A System for Automating the Detection and Counting of Frogs in Small Passages

David Benedikt Heye
Marc Ludevid Wulf
Ben Robert Sturgis
École Polytechnique Fédérale de Lausanne, Switzerland

***Abstract*—The aim of this project is to classify and count frogs that pass subterranean passages. To solve this problem, we took two different approaches: the first one uses image classification with ResNet [1], the second approach builds upon the YOLOv5 [2] object detection algorithm.**

I. Introduction

Amphibians that primarily live in terrestrial habitats such as forests often need to cross roads to get to a waterfront location where they breed. This poses a great danger to their lives. To help these animals survive this task without damage, amphibian fences are often built to prevent them from entering the street. The amphibians are then manually carried to the other side of the road. An alternative, less laborious approach is to install small subterranean tunnels that allow amphibians to safely get to the other side of the road on their own.

In our project, we implemented two methods to help the Institute of Natural Resource Sciences at the ZHAW to automatically count the amount of frogs passing through those passages while taking note of the direction in which they are moving. In the first approach amphibians are localized based on their movement and then classified using ResNet [1], a deep convolutional neural network. For the second approach, we use the YOLOv5 [2] object detection algorithm trained on a custom dataset to detect the frogs in each frame, combined with an object tracking algorithm optimized to our use case to determine how many frogs move in which direction. Finally, we compare the two methods based on different aspects such as effort of training or performance.

II. Dataset

We were initially given a dataset of about 45,000 videos that were automatically recorded using motion sensitive cameras in the middle of 17 subterranean passages. The videos are composed of the three channels red, green and blue, with all color components being equal at night, resulting in a black and white video. These videos have already been manually reviewed to filter out false triggerings and to manually count the amount of animals over the course of several months. In the end, we are left with around 9,000 events where there is indeed one or more animals moving through the passage. These animals do not only include frogs, however we focus only on the detection and counting of frogs for our project.

A. Movement Detection

Introduction: In our first approach, we used a movement tracking algorithm in combination with an image classification model. The tracking algorithm uses basic image manipulation operations offered by *OpenCV* [5] and for the image classification we decided to use the pre-trained *ResNet34* model, a deep residual convolutional neural network specialized on image classification [1]. Further training for our specific use case was required and will be explained below.

Movement Detection and Tracking: The first step to track the movement of animals in the videos is to detect any movements and to generate bounding boxes around these. To do so we processed the frames of the video sequentially and used the absolute difference between successive frames to get the areas containing changes. To stabilize this process we used a combination of buffering, denoising and blurring of the images.

Afterwards, this information is used by a tracking algorithm that outputs the position of a moving entity in every frame of the video. The algorithm tries to match any area of movement to a close one in the previous frame. If that is not possible we assume that the movement is generated by an animal that was previously stationary. We therefore set the position of that animal in all previous frames to the start of the movement. For all movement areas that were not matched to any new ones we maintain the location of the animal.

Our testing showed that this algorithm by itself produced good tracking behavior and only had major issues tracking long jumps of frogs. In those cases we often observed the algorithm generating two animal tracking entities, one for the movement before the jump and one after. To further improve the algorithm we introduced a jump detection step by looping over all pairs of entities and checking if the beginning of the movement of one entity is near the end of the movement of another entity and if the movement ended at a similar frame as the start of the movement in the other tracking entity. In these cases we assume that there has been a jump of a frog and we merge them.

Dataset Generation: To train the *ResNet34* [1] model we needed a dataset of images and the correct labels. Since we only consider frogs we only needed three labels, one for a single frog, one for an *amplexus* [4], i.e. a mating behavior of frogs where a male frog grasps a female with his front legs (see fig. 1), and one for empty images. If the movement detection and tracking algorithms were perfect the empty label would not be necessary, but we introduced it as an additional measure to detect false positives.

For the generation of the dataset we did not require any additional manual labeling, we were in fact able to fully automate this step. To generate the images for the frogs and for the amplexus we used the tracking algorithm to generate potential images and then used the information about the number of frogs that crossed in the dataset that we were given. When the number of movements detected by the tracking algorithm matches the number in the dataset we can assume that the animals tracked are all single frogs. If the number in the dataset is exactly double then all animals tracked are amplexus. With this approach we generated around 7,000 images with single frogs and 4,500 with amplexus, only using the videos provided by the first camera. To generate the empty images we assumed that after an animal has moved away from a part of the image that area would be empty. The algorithm we implemented using this assumption generated around 3,500 empty images. We are thus left with a total of around 15,000 images for training, which we manually split into a training and a separate validation batch at a ratio of around 90% to 10%.

Training: We trained the already pre-trained *ResNet34* [1] model in two steps. First we only trained the last layer, which was untrained since we reduced the amount of classes from 10 to 3, freezing the other layers. Training for 10 epochs was in fact enough to obtain validation accuracies of above 98%. After unfreezing the rest of the model we trained another 10 epochs and got accuracies of above 99.5%. The increase of the learning rate after unfreezing the model initially decreased the performace of the model but then generated even better results than before, as shown in fig. 2.

Combining the movement detection and tracking algorithms with the trained *ResNet34* [1] model we obtained a system that is able to first locate moving entities in the videos and track their position over time to enable the counting of frogs.

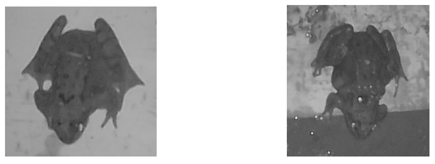
(a) Frog (b) Amplexus

Figure 1: Frog and amplexus

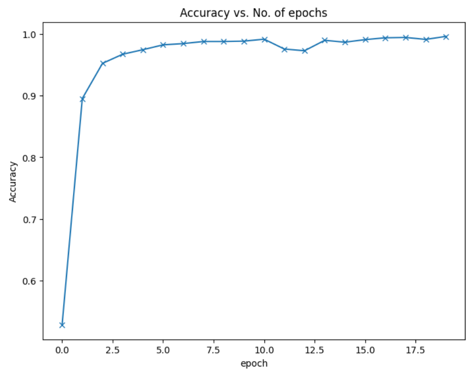

Figure 2: Training Accuracy

B. *YOLOv5 Object Detection*

Introduction: In our second approach, we use an object detection algorithm that detects the frogs contained in each frame of the video. With a second algorithm, we track the detected entities across all frames and determine the direction in which they are moving based on their start and end positions. For the object detection we decided to use the YOLOv5 ("*You Only Look Once*") implementation by Ultralytics [2] to detect instances of frogs and amplexus in the videos given to us. The YOLOv5 algorithm is especially popular because of its increased speed over other object detection algorithms. For the tracking, we implemented our own algorithm that is adapted to our use case.

Data Exploration: To train the YOLOv5 network, we manually labeled a selection of frames from our given dataset with labelImg [7]. We focused on the videos from the first two cameras and extracted three frames from all videos with two or more frogs moving through. After training YOLOv5 [2] on this dataset, we noticed that it was not able to distinguish between a single frog and an amplexus. Since in an amplexus the bounding boxes of the two frogs strongly overlap, one of the two boxes is removed by YOLO's non-maximum suppression (see paragraph on *How YOLO works*). This is why we decided to re-label those images in which an amplexus occurs and introduce a new label "amplexus" besides the original label "frog". Additionally, we labeled some individual frogs to balance our dataset.

We finally obtained a dataset consisting of 1198 images where 85 are background images, i.e., they contain no labeled objects. In total, we labeled 1545 instances of frogs and 1603 instances of amplexus.

How YOLO works: YOLO intends to locate and classify objects in an input image. For this purpose, YOLO first divides the image into a grid of $S \times S$ cells. Each grid cell is then responsible for detecting the objects whose centers fall within the cell. In order to detect more than one object in a grid cell, B anchor boxes are assigned to each cell. Anchor boxes are bounding boxes with predefined height and width that are used as a prior and then adjusted

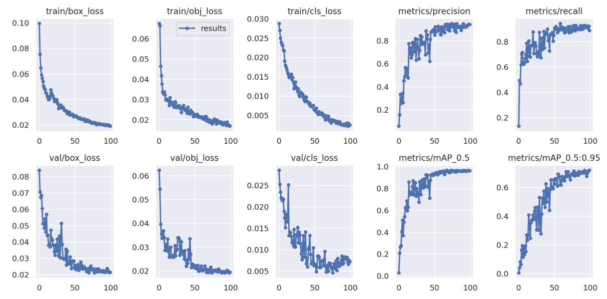

Figure 3: Training results after 100 epochs

Setting	Frogs	Amplexus
High augmentation hyps, random	78%	80%
High augmentation hyps, pre-trained	86%	94%
Standard hyps, random	91%	91%
Standard hyps, pre-trained	92%	96%
Modified standard hyps, random	88%	91%
Modified standard hyps, pre-trained	89%	93%

Table I: Percentage of objects correctly labeled after 100 epochs

during the detection process. Each bounding box is encoded with five parameters that specify its position and confidence score. The underlying YOLO convolutional neural network thus maps the input image to an output of dimension $(S \times S \times (B \cdot (5 + C)))$, where C corresponds to the number of classes to be detected. As a measure of the quality of a bounding box, YOLO uses the intersection over union (IoU) of the predicted and ground truth boxes. This eliminates bounding boxes that poorly estimate the ground truth box. To avoid detecting the same object multiple times, YOLO applies non-maximum suppression. Between overlapping bounding boxes that exceed an IoU threshold, the one with the highest probability is selected and the others are removed. [6]

Training and Hyperparameter Optimization: As there is no pre-trained model for YOLOv5 [2] that suits our use-case of frog detection, we decided to train our own model. We split our dataset into three parts: 100 images for validation, 100 images for testing and the remaining 998 images for training. To maximize the accuracy of our training, we trained YOLOv5 multiple times with different settings, both from randomly initialized weights and from pre-trained weights trained on the COCO [3] dataset by the YOLO team [2], and with different initial hyperparameters. We found that training on a slightly adapted version of the default hyperparameters and starting from pre-trained weights yields a very good result that we decided to build upon, even though training with non-modified default hyperparametery yields a slightly better result (see table I). We made this decision due to YOLO's behavior during hyperparameter evolution, where it does not evolve hyperparameters whose value is initially 0.

YOLOv5 has around 30 hyperparameters used for training. These parameters include settings such as the

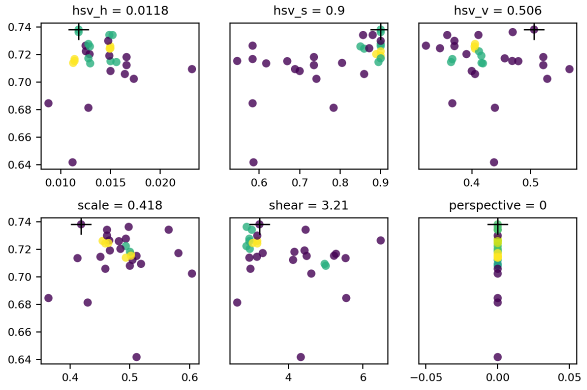

Figure 4: Evolving hyperparameters for 30 generations

momentum for SGD as well as parameters that control data augmentation, which YOLOv5 takes care of automatically during training. YOLOv5 augments data for example by flipping or stretching images or by combining multiple images into a mosaic. The default hyperparameters are optimized for training YOLOv5 on the COCO [3] dataset from scratch, however YOLO is capable of running a “hyperparameter evolution”, i.e., it can improve the values of the hyperparameters to yield better training results. YOLOv5 tries to maximize a *fitness* value which depends on the IoU of the predicted and the ground truth bounding box and the mean average precision metric, which is calculated as the mean of the average precision over all classes. Average precision is calculated as the area under a precision vs recall curve per class for the model. A prediction is considered to be correct if its IoU is above 0.5. YOLOv5 tries to find the optimal parameters using a so-called “genetic algorithm” [2] as grid search is highly unsuitable for finding an optimum in around 30 dimensions. Running hyperparameter evolution is very costly and takes a large amount of time. We ran evolution based on the weights learned by the 100 epoch training explained above.

We performed hyperparameter evolution based on our 100 epoch training described above by training an additional 10 epochs 30 times while YOLOs genetic algorithm tries to optimize the values of the hyperparameters. Figure 4 shows the results of the evolution for a selection of hyperparameters which control parts of the data augmentation. The y-axis shows the fitness and the x-axis shows the value of the respective hyperparameter. The optimal value chosen by the algorithm is marked with a small cross. A vertical distribution means that evolution on the respective parameter is disabled. To improve the predictions made by our model, we trained YOLOv5 for another 50 epochs starting from our initial 100 epoch training, using the evolved hyperparameters. Indeed, this training increases the percentage of frogs that are correctly labeled from 89% to 91% and the percentage of amplexus from 93% to 97%. Furthermore, the percentage of frogs detected in the background, i.e., where there should be nothing detected

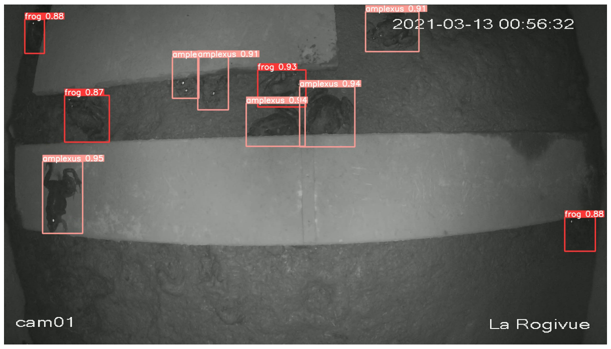

Figure 5: YOLOv5 predictions

	Species, count, direction	Only species and direction
YOLOv5	68.39%	85.65%
ResNet	64.24%	79.82%

Table II: Accuracy of YOLOv5 and ResNet

(false positive), is decreased by 25% to 42%. Figure 5 shows the result of frog detection by YOLOv5 in one of the videos from our dataset.

Tracking Algorithm: Using the YOLOv5 object detection algorithm, the frogs contained in each frame of the video are detected and localized using bounding boxes. To determine how many frogs occur throughout the video and in which direction they move, we implemented a tracking algorithm. The tracking algorithm maps the bounding boxes of the frogs from the previous frame to the bounding boxes in the current frame and updates their position accordingly. The assignment of the bounding boxes of the successive frames is done by a slight modification of the *Hungarian algorithm.* It takes into account that frogs can enter or leave the image in each frame and to a certain extent that the YOLOv5 object detection algorithm can make mistakes sometimes. In some videos it happens that two single frogs come together and form an amplexus or that an amplexus splits into two frogs. This can lead to problems while tracking as now two bounding boxes (frogs) from the previous frame may merge into a single box (amplexus), or one box (amplexus) may divide into two (frogs). However, this phenomenon is also caught by the tracking algorithm. In order to finally determine to which class a tracked object belongs, the number of times the object was assigned to a class and the probability of this assignment are taken into account.

III. Results

With both tracking algorithms completed, we are now able to count animals and to find out the direction in which they are moving. For this purpose, we implemented a simple counting algorithm that checks for the tracked objects whether they have moved a certain distance downwards (forest) or upwards (water) and crossed the center of the image.

Comparing the results of counting frogs with YOLOv5 and ResNet, we see that both approaches yield similar results (see table II). They produce exactly the same result as the manually created dataset that was given to us in around 65% of all cases. When not considering the amount of animals but only their species (frog) and the direction in which they move (water or forest), the results are the same in around 80% of all cases. Both times, YOLOv5 performs slightly better. Both methods of counting have no false positives, i.e., they never count frogs that do not exist in the videos.

Nevertheless, the accuracies given in table II should be taken with caution, as the counting in the given dataset is sometimes not very consistent. For example, in the first video of camera 01, there is one frog walking towards the water. However the dataset counts five frogs. Both YOLOv5 and ResNet were able to identify that in this case there was indeed only one frog walking towards the water.

IV. Conclusion

To accomplish the task of classifying and counting amphibians, we took two different approaches, each with its own advantages and disadvantages. Since we could not find a pre-trained model or a labeled dataset for amphibian detection, it was necessary to label a great number of images ourselves to achieve good results with YOLOv5. We believe that using a larger dataset would further improve the results. The motion detection method simplifies this time-consuming work of manually creating a dataset for training. It automatically extracts the bounding boxes of the objects and selects the appropriate class using the data from the original dataset given to us. At the same time, it reduces the object detection problem to the image classification problem, for which models can be trained more quickly and which is generally easier to solve. On the other hand, object detection with YOLOv5 generally simplifies the amphibian tracking. However, the introduction of the amplexus label in addition to the frog label complicated this process. Manual testing shows that both methods work well in general, but can be improved in exceptional cases.

Our work shows that accurately automating the detection and counting of animals in amphibian passages is not a trivial task. Future work may include training our methods on a larger dataset to improve predictions made by our models. Adding new species to be tracked should be an easy task for both approaches.

References

[1] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016, pp. 770–778.

[2] Glenn Jocher et al. *ultralytics/yolov5: v6.0 - YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support.* Version v6.0. Oct. 2021. DOI: 10.5281/zenodo.5563715. URL: https://doi.org/10.5281/zenodo.5563715.

[3] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context.* 2015. arXiv: 1405.0312 `[cs.CV]`.

[4] Ivelin A Mollov et al. "Cases of abnormal amplexus in anurans (Amphibia: Anura) from Bulgaria and Greece". In: *Biharean Biologist* 4.2 (2010), pp. 121–125.

[5] OpenCV team. *Open Source Computer Vision Library.* 2021. URL: https://github.com/opencv/opencv.

[6] DoCong Thuan. "Evolution of YOLO algorithm and YOLOv5: The state-of-the-art object detection algorithm". In: 2021.

[7] Tzutalin. *labelImg.* 2015. URL: https://github.com/tzutalin/labelImg.