions between different fishes.

According to the requirements of the project, for every picture it was required to perform instance segmentation and object detection to locate each fish in the image and segment at the pixel level their shape.

2.1. Object detection

The main goal of object detection is to allow an IT algorithm to separate all the instances of different objects in an image or video in different classes. The algorithm can locate the instances of each class and mark them using a square rounding them called bounding box [5].

2.1.1. YOLO

An example of an object detection model is YOLO, where a simple convolutional network predicts multiple bounding boxes and class probabilities at once. The object detection is framed as a regression problem to spatially separated bounding boxes and associated class probabilities [6].

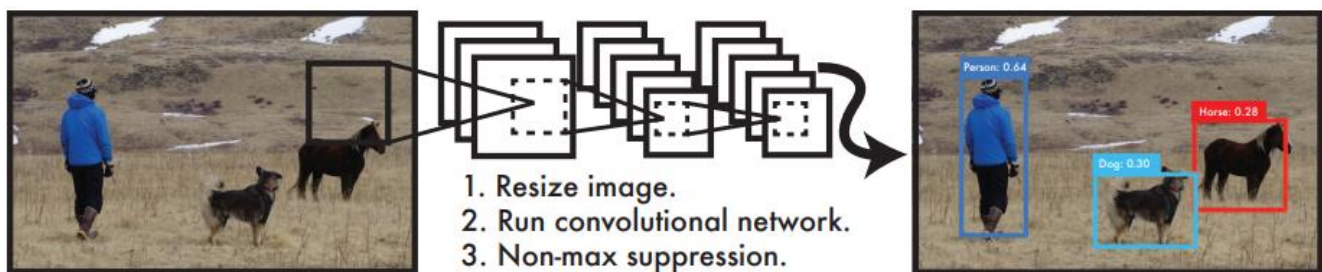


Figure 1: YOLO detection system, from [6]

However, despite the speed of the predictions, in the case of objects one close to the other or in case of occlusions, these methods do not have enough precision and the instance segmentation approach is preferred [6].

2.2. Instance segmentation

Instance segmentation predicts the class label and the pixel-specific instance mask allowing to obtain higher accuracy in close objects compared to object detection. It comprehends different approaches and according to the literature different approaches exist [7].

a. Classification of mask proposals

This technique involves the generation of mask proposals which are classified in the corresponding class. “Selective search” is a first approach following this technique: it detects bounding boxes and then performs semantic segmentation [8]. However, this family of techniques was replaced after Deep Learning became more popular with solutions like “R-CNN”, “Fast R-CNN” and “Faster R-CNN”.

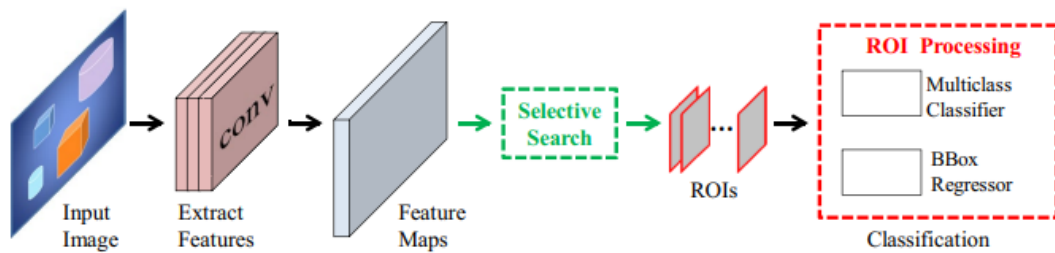


Figure 2: Framework for Classification of Mask Proposals technique, from [7]

b. Detection followed by segmentation

This approach involves object detection using a bounding box followed by the segmentation of the object within the bounding box. Some examples of this approach are “Mask R-CNN” and “Faster R-CNN”.

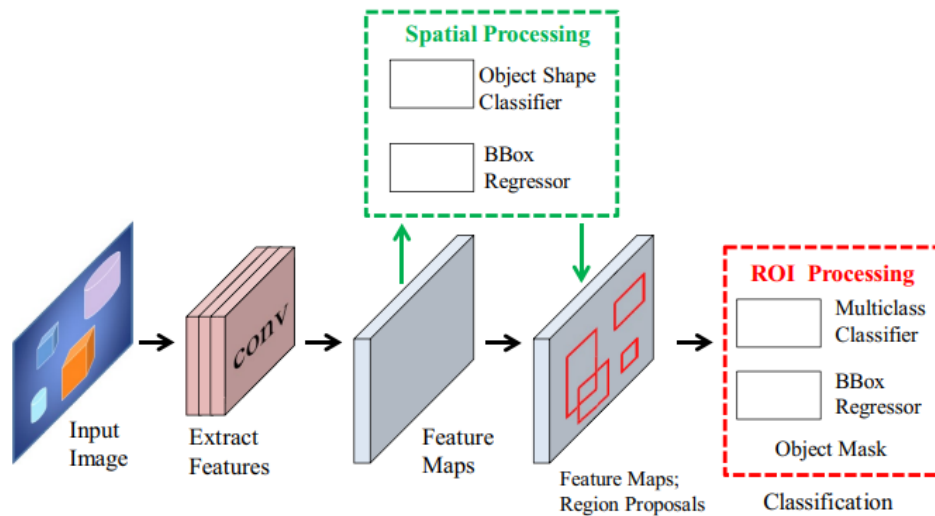


Figure 3: Framework of Detection followed by segmentation approach, from [7].

c. Labelling pixels followed by clustering

It performs categorical labelling of every image pixel, followed by a grouped process of the pixels into object instances using a clustering algorithm. “Deep Watershed Transform” and “Instance Cut” are approaches following this methodology.

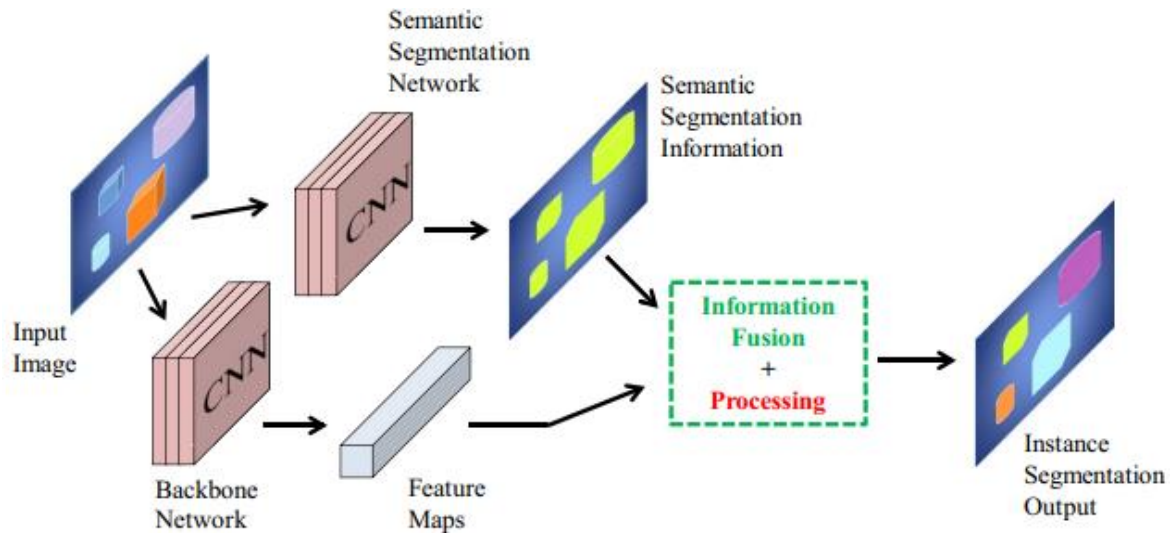


Figure 4: Framework for labelling pixels followed by clustering approach, from [7]

d. Dense Sliding Window Methods

These methods use CNNs for generating the mask proposals by using dense sliding window techniques. A sliding window is a rectangular region with fixed width and height that moves across an image for detecting the exact position of each object. “TensorMask”, “DeepMask” and “InstanceFCN” are examples of algorithms using this approach.

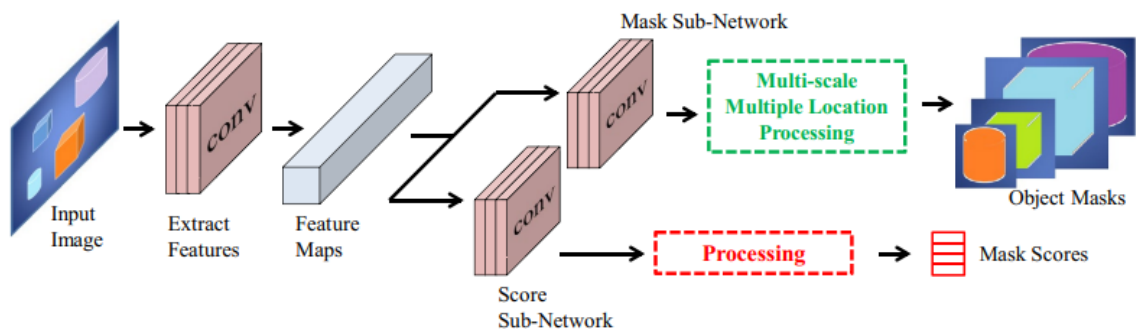


Figure 5: Framework for dense sliding window approach, from [7]

2.2.1. Mask R-CNN

Mask R-CNN introduces an improvement to overcome Faster R-CNN, by adding a new branch to the existing one for object detection in Faster R-CNN, to perform object mask prediction in parallel [9]. This translates into a new model, offering multiple advantages. Mask R-CNN outputs for each candidate object, a class label and a bounding box offset (Faster R-CNN output) combined with the object mask segmented at the pixel level.

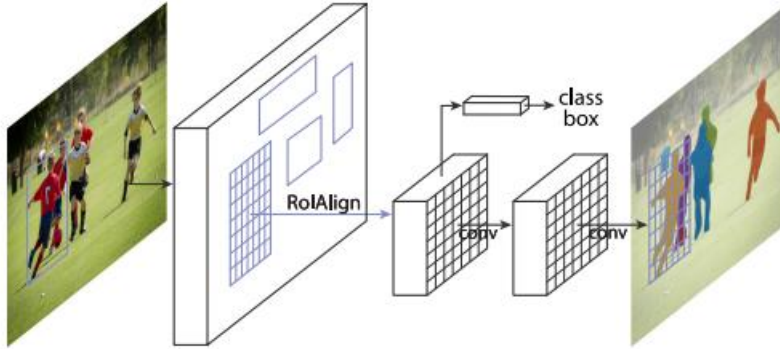


Figure 6: Mask R-CNN framework for instance segmentation, from [9]

2.2.2. Cascade

Cascade is an architecture in object detection, with improved performances in its tasks thanks to multi-stage architecture for object detection. Nevertheless, when this approach is shifted to instance segmentation, it brings to a limited gain in the accuracy of the mask compared with the bounding box [10].

2.2.3. HTC (Hybrid Task Cascade)

HTC is a framework for instance segmentation improving the cascade methods [10]. HTC was developed with the goal of overcoming the limited gain accuracy problem, improving the information flow by using a multitasking approach at each stage where the bounding box and the mask prediction are combined together and by introducing a fully convolution branch helping the distinction between hard foreground and cluttered foreground.

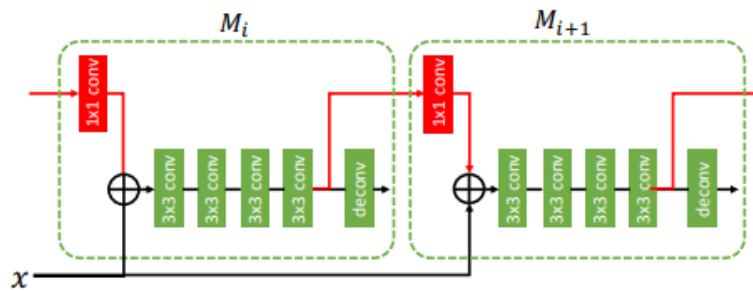


Figure 7: Architecture of multi-stage mask branches of Hybrid Task Cascade, from [10]

2.2.4. PANet

The PANet model focuses on the way the information propagates in deep neural networks. It implements shorter information paths across the lower layers and the features at the top of the network, making more relevant information flow across the subnetworks generating the proposals. Furthermore, in the end of the network a branched segment captures some proposals view to improve the prediction of the generated masks [11].

As shown in Figure 8, the architecture is based on the following steps:

- FPN backbone:** path augmentation and aggregation are performed to improve performances.
- Bottom-up path augmentation:** makes low-layer information easier to propagate.
- Adaptive feature pooling:** allows each proposal to access information from all levels for prediction.
- Box branch:** complementary path added to the mask-prediction branch.
- Fully connected fusion:** made by a Fully Connected Network to extract the final mask.

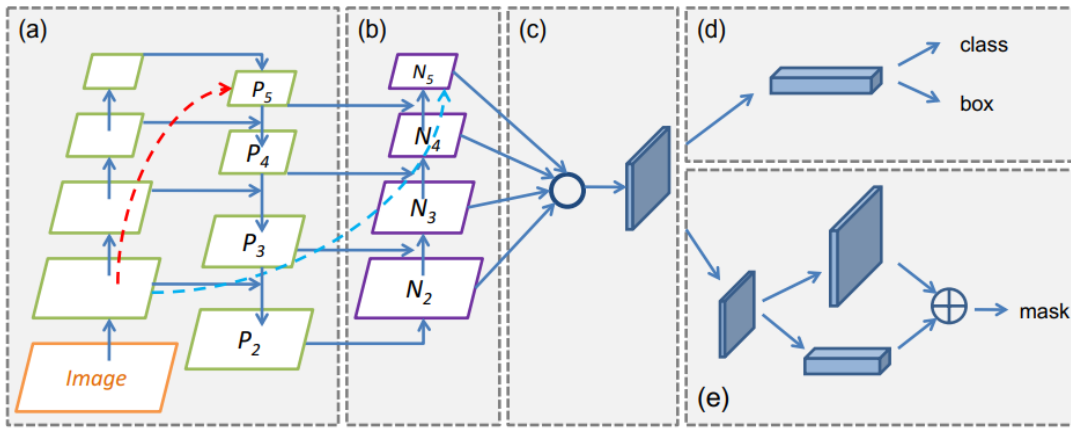


Figure 8: Architecture of PANet framework, from [11]

2.2.5. Mask Scoring R-CNN

Mask Scoring R-CNN was created to solve the problem presented by Mask R-CNN, which presents some loss of generality and in some situations, it was not performing optimally. The idea is to combine Mask R-CNN with an additional Mask-IoU module, learning the Mask-IoU aligned mask score because the quality of the mask with IoU gives very accurate results [12].

The arrangement created in MaskScoringCNN is used to predict the IoU between the input mask and the ground truth mask. The model contains a network block learning the predicted mask quality combining the features of the instance with the predicted mask to regress the predicted mask IoU. Therefore, by analyzing and comparing each time the quality and score of the mask, the performance of the instance segmentation process is improved.

The architecture of the model is shown in Figure 9, where the input image is fed into a backbone network to generate Region of Interest (RoI) via Region Proposal Network (RPN) and RoI features via RoIAlign. Using the predicted mask and RoI features as inputs, the MaskIoU is predicted with 4 convolutional layers and 3 fully connected layers.

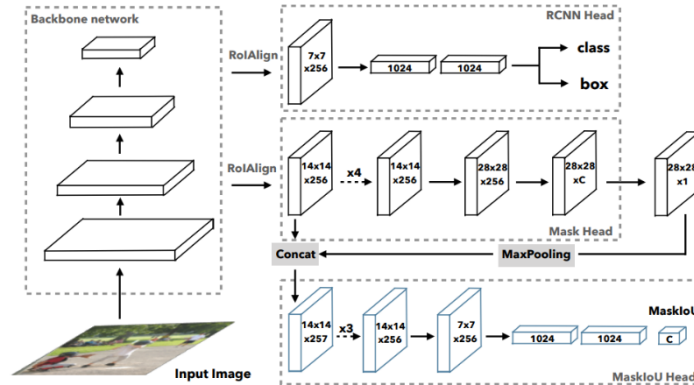


Figure 9: Architecture of Mask Scoring R-CNN, from [15]

2.2.6. MPN (Multi-Path Network)

MPN is the result of three modifications performed in the Fast R-CNN model. First, by skipping certain connections, it provides the object detector the capability of accessing to features of different layers, giving to the network a fast-learning process. Moreover, the introduction of an element exploiting the context of the objects allows to learn more specific features. Finally, the integral loss function together with the appropriate network adjustment improves the localization [13].

As a result, in the network, the information can flow following multiple paths as features, multiple layers and from multiple object views. This improvements of the Fast R-CNN model yield in a better model in the COCO 2015 detection and segmentation challenges.

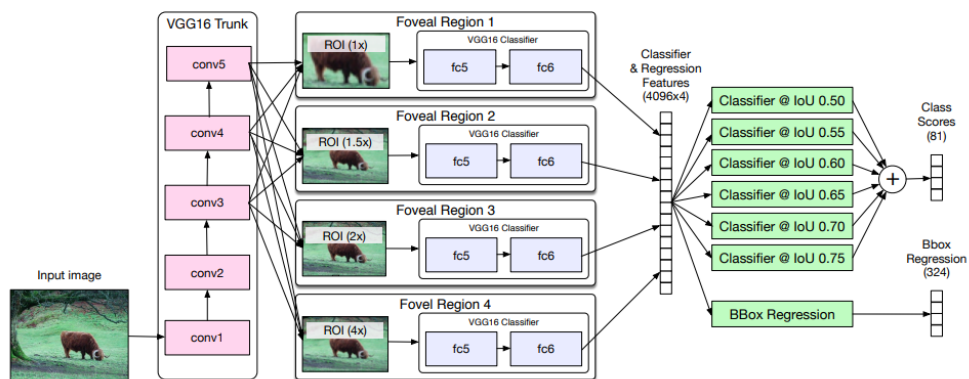


Figure 10: Architecture of MultiPath network, from [12]

2.2.7. YOLACT

YOLACT is an instance segmentation model focused on instance segmentation in real-time. It is the fastest real-time instance segmentation technique thanks to the way it is implemented. To achieve the results obtained it divides the image segmentation in two parallel sub-tasks:

- 1) Generation of the prototype masks
- 2) Prediction of the mask coefficients for each mask

Then, the instance masks are produced by linear combination of the prototype masks by using the coefficients of the masks. With this implementation, the network can learn localization [14].

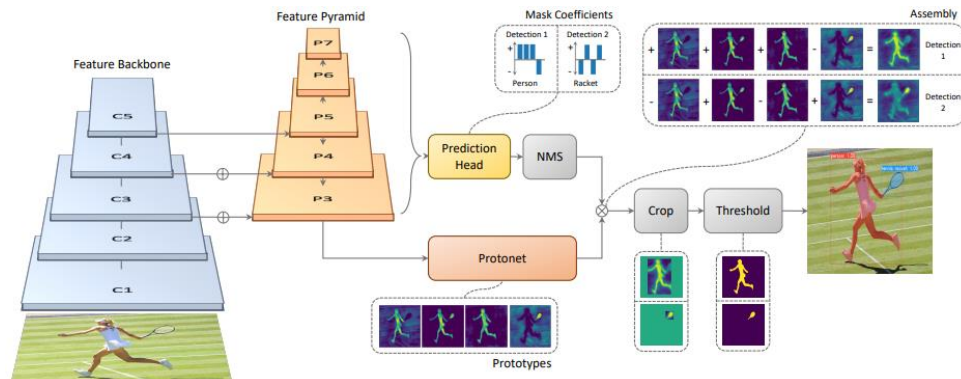


Figure 11: YOLACT Architecture, from [19]

3. Positioning of the solution in relation to the existing ones

According to the literature and from the results shown in Figure 12 extracted from instance segmentation on the Microsoft COCO dataset, there are many models outperforming Mask R-CNN in terms of Average Precision (TensorMask, MaskLab+, CenterMask, Mask Scoring R-CNN, Non-local Neural Networks, SOLO, BlendMask, GCNet, SOLOv2, PANet and HTC) [7][15].

Method	Average Precision (AP)	Year
Hybrid Task Cascade [81] (With Extra Training Data)	43.9%	2019
PANet [79]	42.0%	2018
SOLOv2 [134]	41.7%	2020
GCNet [130]	41.5%	2019
BlendMask [135]	41.3%	2020
SOLO [136]	40.4%	2019
Non-local Neural Networks [113]	40.3%	2017
Mask Scoring R-CNN [133]	39.6%	2019
CenterMask [137]	38.3%	2019
MaskLab+ [22]	38.1%	2017
TensorMask [91]	37.3%	2019
Mask R-CNN [67]	37.1%	2017
PolarMask [138]	32.9%	2019
YOLACT [132]	29.8%	2019
MultiPath Network [72]	25.0%	2016

Figure 12: Instance segmentation on the Microsoft COCO dataset, from [7]

It is important to remark that the main task to be performed in the project is “detection followed by segmentation” approach and only the models “Mask Scoring R-CNN”, “PANet”, and “Mask R-CNN” belong to this category.

3.1. Mask Scoring R-CNN vs Mask R-CNN

Mask Scoring R-CNN is a model based on the awareness of the quality of its own predictions by regressing the mask Intersection over Union (IoU) and prioritizing the most accurate predictions during the evaluation phase. However, the main downside presented by this model is the training time since the two tasks performed by Mask Scoring R-CNN are hard to train with a single objective function [12].

Moreover, the training presents a problem that Mask Scoring R-CNN model does not solve completely: the mask score learning task is decomposed as mask classification and MaskIoU regression and therefore, the problem arises when the Mask IoU needs to deal with Regions of Interest (RoI) that may contain multiple categories of objects. There are several solutions:

- 1) Learning the target category, while the others in the proposal are ignored.
- 2) Learning the MaskIoU for all the categories.
- 3) Learning the MaskIoU of all the categories appearing in the RoI [12].

3.2. PANet vs Mask R-CNN

PANet focuses on improving the way the information propagates in the Neural Network to beat its competitors, especially Mask R-CNN, which has important points of improvement, namely:

- 1) There is a long path from low-level structure to topmost features, increasing the difficulty to access accurate localization information.
- 2) Each proposal is predicted based on feature grids pooled from one feature level, assigned heuristically so it can be updated since information discarded in other levels may be helpful for final prediction.
- 3) The mask prediction is made on a single view, so it loses the possibility of gathering more diverse information [11].

With PANet is instead possible to:

- 1) Shorten information path by introducing a bottom-up path augmentation propagating low-level features to improve the features hierarchy.
- 2) Introduce adaptive features pooling that aggregates features from all feature levels for each proposal.
- 3) Augment the mask prediction with fully connected layers that increase the information diversity and mask with better quality are created [11].

The selected solution is the Mask R-CNN model because it best fits the project requirements, and despite being outperformed in terms of AP by different models, it gained the first position in the 3 COCO suite 2016 challenges, in which it was considered instance segmentation and detection of bounding boxes [7]. Furthermore, because it is a well-known algorithm and it is used in many implementations, the development, the support, and the maintenance of the code is easier. Hence, even if other solutions might lead to better results, they are generally slower in the training phase, making the Mask R-CNN a better model to use for the project.

4. Work done

All the code developed has been made available on GitHub (<https://github.com/alessiodimonte/TER>). The repository was used by the team to track the development of the project and can be used by other people as a guide to duplicate the work done. The repo is divided in two branches:

- 1) **master**: from which it is possible to download the ready-to-use code
- 2) **archive**: from which it is possible to see all the code divided by specific topic (e.g., instance segmentation, user interface, ...)
- 3) **google-colab**: from which it is possible to download the code adapted to work on Google Colab in case the local machine gives problems

Moreover, it is possible to find the master branch adapted to work on Google Colaboratory stored on Google Drive, in the case the machine used it is not working correctly or in case of errors (<https://tinyurl.com/4wtwmy3m>).

As for every Data Science project, it is important to follow a well-defined and iterative workflow to produce a valuable final product. As shown in Figure 13, the process consists in various steps, which the team performed and adapted according to the project requirements.

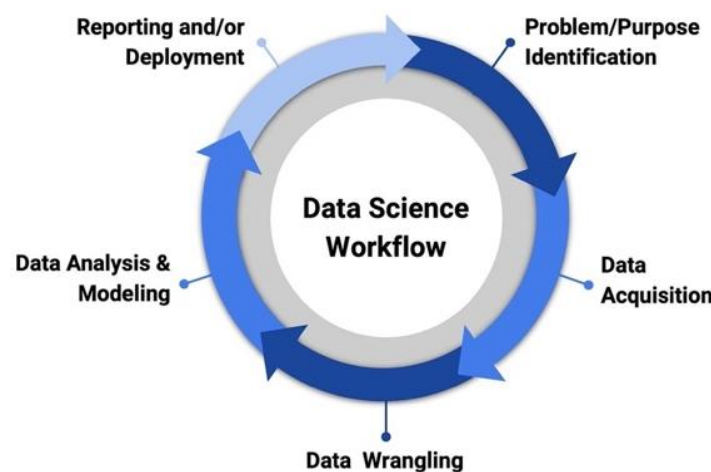


Figure 13: Data Science workflow, from

4.1 Identifying the problem and the purpose

This first step is useful for guiding the entire project, informing how all the steps need to be executed. In this case, the commitment of the project concerns the ecological surveillance, the biodiversity monitoring, and the marine ecosystem analysis. However, this goal is extremely general and for this reason the team started from a first job regarding the monitoring of sea areas.

The project stakeholders have the necessity to detect fishes in the marine environment in an automatic and fast way. From here the main idea of the project: exploiting Deep Learning and its potential to simplify and speed the work of Mediterranean Sea biologists (as mentioned in the chapter 1.2). Basically, it is needed to develop a software through which experts can start annotating fish images and train Deep Learning models over them, in order to perform object detection, instance segmentation, object tracking and counting.

4.1 Dataset acquisition

Once the problem statement was fixed, the source fueling the project, namely the dataset, must be selected. The first requirement given to the team was to use the “SeaCLEF 2017” dataset, a famous visual dataset for marine organisms [3]. Unfortunately, this dataset is no longer available and, after agreeing with the supervisors to use another dataset, the team started to work with the DeepFish one, which is containing approximately 40 thousand images and, differently from other datasets, is capturing the complexity of underwater habitats, considering also blurry pictures [4].

4.3 Megadetector adaptation

According to the Document of Work, the initial idea was to exploit Megadetector and use it to put bounding boxes automatically on the dataset. In fact, Megadetector is a model used to detect animals, people, and vehicles in camera trap images without classifying them, but just localizing them [16].

To achieve this, the team followed the instructions given in the GitHub page covering the previous work done by Fanny Simoes, who worked with the Megadetector in another project [17] and, after installing all the required dependencies, some tests have been performed:

- 1) Change the configuration of the model “pnmdetector.config” by trial and error with different values for SoftMax, batch size, steps, metric set, data augmentation, size images and optimizer.
- 2) Change the classes and substitute the set containing "human", "chamois", "roe deer", "doe", "deer", "fox", "badger", "wolf", "hare", "dog", "wild boar", "bicycle", "ibex" to the single class "fish".

Unluckily, results in the test phase were not adequate for the task. The blurriness of images and the difficulty of recognizing the different fish shapes lead to unsuitable outcomes. As shown in Figure 14, in some cases the detection was performed correctly (or partially), while in other cases it was completely wrong or absent.



Figure 14: Megadetector results on the DeepFish dataset

After these results, the team decided together with the supervisors to avoid the Megadetector part (which only returns bounding boxes) and focus on the study of different approaches to achieve the goal, as mentioned in the chapter 2.2.

4.4 User Interface for annotation

At the same time, three different User Interfaces were developed, according to the different needs that came up throughout the process. Nonetheless, according to the other phases, the third version of the UI was the best suit for the project needs.

1) First UI:

- a. Considers the input images and bounding boxes coming from the Megadetector.
- b. This UI was developed during the Megadetector adaptation phase, while it was not yet considered abandoning its use.
- c. This UI is written in Python using the PySimpleGUI library.
- d. It provides a way for the users to specify in the bounding box of one image is correct or not and in the first case annotate the name of the fish within the bounding box. Moreover, after the annotations are finished, they are saved in a .csv file so that they can be loaded again in the UI or used for another task.

2) Second UI:

- a. Considers the annotation format for object detection and bounding boxes.
- b. This UI was taken and adapted starting from the “Labellmg” available on GitHub [18].
- c. The UI is written in Python and through it is possible to put new bounding boxes on images, correct and delete them, choose the input images directory and the output directory where to store annotations.
- d. The format of annotations is YOLO, meaning that it creates a .txt file where for each box identifying an object a new line is created in the format “<object-class> <x> <y> <width> <height>”.

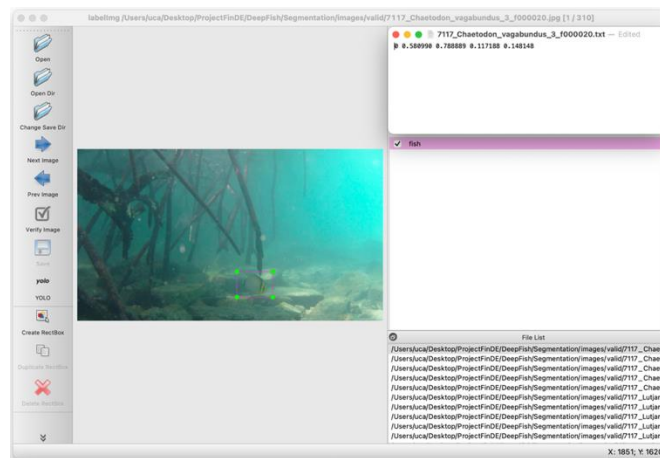


Figure 15: Labellmg User Interface

3) Third UI:

- a. Considers the annotation format for instance segmentation and segmentations masks.
- b. This UI was taken and adapted starting from the “MakeSense” available on GitHub [19].
- c. This UI is written in TypeScript and requires node.js version 12.x.x to work correctly.
- d. The UI supports all the annotations formats, but it is adapted to the annotation format COCO JSON for segmentation.
- e. Installation process:
 1. Install “node.js” version 12.x.x (<https://nodejs.org/download/release/v12.9.0/>).
 2. Download the modified version of the UI from the project GitHub repository (the folder name is “AnnotationTool”).
 3. Open the terminal and enter in the just downloaded folder (e.g., if the folder is downloaded on the Desktop it is needed to write “cd Desktop/AnnotationTool”).
 4. From the terminal write “npm install” and wait for the installation.
 5. From the terminal write “npm node clear node-sass” to bind the current OS with the npm installation.
 6. From the terminal write “npm start” and wait for the UI to be opened.
 7. Now, each time the UI is needed perform step 3 and 5 (the installation process of step 4 needs to be done just the first time)

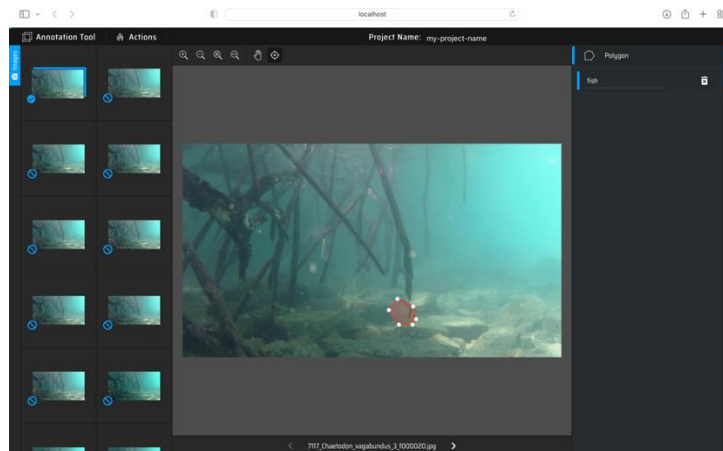


Figure 16: AnnotationTool User Interface

4.5 Model research

In parallel to the development of the UI, to decide the best techniques to exploit for the task, the team started to study and test different methods for instance segmentation and object tracking. A discussion of the models that might be used for this kind of task are described in chapter 2 and 3.

4.5.1 Mask R-CNN on a custom dataset

The chosen option was to perform Mask-RCNN on a custom dataset. Mask-RCNN is an approach that efficiently detects objects in an image and simultaneously generates a segmentation mask for each instance (chapter 2.2.1). In order to test it on the DeepFish dataset and see how well it performs a well-defined process was followed.

The first step was the creation of the annotations file for the selected 310 images, using the ready-to-use masks stored in the DeepFish dataset. The format compatible with Mask-RCNN, describing instance segmentation, is either “VGG JSON” or “COCO JSON”. For the task, it was decided to use the “COCO JSON” format, through which it was easier to retrieve the segmentation mask. Then, it was necessary to find a way to switch from a mask in black and white to a COCO JSON annotation file. To do so a script available on GitHub was adapted according to the needs to return a “COCO JSON” file storing the coordinates of each of the 310 masks [20].

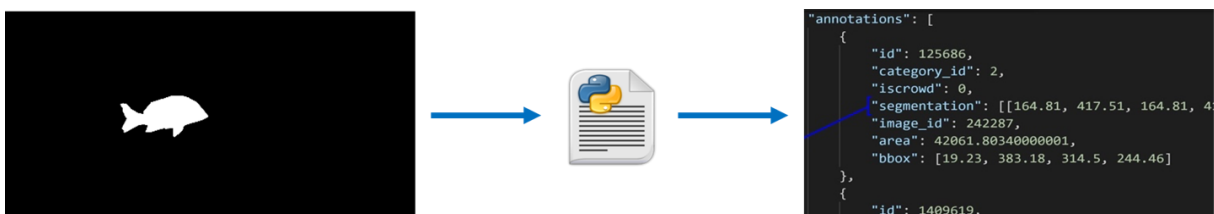


Figure 17: From Mask to COCO json process

After having the annotation file, it was possible to train the Mask-RCNN on the custom dataset. Following the instructions made available on GitHub for Mask-RCNN, the code was adapted as needed.

The model was not trained from scratch, but transfer learning was employed and starting from the pre-fit Mask-RCNN model on the MS-COCO dataset, it was tailored on the custom dataset by training the heads for 5 epochs.

As shown in Figure 18, the results generally provide a correct detection and segmentation of fishes with a higher level of accuracy. Nonetheless, the model is not at its finest because it still fails in certain cases, detecting fishes in a wrong way.

Anyhow, considering the limited number of images taken into consideration for the training (only 310) and the training for 5 epochs, it is sure that, incorporating in the training phase more images can significantly improve the precision of the detection and segmentation.



Figure 18: Mask-RCNN on DeepFish dataset

4.5.2 Object tracking with OpenCV

Even though a dataset containing videos on which we could test object tracking was not available, the team examined the DeepFish dataset and found that some images were temporally taken one after the other. Thus, multiple frames were put together to create a video to try to test this first approach on the DeepFish dataset.

In this first methodology only OpenCV was used, which has useful built-in functions for detection and tracking. For detection a naïve technique based on image processing was used, while for tracking it was applied a mathematical function measuring the distance between centroids in two subsequent frames.

First, the “object_detector” was created with the “createBackgroundSubtractorMOG2()” function, which creates a mask in black and white for the moving elements. This is one kind of background subtraction methods offered by OpenCV. After that, boxes around the detected objects were drawn, assigning a unique ID to each box, and the “EuclideanDistanceTracker()” was employed to perform object tracking [21].

Results from this technique are very poor, as shown in Figure 19. This is mainly because of the input video, which is hard to follow even for a human eye due to the large distance between fishes in two subsequent frames.



Figure 19: Object tracking with OpenCV on DeepFish dataset

4.5.3 Object tracking with DeepSORT

The second approach studied for object tracking was DeepSORT. Unluckily, it was not possible to test it on the DeepFish dataset because of the dataset's nature, which has not video content. Nonetheless, for personal interest the team tried it on a sample video, and results showed a significant potential for this algorithm if applied in the project context, as shown in Figure 20 and on the video demo visible [here](#):

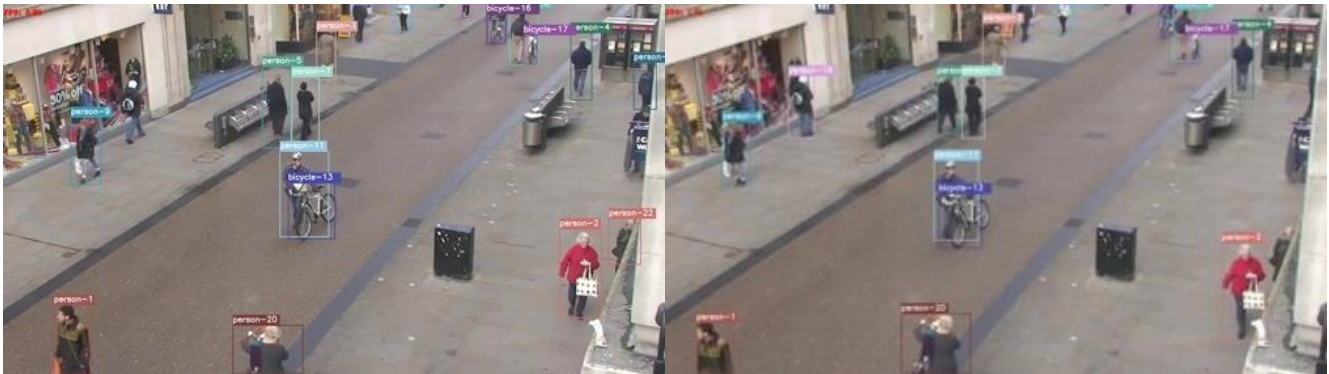


Figure 20: DeepSORT on a video sample

4.6 Project refinement

Together with the supervisors and stakeholders, the team decided to give to the project a different extent: avoid the tracking part because of the incompatible dataset and focus on enriching the user interface. The new UI should allow to start the Mask-RCNN training phase and see the results on new images directly from the User Interface.

4.7 User Interface modification

In order to satisfy the new requirements, some adjustments to the UI needed to be performed.

4.7.1 “Start training” button

The first change was the addition of a clickable button to start the training phase. Moreover, after the training, the model can do predictions on new images and, directly from the UI, experts have the opportunity to check and correct the predicted masks, in order to constantly ameliorate the quality of the training dataset.

This functionality was conceived such that:

1. From the UI it is possible to annotate images.
2. Click “Start training” and pass annotations and images to the Mask-RCNN model.
3. After the training part ends, the test phase returns new images never seen by the model and the related annotations.
4. From the UI it is possible to correct the annotations just made by the test phase.
5. Start the training phase with the training dataset enriched by the images and annotations coming from the step 3 and 4.
6. Repeat step 2, 3, 4, 5.

4.7.2 Flask

To run the training phase from the UI the team came up with the solution provided by Flask, which through a simple python code can call the Mask-RCNN script after clicking the button “Start training”. Flask in fact allows to create web-based application and performs calls to executables [22]. In this specific case the executable to call is the script for the Mask-RCNN training and the code to do so is shown in Figure 21.

A screenshot of a code editor window titled '*startTrainingFlask.py - /Users/uca/Desktop/GitHubMakeSenseModified/MakeSenseM...'. The code is written in Python and uses the Flask web framework. It defines a Flask application 'api' and a route '/start_training' that, when accessed via GET, runs a Jupyter notebook 'maskRCNN_24012022_v3/finalNotebook_trainAndTest.ipynb' and returns a JSON response with 'success': True. The code also includes a standard if __name__ == '__main__': api.run() block.

```
from flask import Flask
import json
import os

api = Flask(__name__)

@api.route('/start_training', methods=['GET'])
def startTraining():
    os.system("jupyter notebook ../maskRCNN_24012022_v3/finalNotebook_trainAndTest.ipynb")
    response = json.dumps({'success': True}), 200, {'Access-Control-Allow-Origin' : '*'}
    return response

if __name__ == "__main__":
    api.run()
```

Ln: 9 Col: 20

Figure 21: Flask code

To make the button work correctly it is fundamental to run the Flask python script before clicking the button “Start training” on the UI. It was necessarily needed to pass through Flask because for safety reasons there is no way to execute a program directly from another application and to respect these requirements the following commands must be executed:

1. Open the terminal and enter in the location of the folder of the downloaded UI (same as step 4.3: User Interface for annotation).
2. Write the command “python startTrainingFlask.py”.
3. Open the UI and click “Actions” and then the “Start training” button.

5. Conclusions

As described, the project evolved throughout the process, due to the requirements imposed and the difficulties faced. In the Document of Work, it was explained the initial solution through a use-case diagram (see Annex 1), but the final product offered is a built-in User Interface including all the functionalities.

As shown in Figure 22, the UI plays a central role in all the processes: through the UI is in fact possible to:

1. **Import images:** these images are the ones that experts need to annotate from scratch or to check/correct in the case the annotations are provided by the Mask-RCNN model.
2. **Import annotations:** the Mask-RCNN model produces an annotation file in the format of “COCO JSON” for the masks related to the test images.
3. **Create/modify/export annotations:** after creating from scratch or correcting the annotations, experts need to export the annotations in a “COCO JSON” format to pass it to the Mask-RCNN model for the training phase.
4. **Launch training phase:** when experts finish to annotate/correct annotations they can start the training phase with the set of images and annotations they just created.

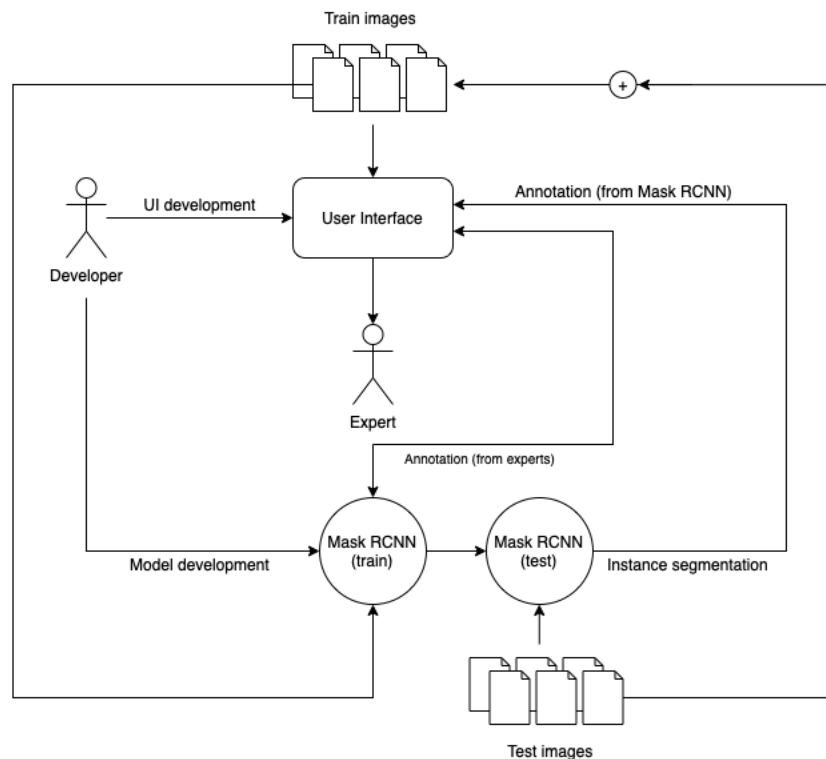


Figure 22: Use case diagram

With respect to the initial goal, it was not implemented in the final solution the object tracking and the counting parts because of the dataset unsuitability and the new project constraints. Nonetheless, a study on how to perform these tasks was made, as shown in chapter 4.5.2 and 4.5.3, to leave them as a basis for future developments.

Further extensions of the project are:

- 1) Since the project is modular, it is possible to change the instance segmentation algorithm used and find a Mask R-CNN algorithm working with Tensorflow 2 (instead of Tensorflow 1) and/or substitute Mask R-CNN with another up-to-date algorithm.
- 2) Add to the model the multi-class classification in order to train the model and recognize different species of fishes. The team decided to train the model to distinguish fishes because of the little availability of annotated data and because of the preference to focus on the detection rather than classification.
- 3) Implement object tracking and counting. Starting from a different dataset which includes videos, train the DeepSORT model on it.
- 4) Use the university virtual machine. Because of the network communication it was difficult to train the model on the VM and because of the impossibility to install VNC on the VM it was impossible to use the User Interface directly on the VM. Hence, it would be useful to find a way to maintain a stable network communication and provide the access to the VM through a VNC client.

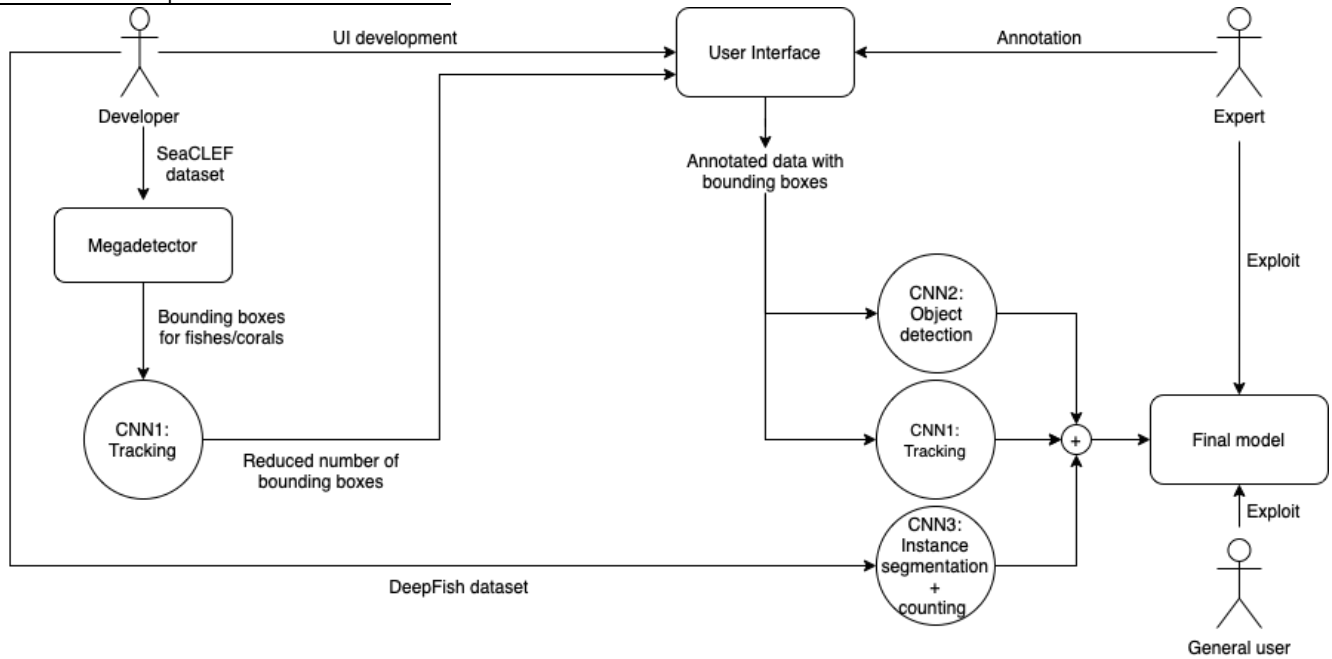
6. References

- [1] Computer Science, Signals and Systems Laboratory of Sophia Antipolis, <https://www.i3s.unice.fr/>
- [2] Ecology and Conservation Science for Sustainable Seas, <http://ecoseas.unice.fr/>
- [3] SeaCLEF, 2017, <https://www.imageclef.org/lifeclef/2017/sea>
- [4] Alxayat Saleh, Issam H. Laradji, Dmitry A. Konovalov, Michael Bradley, David Vazquez, Marcus Sheaves, 2020, “A Realistic Fish-Habitat Dataset to Evaluate Algorithms for Underwater Visual Analysis”, Arxiv, <https://arxiv.org/abs/2008.12603>
- [5] Yali Amit, Pedro Felzenszwalb 2020, “Object detection”, ReserachGate, https://researchgate.net/publication/339792032_Object_Detection
- [6] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, 2015, “You Only Look Once: Unified, Real-Time Object Detection”, <https://arxiv.org/abs/1506.02640>
- [7] Abdul Mueed Hafiz, Ghulam Mohiuddin Bhat, 2019, “A survey on instance segmentation: state of the art”, Springer, <https://link.springer.com/content/pdf/10.1007/s13735-020-00195-x.pdf>
- [8] Koen E. A. van de Sande, Jasper R. R. Uijlings, Theo Gevers, Arnold W. M. Smeulders, 2011, “Segmentation As Selective Search for Object Recognition”, <https://ivi.fnwi.uva.nl/isis/publications/bibtexbrowser.php?key=vandeSandeICCV2011&bib=all.bib>
- [9] Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick, 2017, “Mask R-CNN”, Arxiv, <https://arxiv.org/abs/1703.06870>
- [10] Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, Chen Change Loy, Dahua Lin, 2019, “Hybrid Task Cascade for Instance Segmentation”, Arxiv, <https://arxiv.org/abs/1901.07518>
- [11] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, Jiaya Jia, 2018, “Path Aggregation Network for Instance Segmentation”, Arxiv, <https://arxiv.org/abs/1803.01534>
- [12] Zhaojin Huang, Lichao Huang, Yongchao Gong, Chang Huang, Xinggang Wang, 2019, “Mask scoring R-CNN”, Arxiv, <https://arxiv.org/abs/1903.00241>

- [13] Sergey Zagoruyko, Adam Lerer, Tsung-Yi Lin, Pedro O. Pinheiro, Sam Gross, Soumith Chintala, Piotr Dollár, 2016, "A MultiPath Network for Object Detection", Arxiv, <https://arxiv.org/abs/1604.02135>
- [14] Daniel Bolya, Chong Zhou, Fanyi Xiao, Yong Jae Lee, 2019, "YOLACT: Real-time Instance Segmentation", Arxiv, <https://arxiv.org/abs/1904.02689>
- [15] Lin T-Y, Maire M., Belongie S., Hays J, Perona P, Ramanan D, Dollar P, Zitnick L, 2014, "Microsoft COCO: Common Objects in Context", Springer, https://link.springer.com/chapter/10.1007/978-3-319-10602-1_48
- [16] Dan Morris, Siyu Yang, Sara Beery, Louis Beaumont, Christopher Yeh, Brian Hogg, 2022, "Megadetector", GitHub, github.com/microsoft/CameraTraps/blob/main/megadetector.md
- [17] Fanny Simoes, 2021, "PNM detector v1.0", GitHub, github.com/FannySimoes/Projet_PNM
- [18] Tsutalin, 2022, "LabelImg", GitHub, <https://github.com/tzutalin/labelImg>
- [19] Piotrek Skalski, 2022, "Make-Sense", GitHub, <https://github.com/SkalskiP/make-sense>
- [20] Chrise96, 2021, "Image to COCO json converter", GitHub, <https://github.com/chrise96/image-to-coco-json-converter>
- [21] OpenCV, https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html
- [22] Grinberg, M, 2018, "Flask web development: developing web applications with python", Flask, <https://flask.palletsprojects.com/en/2.0.x/>

7. Annex

7.1 Use case presented in the DoW



Annex 1: use-case diagram presented in the DoW

7.2 User manual

[LOCAL MACHINE]

1. Install the UI

1.1 Download from the GitHub repository the branch “master”

- In the terminal window write “git clone <https://github.com/alessiodimonte/TER.git>”



1.2 Download and install node.js version 12.x.x and npm version 6.14.x (other version might not work with the UI) (for example: <https://nodejs.org/download/release/v12.22.8>)

1.3 Open the terminal in the just downloaded folder in the path “TER/annotationTool”

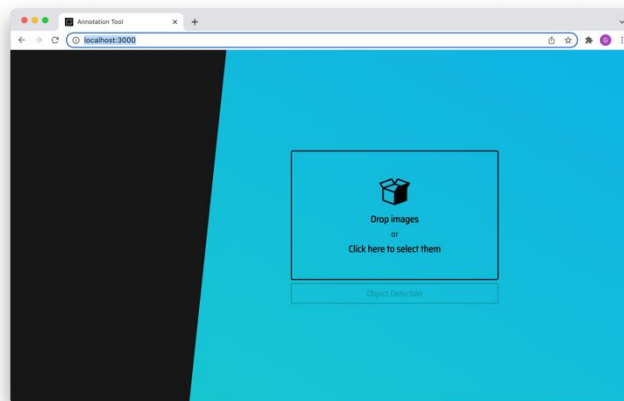
- For example, if you have downloaded the folder on the Desktop, in the terminal you need to write “cd Desktop/TER/annotationTool”

1.4 In the same terminal window write “npm install”

1.5 In the same terminal window write “npm rebuild node-sass”

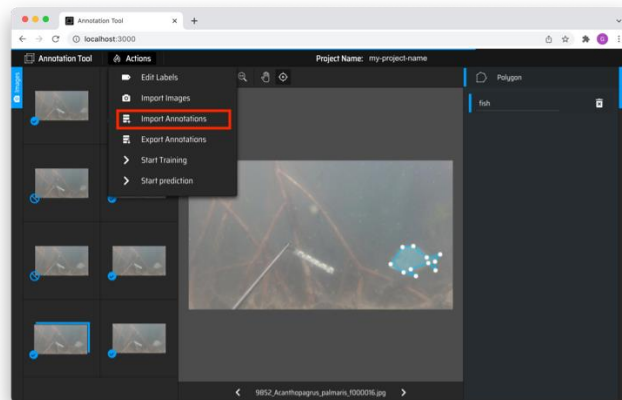
1.6 In the same terminal window write “npm start”

1.7 The UI will be automatically opened in a browser tab (the process might take several minutes), in the case nothing is opened just write on the browser “http://localhost:3000/”

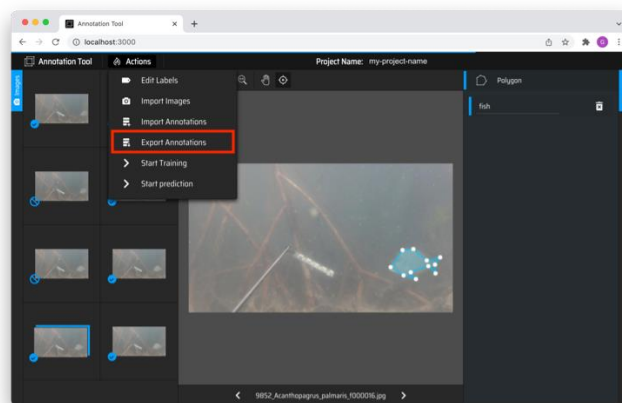


2. Use the UI

- 2.1 Click on the “Drop images or Click here to select them” button and select the images you want to annotate
- 2.2 Click on the “Object Detection” button
- 2.3 Define label(s) by either click on the “+” or “Load labels from file” button
- 2.4 Click “Start project” button
- 2.5 Load the annotations with the button “Actions” → “Import Annotations” and select the annotation file corresponding to the images uploaded in step 2.1 (or make annotations by yourself)
 - The images with the tick icon means that are annotated, the ones with the forbid icon are not



- 2.6 From the UI it is possible to create new annotations and change the imported annotations masks
- 2.7 It is possible to export the annotation by clicking “Actions” → “Export Annotations” → “COCO JSON”



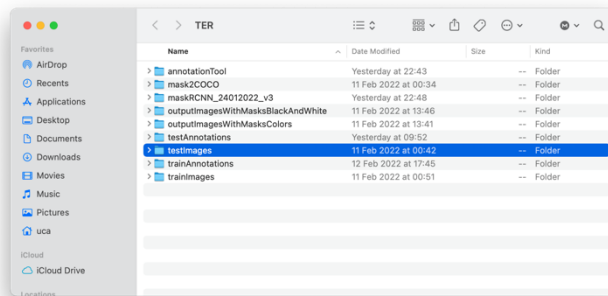
3. Training and testing

- 3.1 Open a new terminal window in the folder “TER/ maskRCNN_24012022_v3”

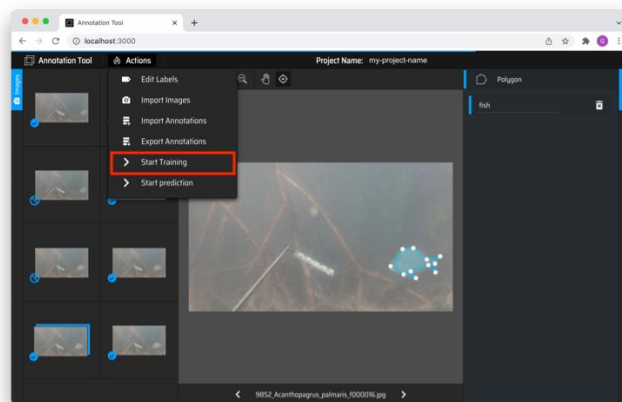


3.2 In the same terminal window write “python startTrainingFlask.py”

3.3 In the folder “TER/testImages” put the images on which you want to automatically create your annotations



3.4 Return to the UI and click on the “Start Training” button in the UI




3.5 The script is run in background and saves the annotation file in the folder “testAnnotations”

[GOOGLE COLAB (in the case the training and/or testing does not work on the local machine)]

4. Training and testing

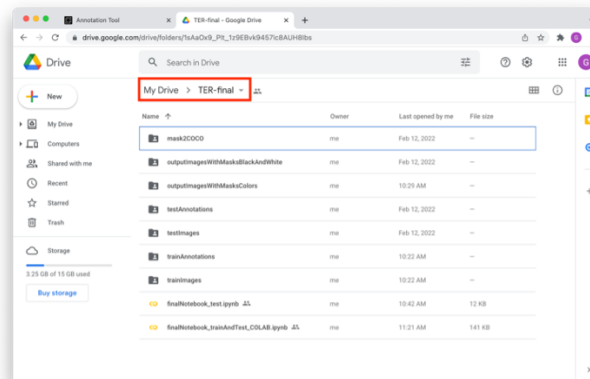
4.1 Download from the GitHub repository the branch “google-colab”

- In a terminal window write “git clone google-colab <https://github.com/alessiodimonte/TER.git>”



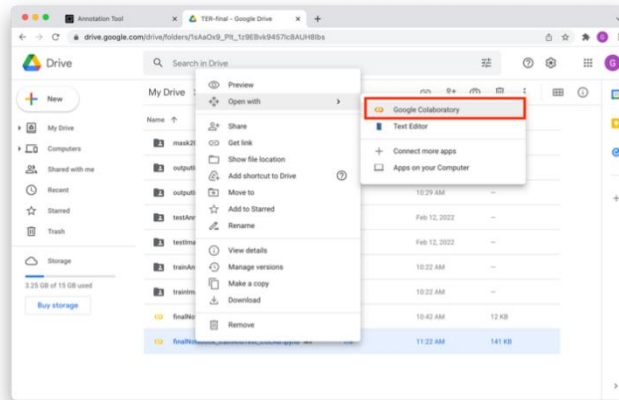
```
uca ~ -zsh - 80x24
Last login: Mon Feb 14 11:13:34 on ttys000
(base) uca@MacBook-Pro-di-Alessio ~ % git clone google-colab https://github.com/alessiodimonte/TER.git
```

4.2 Copy the folder “src/MaskRCNN/TER-final” on your Google Drive in the path “My Drive”

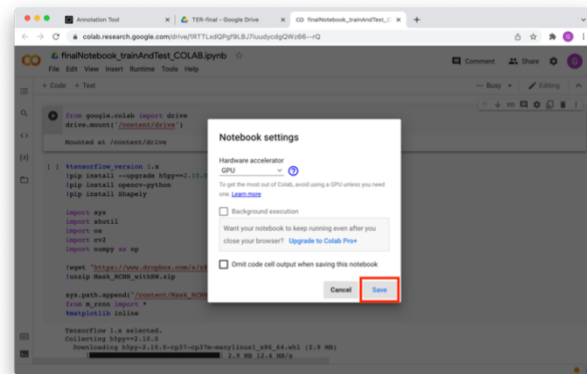
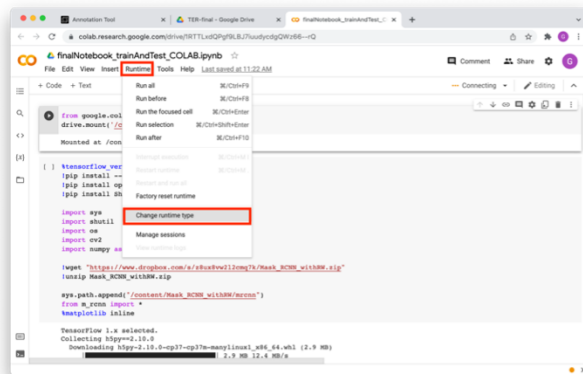


- 4.3 Make sure that the folders “outputImagesWithMasksBlackAndWhite”, “outputImagesWithMaskColors”, “testAnnotations”, “testImages”, “trainImages” and “trainAnnotations” are empty (otherwise, empty them)
- 4.4 Put in the “testImages” folder the images on which you want to automatically create your annotations
- 4.5 Put in the “trainImages” folder the images on which you want to train the model
- 4.6 Put in the “trainAnnotations” folder the json annotation file corresponding to the images of the step 4.5
 - You can use the annotation file generated from the testing phase
 - You can use the annotation file generated from manually annotated images
 - You can use the annotation file generated from external sources

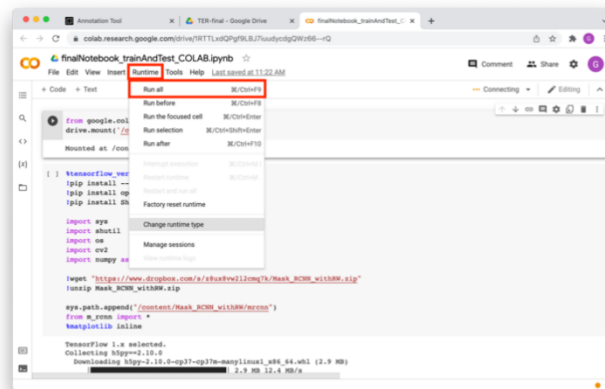
4.7 Right click with the mouse on the file “finalNotebook_trainAndTest_COLAB.ipynb” → open with → Google Colaboratory



4.8 Click on Runtime → Change runtime type → Hardware accelerator → GPU → Save to enable the GPU acceleration

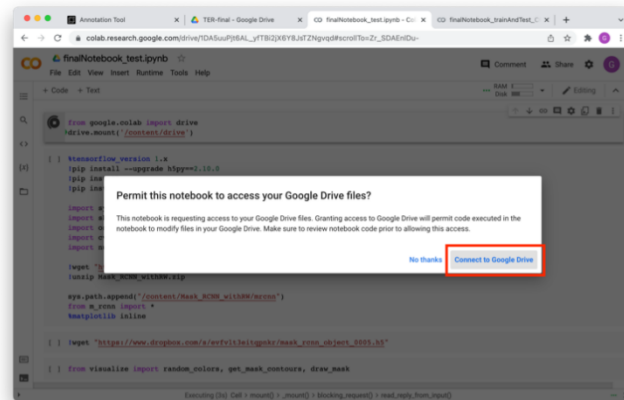


4.9 Click on “Runtime” → “Run all” to run the whole notebook

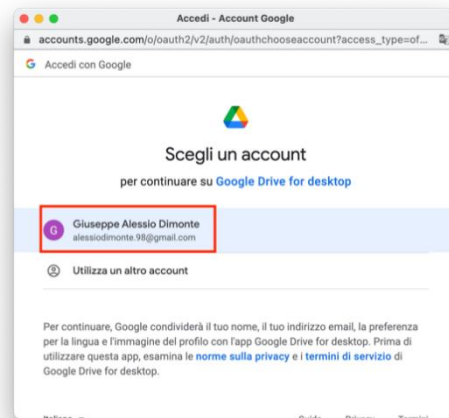


4.10 Google Colab will you ask to mount your files (it is a mandatory step to perform)

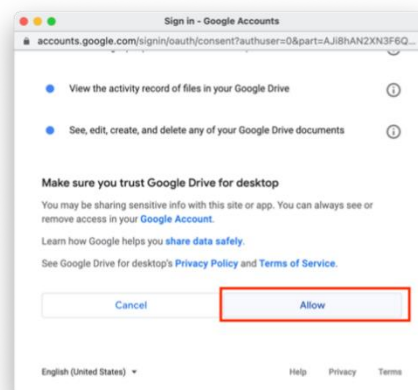
- Click on “Connect to Google Drive”



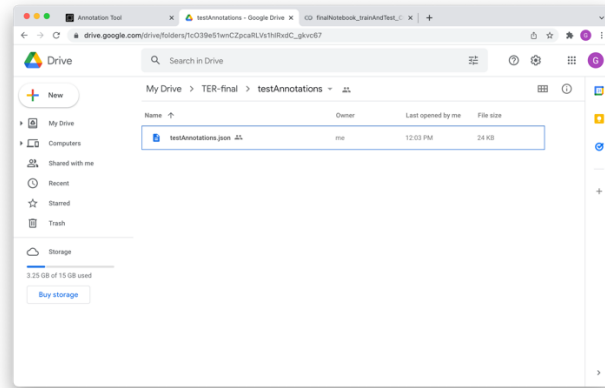
- Click on your account name



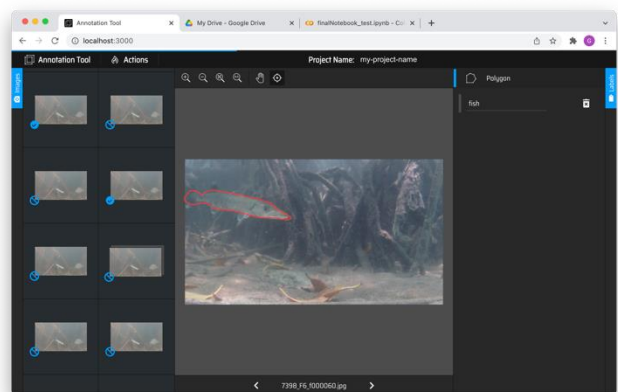
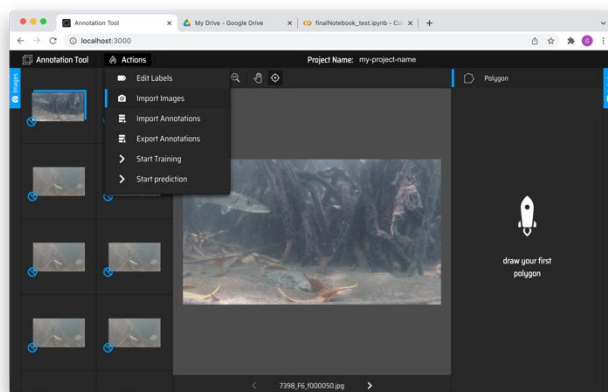
- Click on “Allow”



4.11 Go back to the Drive folder and download the “testAnnotations.json” file located in the folder “/MyDrive/TER-final/testAnnotations”



4.12 Open the UI installed locally and follow the instructions of steps 1 and 2 (import images and annotations → see the result)



5. Testing

5.1 Download from the GitHub repository the branch “google-colab”

- In a terminal window write “git clone google-colab <https://github.com/alessiodimonte/TER.git>”

5.2 Copy the folder “src/MaskRCNN/TER-final” on your Google Drive

5.2.3 Make sure that the folders “outputImagesWithMasksBlackAndWhite”, “outputImagesWithMaskColors”, “testAnnotations”, “testImages”, “trainImages”, “trainAnnotations” are empty (otherwise, empty them)

5.3.4 Put in the “testImages” folder the images on which you want to automatically create your annotations

5.5 Right click on the file “finalNotebook_test_COLAB.ipynb” → “Open with” → “Google Colaboratory”

5.4.5.6 Click on “Runtime” → “Change runtime type” → “Hardware accelerator” → “GPU” → “Save” to enable the GPU acceleration

5.55.7 Click on “Runtime” → “Run all” to run the whole notebook

5.65.8 Google Colab will ask you to mount your files (it is a mandatory step to perform)

- Click on “Connect to Google Drive”
- Click on your account name
- Click on “Allow”

5.75.9 Go back to the Drive folder and download the “testAnnotations.json” file located in the folder
“/MyDrive/TER-final/testAnnotations”