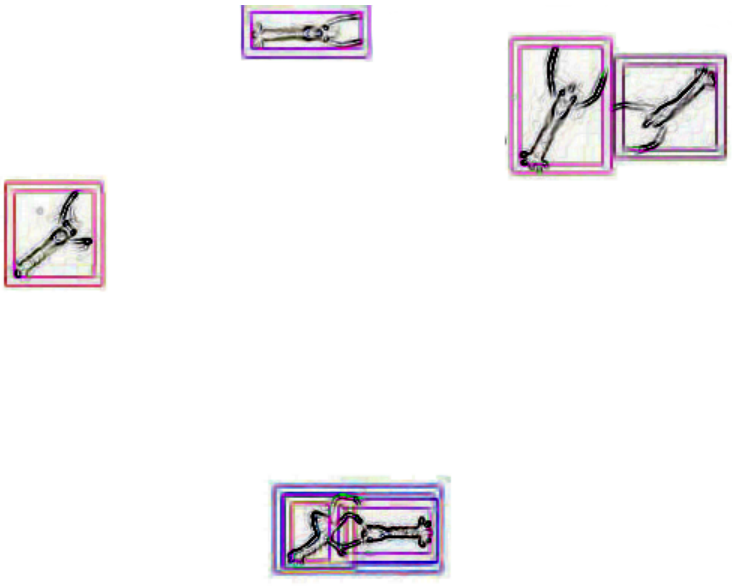
Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Fritz-Olav Myrvang

Interaction detection, tracking and pose estimation in lobster farming

Master's thesis in Computer Science
Supervisor: Frank Lindseth
June 2020



NTNU
Norwegian University of Science and Technology

Fritz-Olav Myrvang

Interaction detection, tracking and pose estimation in lobster farming

Master's thesis in Computer Science
Supervisor: Frank Lindseth
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

The field of lobster farming has remained relatively untouched by machine learning applications for a long time, and as the price and demand of lobsters have steadily increased in recent years, there are opportunities to innovate in farming methods to meet the increasing demand. However processes such as feeding is labour intensive, and state of the art facilities are barely scaleable, where most facilities keep lobsters in separate holding cages. In this thesis we will address parts of a possible implementation of multi-lobster pools, to address the challenges current solutions face. To succeed, one has to reinforce specific traits through selective breeding.

The issue with raising multiple lobsters in shared pools is due to their territorial nature, where individuals will try to establish dominance and ultimately dispose of any possible rival. These interactions happen unpredictably and can last mere fractions of a second. The requirements of human observers to chart individual behaviour are patience and quick reflexes, and the task itself is incredibly time consuming as behaviour has to be tracked for extended periods of time.

We propose a possible solution to automate the task of behavioral tracking, using state of the art computer vision techniques. Applying common architectures such as R-CNN, RetinaNet and YOLO, we explore the possibility of using object detection and pose estimation frameworks to detect lobster interactions. The output of these networks are then fed into the SORT algorithm to track the positions of individual lobsters over extended periods of time. The result is a simple, scalable automated system to serve information to lobster breeders as to which individual lobster is suitable for further breeding.

Preface

This thesis was written by Fritz-Olav Myrvang during the spring of 2020, as the final delivery of a masters programme at the department of computer science (IDI). My motivation for writing this thesis was to perform and experience a from scratch machine learning project, from collecting data, to applying state of the art methods to solve real problems. I have learnt a lot from my mistakes since starting the project, and I know this experience will be very useful in my future endeavours. I sincerely hope my efforts will also be useful for any future projects that will follow this thesis, and someone new will delve new insights from the work presented here. Finally, I would like to thank my supervisor Frank Lindseth for useful tips and guidance during the project, and valuable insights when writing this thesis.

Fritz-Olav Myrvang
June 14th, 2020

Table of Contents

Preface **2**

List of Figures **ii**

List of Tables **iii**

1 Introduction **1**

1.1 Motivation 1

1.2 Goals and research questions 1

1.3 Contributions 1

1.4 Thesis outline 2

2 Background **3**

2.1 Artificial Neural Networks 3

2.2 Convolutional Neural Networks 4

2.3 ResNet 4

2.4 YOLOv3 network 5

2.5 R-CNN networks 6

2.6 RetinaNet 7

2.7 SORT algorithm 7

2.8 Pose estimation 7

2.9 Metrics 8

2.10 Annotation Tools 9

2.11 Detectron2 9

3 Methodology **10**

3.1 Dataset creation 11

3.2 Interaction detection 12

3.3 Keypoint detection 12

3.4 SORT tracking 12

4 Results **13**

4.1 Dataset creation 13

4.2 Object and Interaction detection 13

4.3 Pose estimation 13

4.4 SORT tracking 16

5 Discussion **19**

5.1 Object detection 19

5.2 Pose estimation 20

5.3 Tracking with SORT 20

6 Conclusion **22**

7 Future Work **23**

Bibliography **25**

List of Figures

1 Example of the concept of an artificial neuron and neural network. 3

2 A simple visualization of the parameter optimization problem with 3 parameters. The surface shown is the error given certain parameter values. The goal is to find a point on this surface where the error is as low as possible. 4

3 Representation of a simple convolutional layer computing output features given a kernel and input. 5

4 An overview of a complete convolutional network, from input image to output predictions. 5

5 The residual connection used in ResNet architechtures. 6

6 The YOLOv3 architecture. 6

7 A visualization of the output format of the YOLO network. 7

8 The RetinaNet architechture. 7

9 Faster-RCNN concept. 8

10 Example of person keypoint detections, overlayed on an input image. 9

11 A visual representation of IoU to the left, an example of a precision-recall curve to the right. 9

12 A flowchart of the complete system, with an abstract overview of all components included in the proposed solution. The keypoint network and related output is greyed out as the component is not used for anything in the system as is. 10

13 The keypoint annotations and lobster-interaction annotations. The lobster category and interaction category to the left, and the keypoints to the right. The lobster category is shown as a red bounding box, and the interaction category is shown as a yellow bounding box. The keypoints per lobster was set such that there was one for each claw, one for each joint, one for each eye, and a point for the tail. 11

14 Good interaction and lobster detections. Interactions in red, lobsters in yellow. . . 14

15 Bad lobster and interaction detections. Interactions in red, lobsters in yellow. . . . 14

16 Good pose estimates. 15

17 Bad pose estimates. 15

18 An example of SORT tracking on the output of a R-CNN model using only predicted lobster bounding boxes. Starting from the top, the next video frame is 3 seconds apart from the one directly below it. 17

19 A snapshot of a run in progress. A table of different tracking trajectories, with the number of interaction participations and distance traversed is overlayed in the black box to the top left. 18

20 Concept diagram for the complete lobster network on a single data source to the left, a birds eye view of use with Deepstream to the right. 24

List of Tables

1 Common hyper-parameters across all object detection models. 12

2 Table of the different networks used in the experiment for lobster and interaction detection. 12

3 Hyperparameters used to train the pose estimation model. 12

4 Statistics of the lobster-interaction dataset. 13

5 Statistics for the pose estimation detection dataset. 13

6 Object detection models performance compared. 13

1 Introduction

In this thesis, the use of established visual learning algorithms will be explored to solve challenges in the emerging field of lobster farming, for the purposes of reducing intense labour requirements when tracking and detecting individual interactions of lobster populations in enclosed areas.

1.1 Motivation

Lobster farming has been an emerging field on the cusp of breaking through its pioneering stages in Norway since 2003 [11]. Prices and demand are steadily increasing, and there is untapped potential in exploiting the potential of domestic lobster production.

When considering lobster farming on commercial scales, there are many challenges to face. Studies of the social behaviours of lobsters indicate that large groups of individuals will tend to aggressive territorial behaviour and cannibalism [14]. Especially when the larva are going through the process of molting, they are susceptible to attacks from other lobsters. This problem prevents too many individuals in the same enclosed area, where many lobsters are quickly reduced to a few champions after a short period. Some solutions to this has been growing each individual in separate units, but this prevents large scale operations as the cost per unit quickly becomes prohibitive, considering spatial effectiveness and feeding.

Another approach is to allow for many individuals of lobsters in the same enclosed space, but selectively breed the lobster stock to enhance docile social behaviours. This faces the problem that the lobsters must be observed over long stretches of time, and this is labour-intensive work. This is the setting we will have in mind in the context of this work, were we try to alleviate the labour requirements with automation.

1.2 Goals and research questions

For the work done in this thesis, the goal is to try established methods within the field of object detection and apply it for detecting lobster interactions, where individuals exert aggression on other individuals. In addition, we want to extract pose information from individuals using known keypoint models, as this is possibly useful to determine aggressive and passive behaviour for future selective breeding. Finally, we want to see if tracking individual lobsters is viable as well. There are no publicly available datasets for these types of problems within the domain of lobster farming as is, thus we want to establish datasets for these problems as well.

Below are the research question we try to answer in this thesis:

Question 1: Can a model be used for interaction detection between lobsters?

Question 2: Can a system be constructed to track individual lobsters over time?

Question 3: Can a system be constructed to approximate the aggressiveness of individual lobsters?

Question 4: Can interaction detection be done in real time?

1.3 Contributions

The work in this thesis will contribute to exploring the application of deep learning object detection techniques in the field of large scale lobster farming. All efforts in this thesis will demonstrate the applicability of standard methods for use cases in the field. The datasets and ideas in this thesis can be refined and explored further, as the future of scaling up the capacity of lobster enclosures can potentially have large positive economical consequences.

1.4 Thesis outline

The thesis consists of 6 sections after the introduction: Background, Methodology, Results, Discussion, Conclusion and Future Work.

In the **Background**, we introduce necessary background information on topics such as the different models used, metrics on performance, tools and frameworks utilized.

The **Methodology** section explains the steps taken for the results gathered in this thesis, including dataset creation and training the models.

In the **Results** section, the objective metrics of model performances will be presented, together with some visual examples of the performance.

The **Discussion** section tries to reason about the quality of the results gathered here, and also answer the research questions posed in this thesis.

Conclusion contains the summary of the results and discussion, stating if the questions and goals have been achieved.

Lastly, the section **Future Work** tries to look beyond the scope of this thesis, and describes steps that can be taken in the near future to expand upon the work presented here.

2 Background

In this section we will present the necessary background information related to the work done in this thesis, from explaining the traits of each model used, and more general information about what types of techniques related to measuring performance is used and the fundamentals of the different networks that appear in this thesis.

2.1 Artificial Neural Networks

Inspired by the biological neuron, the basic unit of the artificial neural network is conceptually very similar to it's biological counterpart, as seen in figure 1. The artificial neuron can be seen as having a number of real numbered inputs, where some function is performed on the inputs to produce a output, either some linear or nonlinear combination of the inputs. Individually, these components are limited in what computation they are able to perform, but organize them into different structures, such as in figure 1 you get an example of a fully connected network, as each neuron in a single layer is connected to every neuron in the layer before, you can get more complex behaviour. An example many use as an introductory exercise to understand artificial neural networks, is to create a network able to discern between different 28x28 size images of different numbers between 0 and 9 [13]. A very important element of these networks is to train and adjust the parameters of the networks, such that it more accurately achieves it's intended purpose. Under normal circumstances, the parameters of any new network is initially set to random numbers between some small interval, often in the range of [0, 1]. The chance that you get a usable network after initialization is exceedingly low, as the networks used today usually has thousands, if not millions of parameters to tweak during training.

Treating the training of a network as a optimization problem, one can view each parameter as a single dimension in a multidimensional graph, where we try to find the max or min of some function approximating how far off the output is from the desired value, or alternatively, the "wrongness" of the network outputs. A simple example of how to visualize a graph of such an approximation across 3 parameters is shown in figure 2. The idea is to compute the gradient of the surface, and nudge the parameters such that the output is closer to desired values. A common approach for reducing the wrongness of networks is called the Stochastic Gradient Descent [19], where the idea is to calculate how much each parameter contributes to the error, accumulate this error over a batch of inputs, and adjust the parameter values proportionally to this computed value.

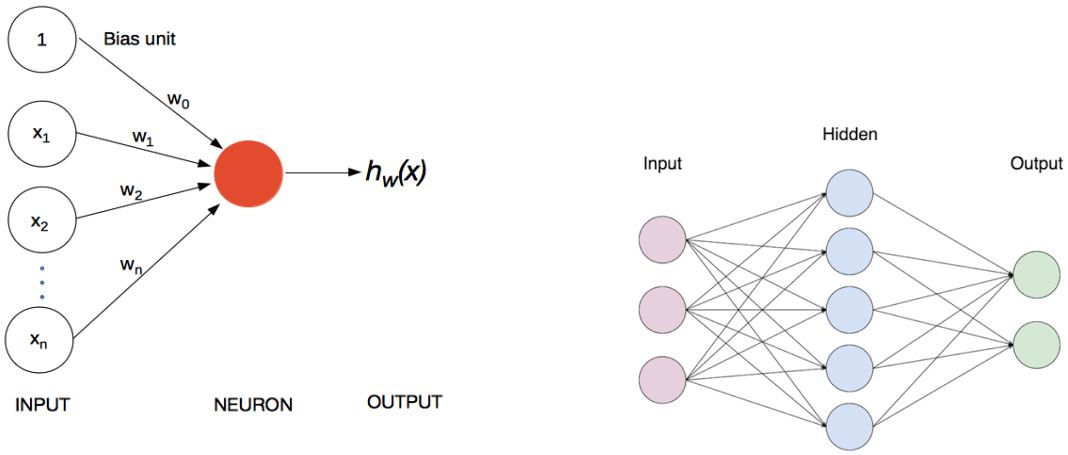


Figure 1: Example of the concept of an artificial neuron and neural network.

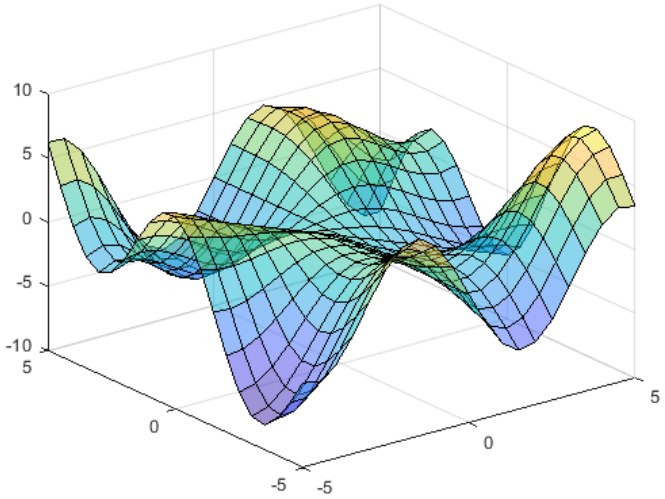


Figure 2: A simple visualization of the parameter optimization problem with 3 parameters. The surface shown is the error given certain parameter values. The goal is to find a point on this surface where the error is as low as possible.

2.2 Convolutional Neural Networks

One problem with earlier fully connected networks such as the ones seen in figure 1, is that it becomes intractable and memory intensive for images with higher resolutions, as increasing the dimensions causes a quadratic increase in the number of pixels. This in turn will lead to at least a quadratic increase in the number of parameters, meaning the problem scales poorly on larger resolution images. The Convolutional Neural Network (CNN) make do with much less parameters, sharing them more across the input [12].

In addition, the fully connected networks do not exploit the very hierarchical nature of image data, and is generally poorly regularized, becoming prone to over-fitting. A feature of the CNN construction is that it is translation invariant, meaning detecting features in the input image is less dependent on the features orientation. It makes no difference if the object is rotated or flipped, the CNN should find the features all the same. This property makes it more difficult to overfit during training, as the network generalizes in favor of memorizing.

In figure 3 we see a simple visualization of a single kernel performing a dot product on a subsection of the input, where each square of the kernel represents a numbered parameter to be multiplied with the corresponding number represented by a square of the input. Sum all the products together, you get the red sum in the resulting output map. This kernel is then applied to all pixels in the input, and produces an output of equal size of the inputs, only "convoluted". In practical networks, for each convolutional layer, you find several kernels, often named "filters", each producing a feature map. Each filter can be thought of as looking for a particular pattern in the input, and the output is a map of where to find the particular pattern.

In figure 4 we see a simple overview of the components of a convolutional network, including commonly used pooling layers, a useful trick to increase translation invariance, and reducing dimensionality by sub-sampling the feature maps.

2.3 ResNet

To achieve better CNN performance, researchers explored different methods to modify and design networks so that they could handle more complicated problems, and solve existing ones more efficiently. One way to handle increasing problem complexity was increasing the network depth.

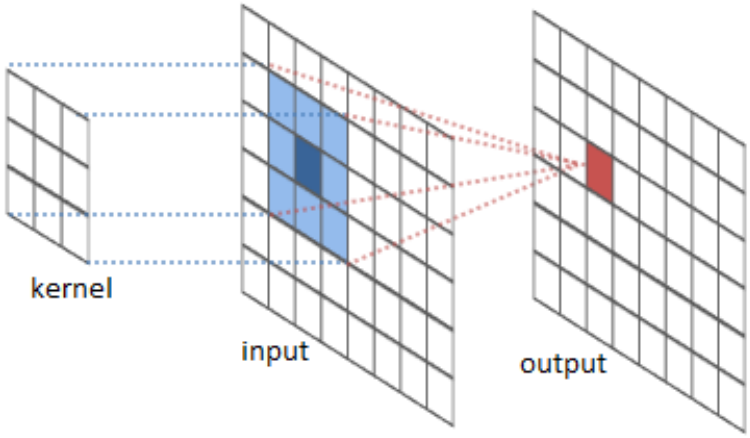


Figure 3: Representation of a simple convolutional layer computing output features given a kernel and input.

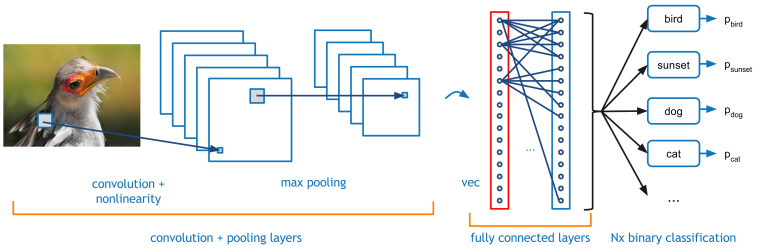


Figure 4: An overview of a complete convolutional network, from input image to output predictions.

However, this only worked up to a point, where adding more layers would actually degrade performance due to the vanishing gradient problem, where early layers would hardly adjust their weights during training [7]. One proposed solution to this was to add residual connections, or shortcuts on "plain" architectures, to allow the error gradient to diminish less when propagating backwards through the network, see figure 5. Compared to similar networks without residual connections, the residual networks saw greater performance.

Since the paper was published, the CNN backbone of choice has often been a type of ResNet architecture. It is often combined with a feauture pyramid network [9], where features are combined on different scales with the help of up and downsampling, see figure 8. More often than not you see a 50 or 101 layer ResNet network as a backbone, with a feature pyramid network on top, providing different networks good baseline performance in general and when features need to be found at many different scales.

2.4 YOLOv3 network

The YOLO object detection system is a network configuration designed to do detections in real-time. As other approaches are often more accurate, the YOLO approach trades pinpoint accuracy for detection speed, combining bounding box prediction and classification into a single network, with all outputs being combined into a single multidimensional output, in contrast to other approaches, such as R-CNN networks were the architecture consists of several distinct parts with separate outputs. See figure 7 for an overview of the structure of the YOLO output. When first published, the network had tremendous inference speed compared to its contemporaries, achieving real time detection rates up to 45fps. Since the first publication of the original YOLO network, improvements has been made in YOLOv2 and YOLOv3, including introducing residual connections in between network layers, and introducing upsampling layers, similar to ResNet, to better detect at different scales of the input image. An overview of the YOLOv3 architechture can be

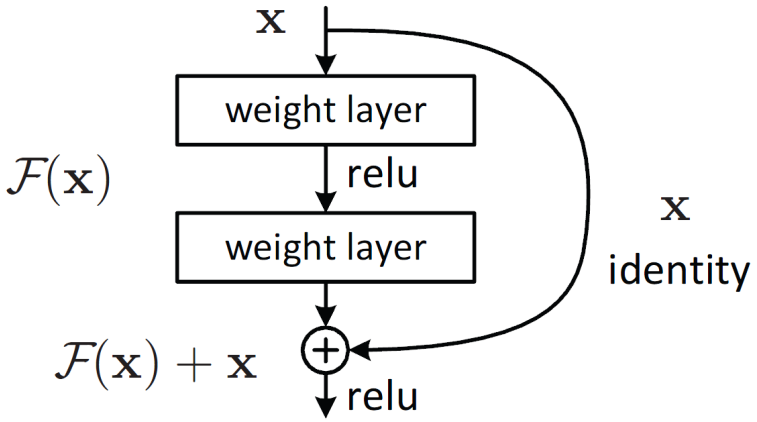


Figure 5: The residual connection used in ResNet architechtures.

seen in figure 6 [15].

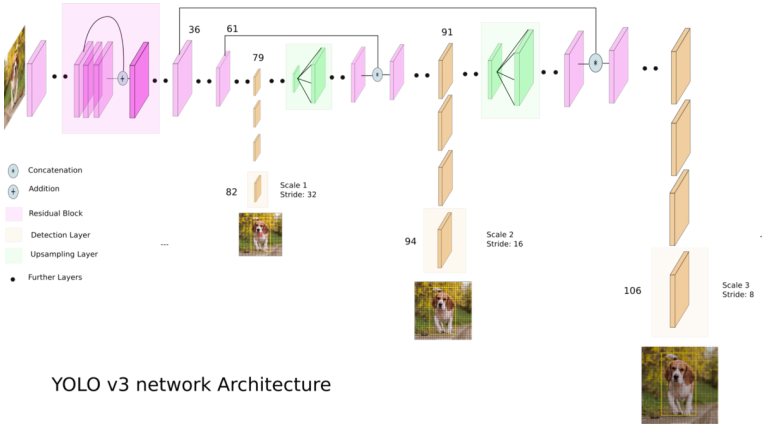


Figure 6: The YOLOv3 architecture.

2.5 R-CNN networks

The R-CNN network family is a construction meant to combat the problem of object detection, finding where in the input image objects are, then classifying the object found. The original R-CNN network [5] solved this by having a fixed algorithm search the input for possible regions of interest, then classifying the regions with a CNN structure, similar to figure 4. Improvements were made, as Fast R-CNN was proposed [4], improving the test and training time with instead of activating the network on each of the region proposals, a common feature map was computed once on the whole image, saving time by computing features once for the entire image, instead of one time for each region proposal. The last iteration of purely object detecting R-CNN-type networks, Faster R-CNN, replaced the region proposal search algorithm with a CNN, jointly training the region proposal network with the rest of the network [16]. This made the model much easier to train, and further reduced test and training times, as the main bottleneck of R-CNN and Fast-RCNN was the region proposal algorithm Selective Search. Despite the convolutional region proposal network, even sharing parameters with the rest of the network, the R-CNN approach is still two staged, as the network has a separate region proposal step. The idea of the Faster R-CNN structure can be seen in figure 9. Mask R-CNN is the latest iteration of the R-CNN family of networks, with support for mask output, where bounding boxes and instance masks are predicted, yielding accurate segmentation, however still remaining a solid framework for object detection [6].

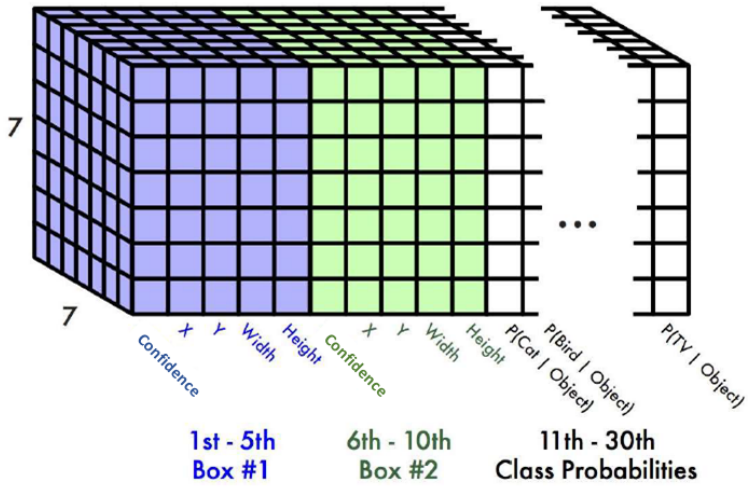


Figure 7: A visualization of the output format of the YOLO network.

2.6 RetinaNet

The RetinaNet architechture was introduced as a way to solve the problem of training single stage detectors, such as YOLO, where YOLO introduces strong spatial restrictions inherent in it's construction, Retinanet attempts to introduce a new way of computing loss, namely "focal loss", by downweighting well-classified examples. The reason to do this is due to an imbalance in foreground vs background classes, as the predominance of background pixels in the training data in any dataset would prove this kind of single stage detectors difficult to train. Combined with a feature pyramid network over a ResNet backbone, the RetinaNet approach yielded good results compared to it's contemporaries when the approach was first published. See figure 8 for a graphical overview of the architecture.

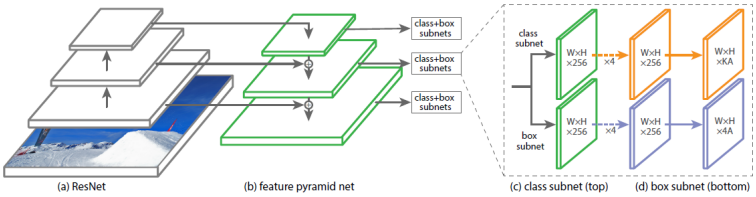


Figure 8: The RetinaNet architechture.

2.7 SORT algorithm

The SORT algorithm [1], short for Simple Online and Realtime Tracking, is a pragmatic approach to the problem of multi object tracking. Combining the output of a CNN trained to predict bounding boxes, a Kalman Filter to track and predict bounding box positions and change, and the efficient Hungarian algorithm to assign Kalman predictions to the CNN predictions, the SORT method is a fast and functional approach to the multi-object-tracking problem, beating many of its contemporary approaches in tracking speed and accuracy.

2.8 Pose estimation

It is useful to not only find the bounding box of objects within an image, but also it's features, such as hands, head, legs and so on for pose estimation. One can modify the Mask R-CNN network to predict single-pixel masks representing keypoints within the bounds of detected objects. See figure 10 for a visual representation of keypoints overlayed on an input image. Pose information is

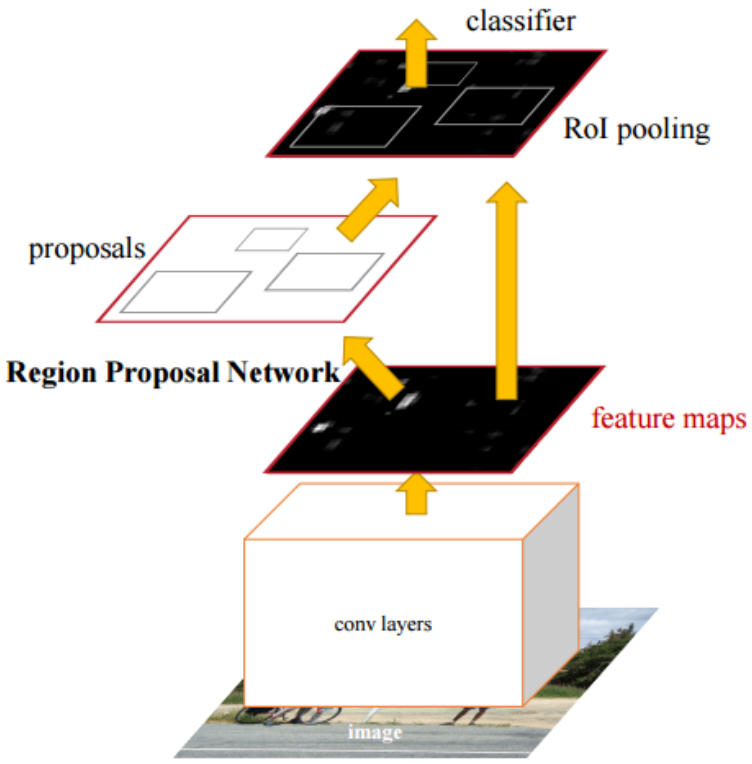


Figure 9: Faster-RCNN concept.

useful, as normal object detection methods reveal information about positioning, but little in terms of what actions the objects are performing. Therefore reasoning about object pose information, especially applied to humans and animals, is useful for predicting behaviour.

2.9 Metrics

To compare the performance of different networks, it is important to have metrics. For the task of object detection, a common metric used is AP, average precision. To compute the AP, one needs to calculate the precision and recall. In information retrieval systems and object detection, precision is defined as the share of true positives compared to the sum of true positives and false positives, meaning the number of relevant results detected amongst proposals. Recall is the share of true positives compared to the sum of true positives and false negatives, meaning how many relevant examples are found amongst all relevant examples. The values of the precision and recall will vary with how many images is used in the computation and how many annotations there are and how many annotations are correctly predicted per image. Plotting the precision on the y-axis, and the recall on the x-axis, and varying the number of images included in the precision-recall plot, you get a curve similar to figure 11. To compute AP for the model performance, you approximate the area under the precision-recall plot. Do this over different object classes and computing the average, you get mean-AP.

When it comes to object detection, what constitutes as a positive detection depends on correctly classifying the object instance, and predicting a bounding box that is "close enough" to the ground-truth. The IoU, or Intersection over Union, is a measure of how well predicted bounding boxes match the ground truth. See figure 11 for a visual representation. In practice, AP is measured across different IoU thresholds, where 0.5 is common.

Figure 10: Example of person keypoint detections, overlayed on an input image.

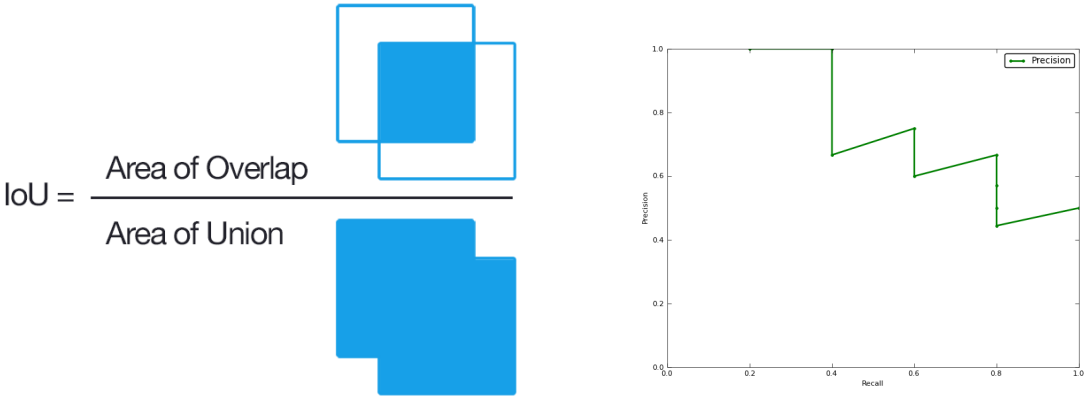


Figure 11: A visual representation of IoU to the left, an example of a precision-recall curve to the right.

2.10 Annotation Tools

In modern object detection tasks that falls outside of the standard publically available datasets, it is necessary to create your own data. To do this as efficiently as possible, many open sourced projects exists to help provide graphical front ends for image annotation, and dataset management. Popular ones are the Computer Vision Annotation Tool (CVAT), and COCO-annotator. They both provide tools for dataset creation, but have slightly different focus. Where CVAT provides support for many different datasets, not every format has full support, as the project is in perpetual development, as is the nature of many open sourced projects [18]. COCO-annotator naturally favors the COCO format [10], and fully supports image segmentation and keypoints [2].

2.11 Detectron2

Detectron2 is an open source project maintained by the Facebook AI Research (FAIR) group of Facebook, and is a software system including many state of the art deep learning algorithms adapted to object detection, pose estimation, instance segmentation, panoptic segmentation and other applications [20].

3 Methodology

In this section the method of creating the interaction detection model, including the creation of datasets, tracking and model training will be detailed.

A brief overview of the work that has been done:

1. Created Lobster-Interaction dataset.
2. Trained Yolo, RetinaNet, R-CNN networks.
3. Created pose estimation dataset.
4. Trained keypoint R-CNN on dataset.

For an overview of the proposed system with all parts assembled, see figure 12.

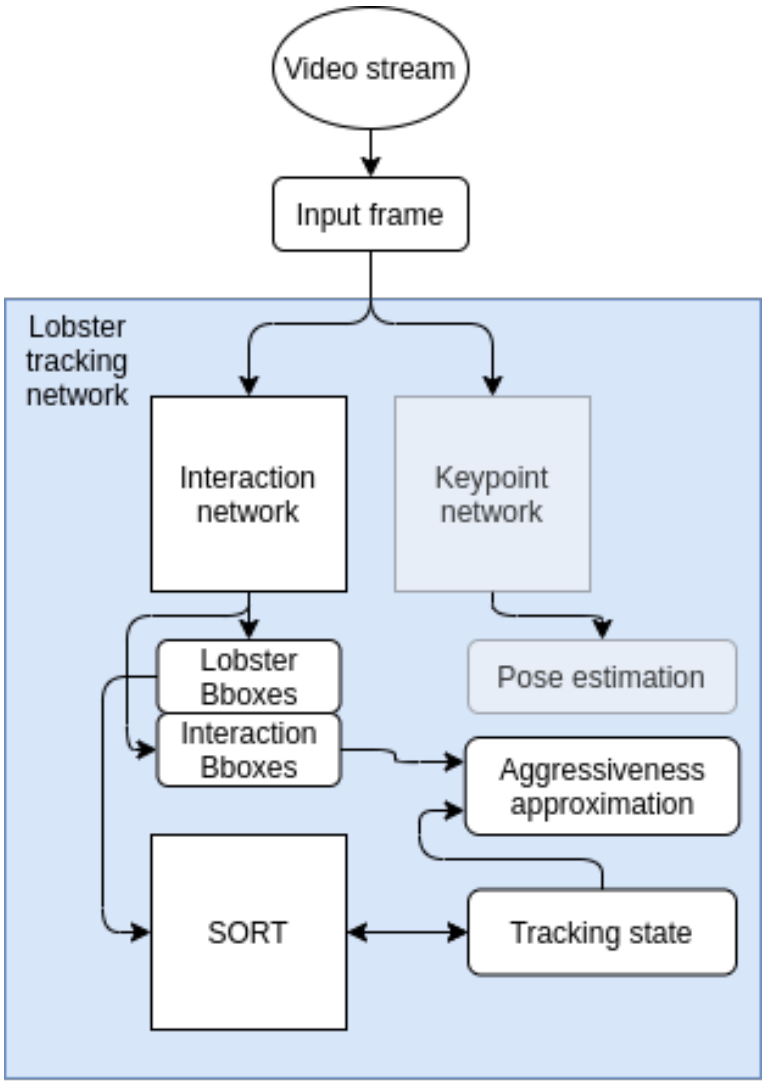


Figure 12: A flowchart of the complete system, with an abstract overview of all components included in the proposed solution. The keypoint network and related output is greyed out as the component is not used for anything in the system as is.

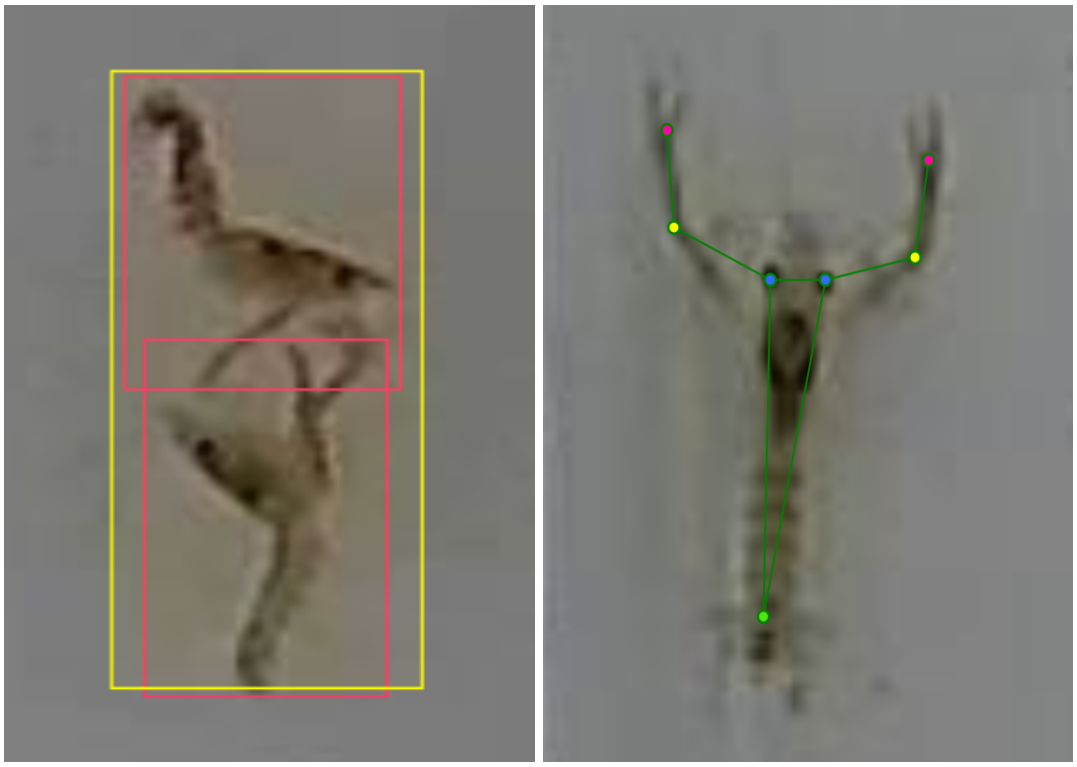


Figure 13: The keypoint annotations and lobster-interaction annotations. The lobster category and interaction category to the left, and the keypoints to the right. The lobster category is shown as a red bounding box, and the interaction category is shown as a yellow bounding box. The keypoints per lobster was set such that there was one for each claw, one for each joint, one for each eye, and a point for the tail.

3.1 Dataset creation

To create the object detection dataset, video recordings in 20 minute segments of lobster enclosures in 1920x1080 resolution was provided. A selection of the raw video was edited into smaller segments of about 1 minute, to distribute the workload of annotation into smaller chunks. To complete the annotation of the selected 1 minute segments, the segments were annotated with the tools CVAT and COCO-Annotator. Some videos were pre-processed with an algorithm from an earlier project, with the intention of estimating bounding boxes for the different object classes, allowing the annotator to adjust rough estimates, instead of starting from scratch.

For this project, two datasets were created. One for pose estimation, and one for lobster and interaction detection. The classes to be annotated is shown in figure 13. The particular keypoints annotated can also be seen in figure 13. The datasets labels are converted into the COCO and YOLO format, to be used during training. See table 4 for an overview of the object detection dataset. Finally, the object detection dataset was split into a 70-30 ratio between training and testing purposes.

Annotating bounding boxes on individual lobsters was relatively straight forward, as usually it is easy to distinguish the confines of individual lobsters pixels. However, defining lobster interactions by the information provided by any one single frames could be more vague than any annotator would like, as there was no clear defining characteristic of an aggressive interaction, as it usually were hard to read the lobsters intent. For every instance of interaction labels, the annotators discretion was used, and due to the nature of the source data being minute long videos, the annotator usually had access to preceding frames of any suspected interaction.

3.2 Interaction detection

The idea was to achieve real-time inference. Shallow, single stage detectors were initially chosen for the effort of object detection, such as Retinanet and YOLO. However Mask R-CNN and Faster R-CNN architechtures has been included as well as baselines. All feature extractors used as backbones in the respective models have been pre-trained on the COCO dataset for about 37 epochs [20]. A range of different architectures were picked to get reasonable comparisons, see table 2 for an overview. The models were trained on a DGX cluster, with a Tesla V100-SXM3-32GB as the GPU against the Lobster-Interaction dataset. For an overview of the used hyper-parameters across all object detection models see table 1. All models except YOLO was implemented in the Detectron2 framework[20]. For YOLO specifically an open source project was adepted to fit this project [8].

Hyperparameter	Value
Iterations	3000
Learning rate	0.01
Images per batch	16
Momentum	0.9

Table 1: Common hyper-parameters across all object detection models.

Model	Feature Extractor
faster_rcnn	R_50_FPN
retinanet	R_50_FPN
mask_rcnn	R_50_FPN
faster_rcnn	R_101_FPN
retinanet	R_101_FPN
mask_rcnn	R_101_FPN
YOLOv3	Darknet-53

Table 2: Table of the different networks used in the experiment for lobster and interaction detection.

3.3 Keypoint detection

For training on the keypoint dataset, a Mask-RCNN network adjusted to detect key-points was chosen, pre-trained on the COCO dataset. See table 3 for an overview of hyper-parameters used to train the model used in this thesis.

Model	keypoint_rcnn_R_50_FPN
Learning rate	0.001
Iterations	900
Batch size per image	512
Images per batch	2

Table 3: Hyperparameters used to train the pose estimation model.

3.4 SORT tracking

For tracking with the SORT algorithm, there was no training involved. Output from one of the best performing models was provided and SORT was run on select test videos. The tracking information was then combined with the interaction bounding box output, and distance traversed, for the purposes of estimating which lobsters possess aggressive traits. The heuristic being that lobsters participating in many interactions, more likely than not are aggressors, and conversely individuals with a low distance and interaction score are probably not starting fights.

4 Results

In this section the results of the work done will be presented, including details of the datasets created, and the performance of the models used for object detection and pose estimation.

4.1 Dataset creation

See table 4 for reference for the lobster-interaction detection dataset, and table 5 for the pose estimation dataset.

Images	3148
Annotations	117380
Categories	2
Annotations per category	
Lobster	70013
Interaction	3378
Images per Category	
Lobster	3148
Interaction	2842

Table 4: Statistics of the lobster-interaction dataset.

Images	224
Annotations	4679
Categories	1
Keypoints per annotation	7

Table 5: Statistics for the pose estimation detection dataset.

4.2 Object and Interaction detection

See table 6 for the performance of each object detection model. See figure 15 and figure 14 for examples of performance on a test video.

Model	Lobster AP	Interaction AP	Inference FPS
mask_rcnn_50	79.42	84.29	13.65
faster_rcnn_50	78.69	83.15	13.68
retinanet_50	78.74	80.98	13.88
mask_rcnn_101	81.01	**85.39**	11.34
faster_rcnn_101	**81.17**	81.73	11.49
retinanet_101	79.87	82.626	11.82
YOLOv3	75.14	78.54	**19.93**

Table 6: Object detection models performance compared.

4.3 Pose estimation

See figures 17 and figure 16 for a sample of the test results after training the keypoint R-CNN model.

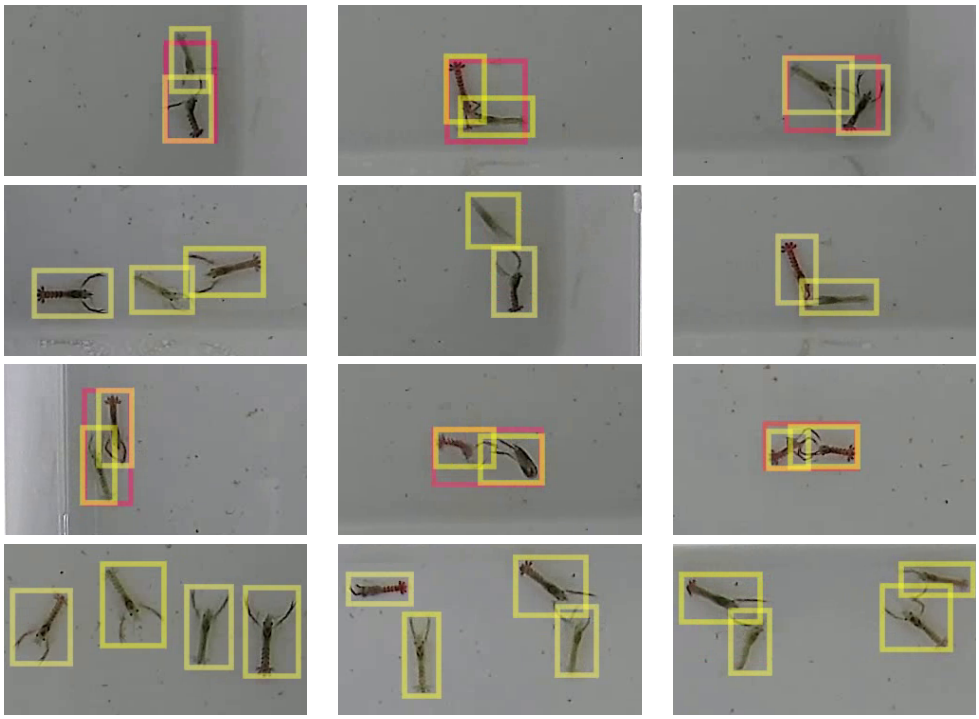


Figure 14: Good interaction and lobster detections. Interactions in red, lobsters in yellow.

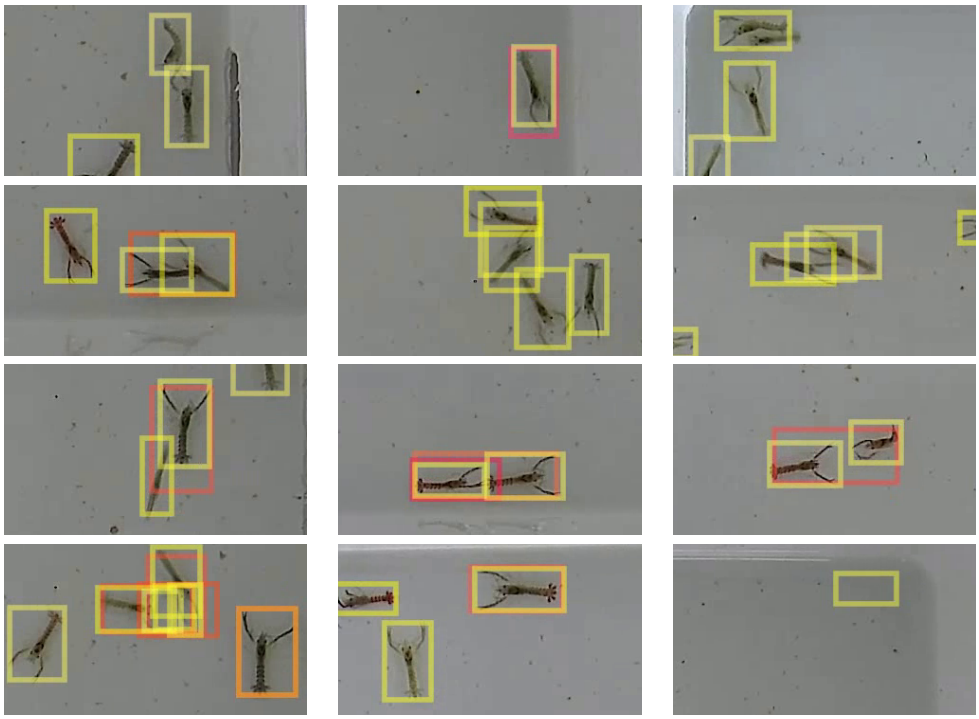


Figure 15: Bad lobster and interaction detections. Interactions in red, lobsters in yellow.

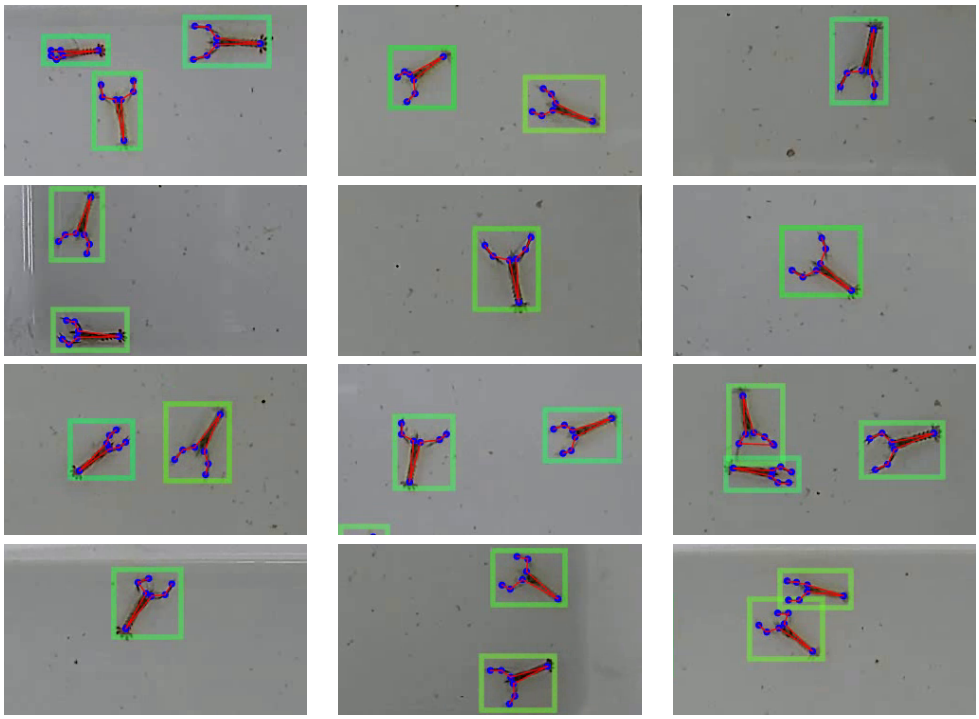


Figure 16: Good pose estimates.

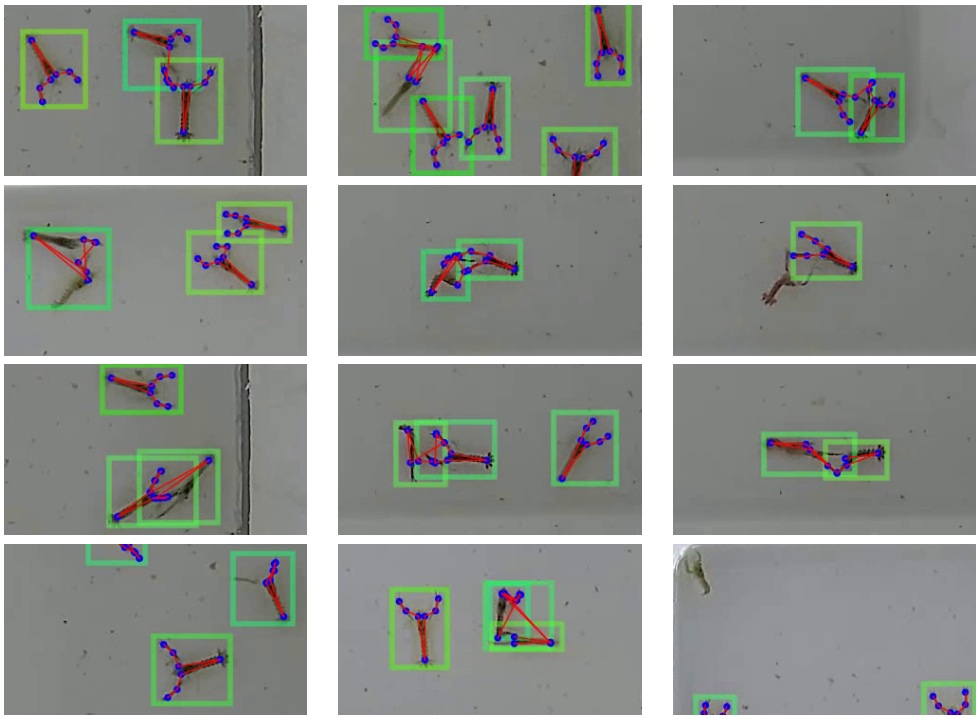


Figure 17: Bad pose estimates.

4.4 SORT tracking

See figure 18 for an example tracking sequence with the SORT algorithm on R-CNN output. For an attempt at approximating aggressiveness combining all except keypoint information, see figure 19 for a sample of the running system as shown in figure 12.

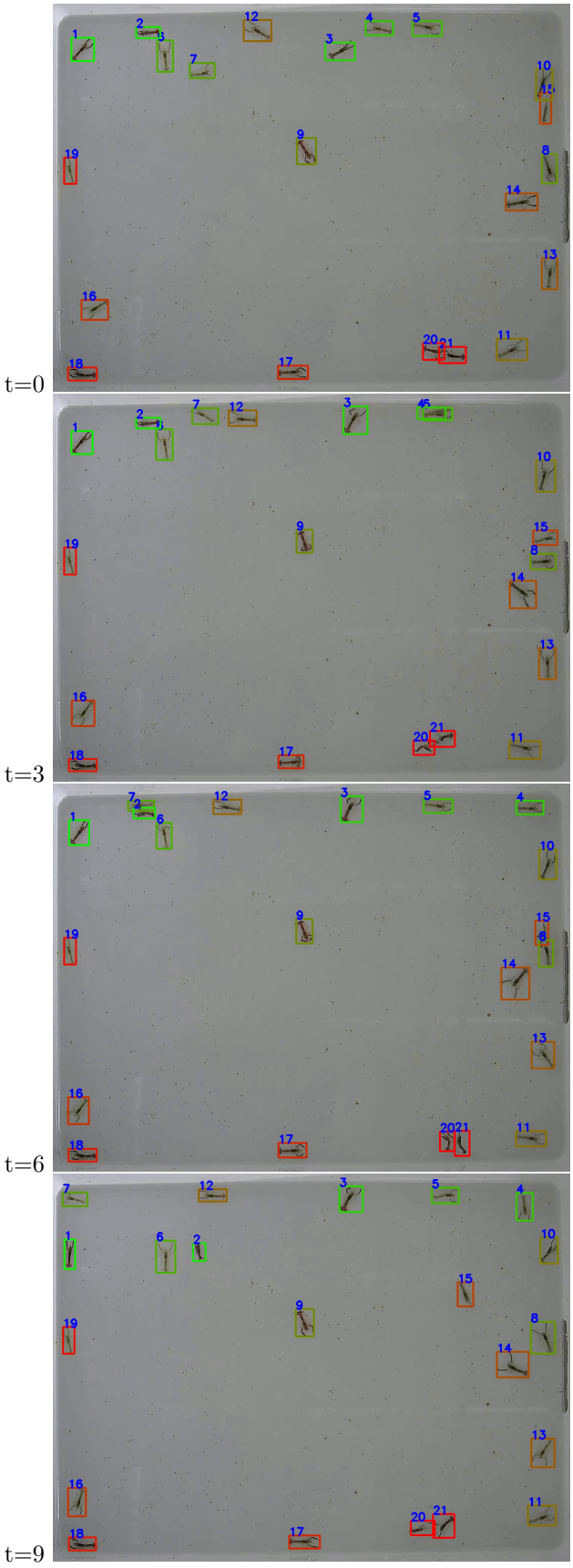


Figure 18: An example of SORT tracking on the output of a R-CNN model using only predicted lobster bounding boxes. Starting from the top, the next video frame is 3 seconds apart from the one directly below it.

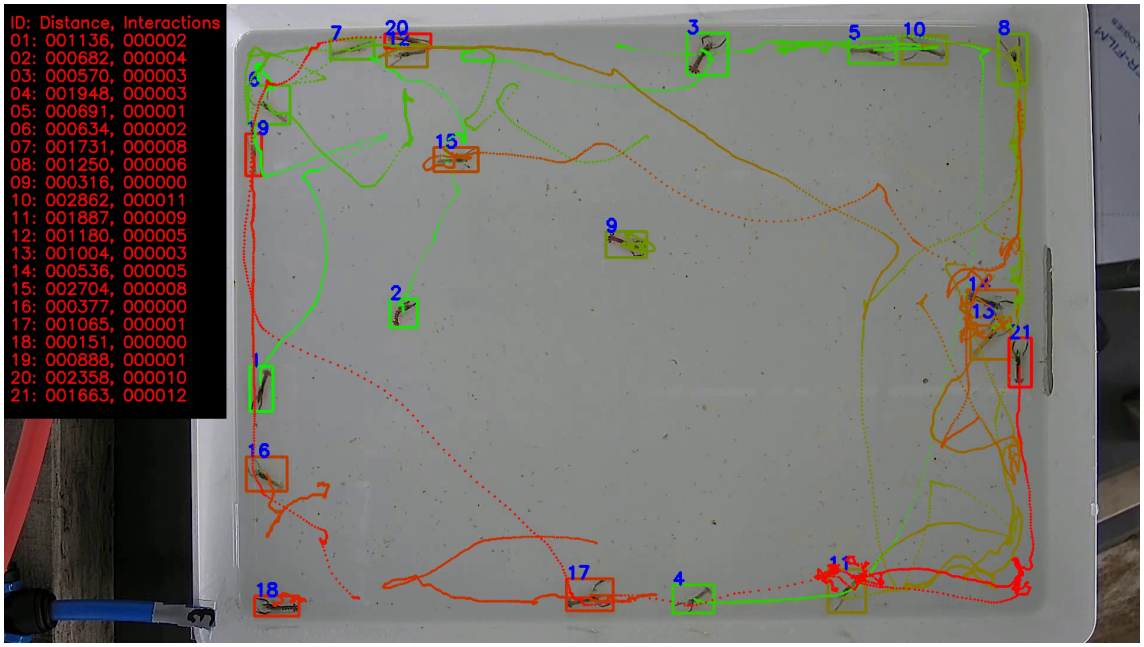


Figure 19: A snapshot of a run in progress. A table of different tracking trajectories, with the number of interaction participations and distance traversed is overlayed in the black box to the top left.

5 Discussion

In this section the results of the work done in this thesis will be discussed, for example if the approach was successful in detecting interactions, if the tracking approach was successful and if the keypoint model was successful in pose estimation of lobster individuals. We will consider the original research questions from section 1.2 during the evaluation, and try to shine light on the good and the bad aspects of the results and methodology.

5.1 Object detection

One important point about the datasets created is that they are relatively small compared to popular established benchmark datasets such as COCO and ImageNet, where COCO2017 has 123,287 images [10], and ImageNet with 14,197,122 images [17]. Each of the 80 and 27 categories of object is relatively evenly split across the images. Compare this to our distribution of object categories, we see that there is a problem of balance. If we look at table 4, the important interaction class, contributes less than 5% of the total annotations. In addition, the interaction class is not strictly an easily defined object. The category includes two or more interacting lobsters, which can be tricky to distinguish without access to previous or future frames. Compounded with the small size of the dataset, it is reasonable to assume that better quality detections can be made with more examples of interactions, as there can be many types, compared to the features of a single lobster.

A second comparison to the benchmark COCO and ImageNet datasets, is the relative density of objects in the lobster dataset. Where usually the per image frequency of objects lies around 5 objects or less in COCO and ImageNet datasets, in the lobster-interaction and pose estimation datasets it is not unusual to see at least 15-20 individual lobsters in any single frame, with possible interactions on top. As is usually the case with dense datasets, there tends to be an overwhelming amount of easily classified examples, with a few other classes sprinkled in between. As discussed earlier, this is definitely the case with our custom object detection dataset. Tons of easily classifiable lobsters, much fewer, more tricky interactions. Considering this, there definitely should have been experiments with training networks solely on the interaction annotations, so as to compare single class predictors against the dual class predictors tested in this thesis.

Another important point to discuss is the subjective quality of the detections of the object detection models. We have objective metrics in average precision, however this fails to capture the perceived quality and actual usefulness of the networks. The test set for the object detection dataset contains about 900 images. Of all the annotations on those 900 images, under 5% of those are interactions. It is important to acknowledge that the AP of the models presented here should be taken with a grain of salt for the aforementioned reasons. It can be shown through examples the quality of the detections of the more successful networks are passable, where the most egregious aggressive interactions most often are detected, while more fleeting encounters remain undetected. In addition, seemingly harmless encounters can sometimes also be detected. Considering research question 1 in section 1.2, the results in table 6 and the test videos sampled in figures 15 and 14, and the quality of the test videos made performing detections on unseen data, there definitely is promise in this method of detecting social interactions between lobsters. Even though there definitely is plenty of mislabeling by the different models, it more often than not clearly responds to clear cut signs of aggression. Even though for the annotator it was tricky to discern hostile interactions from harmless ones by only information contained in any single frame of the input, the networks however are able to detect the interactions more often than not. It is able to discern singular lobsters from multiple lobsters interacting, which is definitely promising. An alternative would be to use some kind of recurrent neural network where more than one frame would be presented to the network so that it has temporal information about interaction as well, or expand the CNN models to input several frames of data at once . This however would complicate the dataset creation considerably, as one would keep sequence information of all images. This object detection approach was chosen as it is simple to create and expand the dataset, and stock object detection models are readily available to solve these types of problems.

About the real time performance of the models, the limitation of the performed experiments was

running the models on a single GPU, on images with the full native resolution 1920x1080, and a naive python implementation for inference to simulate online network behaviour. There are various things that could be tried, such as involving more GPUS, downsampling the input images or trying to optimize the implementation of the detection software. As it stands, the model performances can be considered as borderline too slow. As the source videos were recorded at 30 fps, there are sequences of interactions that is happening just to fast for the 13 fps models to catch the interesting parts of the action. However the YOLO performance of 20 frames per second is the best result so far when considering research question 4 in section 1.2. More effort has to be made to insert the networks into a true online setting, but it seems certainly possible to match the theorized rate of 30 fps input rate, if the lobster tracking system can utilize more computing resources.

When comparing the objective metrics of each model, all models were viable for this task. However some needs more tweaking of hyper-parameters than others. Considering most of the egregious examples of mislabeling were done by the RetinaNets and the deeper 101 layer R-CNN networks, despite features such as focal loss, feature pyramid networks and residual connections. This could very well be an issue with the hyperparameters being wrong for this combination of network and dataset, looking at examples in figure 15, setting a different non-max-suppression threshold could be one source of the issues. Another potential problem is by the nature of the dataset creation process, consecutive video frames are relatively similar. As the complete set of labeled data is shuffled and distributed between test sets and training sets, one can imagine that there are many pairs of very similar images split between the two, making any model training on the datasets prone to overfitting, and producing misleading metrics. Regardless, the seemingly most stable networks was the YOLO network, the metrics in table 6 notwithstanding. A close second was the different 50 layer R-CNN networks based on the same reasoning. With a very humble dataset, pretrained on the COCO dataset, the R-CNN and YOLO networks were very much able to yield stable predictions on unseen test videos, as most of the examples in figure 14 is produced by YOLO, Faster or Mask R-CNN.

5.2 Pose estimation

Concerning pose estimation, the keypoint model used in this work has performed admirably, with the very small dataset as shown in table 5. As there was no more time to add more images to the pose estimation dataset, by the creators discretion, the dataset was not split into a test and training set, as there was concern that the dataset was just to small to get reasonable evaluation. It is the same problem as mentioned earlier, as the total training set is already very small compared to larger benchmark datasets. However, as can be shown in examples, the keypoint R-CNN model looks to be performing very well considering the limited dataset. Individual, isolated lobsters keypoints is detected with minimal errors, but the problems arise when the lobsters interact or occlude. Considering research question 3 in proposed in section 1.2, the original idea was to feed pose data into another model design to assign individual aggressiveness onto lobsters. Unfortunately, the design of this type of network fell outside the scope of this thesis. Despite this, the prototype pose estimation network is able to decently generalize on unseen data based on the test video samples in figure 16.

5.3 Tracking with SORT

With the SORT algorithm, using the Kalman Filter to predict the trajectories of lobsters, and the Hungarian algorithm to assign detected lobsters to Kalman predictions, typical results are sampled in figure 18. Judging the subjective quality of the test videos seen, the method seems undeniably robust. The tracking algorithm assigns ID's to individual lobsters, and barring egregious occlusions and activity, the tracking can remain for extended periods of time. By research question 2 posited in section 1.2, it shows promise in being used for future tracking experiments. The only problem is by the nature of the object detection dataset we do not have any ground truth to do evaluation of performance. Ideally other tracking approaches should be tested against a labeled test set. SORT was chosen for its simplicity and speed, however it is not considered state of the art.

Despite lack of testing data, testing the complete system as seen in figure 12, yielded promising results where instead of a dedicated network, we approximate individual aggressiveness by counting interactions lobsters participate in, as seen in figure 19. There has not been time to properly evaluate this approach, however by all accounts it is worth testing. Considering research question 3 from section 1.2 again, the approach of approximating aggressiveness by penalizing interaction participation and travel distance could very well be a useful heuristic. It needs to be explored further to determine its potential.

6 Conclusion

In conclusion, there is definitely something useful that emerged from the efforts in this thesis. Interactions between individual lobsters are possible to detect with standard object detection methods, and with the underlying motivation of this effort in mind as described in section 1.1, tracking aggressive and docile lobsters seems very possible on larger scales as the experimental results presented in section 4 shows. The combination of SORT and a reliable object detection model, online, long term tracking of behaviour does not seem that far off. There is more work to be done with expanding the datasets created, and shifting from an experimental, qualitative approach, to becoming more data driven, and focusing more on metrics. This can be achieved by adding to the datasets already created, and more exploration in the details of the possible networks.

7 Future Work

In the immediate future, expanding the datasets to be confident in the ability to precisely evaluate the networks performance is important. And as confidence in the evaluation metrics is achieved, one can try to fine tune different models, to find the one that is most suitable for the task of detecting interactions and individual lobsters. In the vein of increasing datasets, if different tracking methods are tried in the future, it is important to have a tracking dataset for evaluation of different approaches. Having datasets for all the different facets of the challenges in detecting lobsters and interactions, including detecting aggressive and docile individuals and tracking performance, is useful for determining the relative success of the complete system.

When it comes to detecting aggressiveness, more effort has to be made to invent some kind of measure of aggressiveness and docility on an individual level. Breeding lobsters could take some time, therefore it would be useful to have a precise measure or approximation of the desired and undesired traits. In this thesis we used a heuristic where penalizing participants of interactions, using the bounding box information to count how many interactions any individual has participated in. Obviously this approach has weaknesses, but should be tested and evaluated against a labeled dataset.

For an idea of the complete network to do lobster breeding on a large scale, one has to handle multiple data streams from many enclosures as the tracking must be done in real time to be useful for deploying into the real world. For a single input source, one could imagine something like the left diagram shown in figure 20, where one instance of the system is monitoring a single unit. In this thesis there has been made no effort to experiment how the system would work for a larger scale scenario where multiple data streams must be managed simultaneously, and probably over a network of some kind. There exists commercial frameworks to manage such systems, such as Nvidia Deepstream [3], and considering the task if managing such a complicated system, it is not unreasonable to experimenting with the Deepstream SDK to test the ideas applied in this thesis could perform well on larger scales, see the right diagram in figure 20 for a conceptual visual of the larger system.

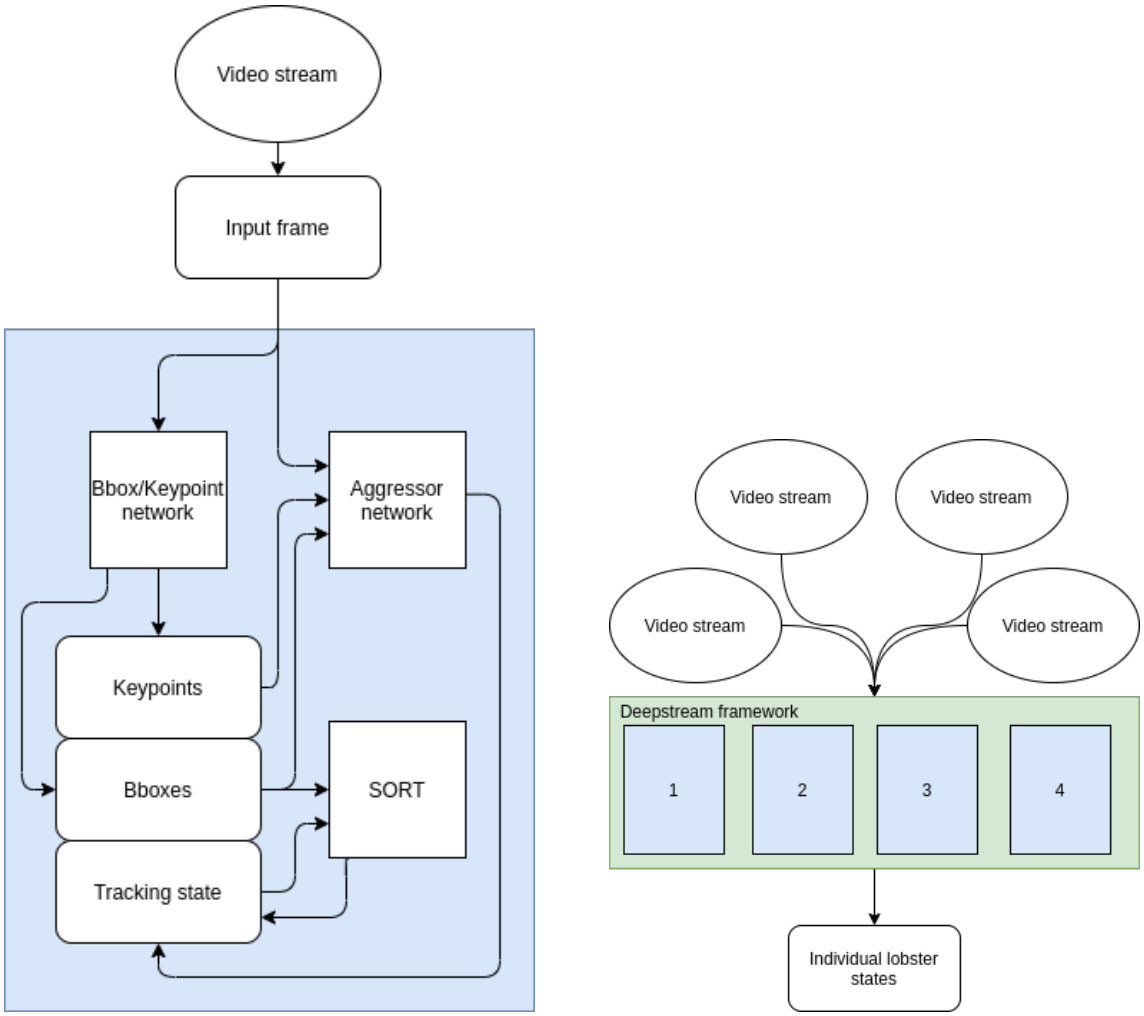


Figure 20: Concept diagram for the complete lobster network on a single data source to the left, a birds eye view of use with Deepstream to the right.

Bibliography

[1] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 3464–3468, 2016.

[2] Justin Brooks. COCO Annotator. `https://github.com/jsbroks/coco-annotator/`, 2019.

[3] NVIDIA Corporation. Deepstream SDK. `https://developer.nvidia.com/deepstream-sdk`, 2020.

[4] Ross Girshick. Fast R-CNN. *arXiv e-prints*, page arXiv:1504.08083, April 2015.

[5] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *arXiv e-prints*, page arXiv:1311.2524, November 2013.

[6] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. *arXiv e-prints*, page arXiv:1703.06870, March 2017.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv e-prints*, page arXiv:1512.03385, December 2015.

[8] Glenn Jocher, Yonghye Kwon, guigarfr, Josh Veitch-Michaelis, perry0418, Ttayu, Marc, Gabriel Bianconi, Fatih Baltacı, Daniel Suess, idow09, WannaSeaU, Wang Xinyu, Timothy M. Shead, Thomas Havlik, Piotr Skalski, NirZarrabi, LukeAI, LinCoce, Jeremy Hu, IlyaOvodov, GoogleWiki, Francisco Reveriano, Falak, and Dustin Kendall. ultralytics/yolov3: 43.1map@0.5:0.95 on coco2014, May 2020.

[9] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection. *arXiv e-prints*, page arXiv:1612.03144, December 2016.

[10] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common Objects in Context. *arXiv e-prints*, page arXiv:1405.0312, May 2014.

[11] Fiskeri og kystdepartementet. Planmessig igangsetting av nye arter i oppdrett. 2003.

[12] Keiron O'Shea and Ryan Nash. An Introduction to Convolutional Neural Networks. *arXiv e-prints*, page arXiv:1511.08458, November 2015.

[13] Dabal Pedamonti. Comparison of non-linear activation functions for deep neural networks on MNIST classification task. *arXiv e-prints*, page arXiv:1804.02763, April 2018.

[14] Bruce F. Phillips and J. Stanley Cobb. The biology and management of lobsters. 1980.

[15] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. *arXiv e-prints*, page arXiv:1804.02767, April 2018.

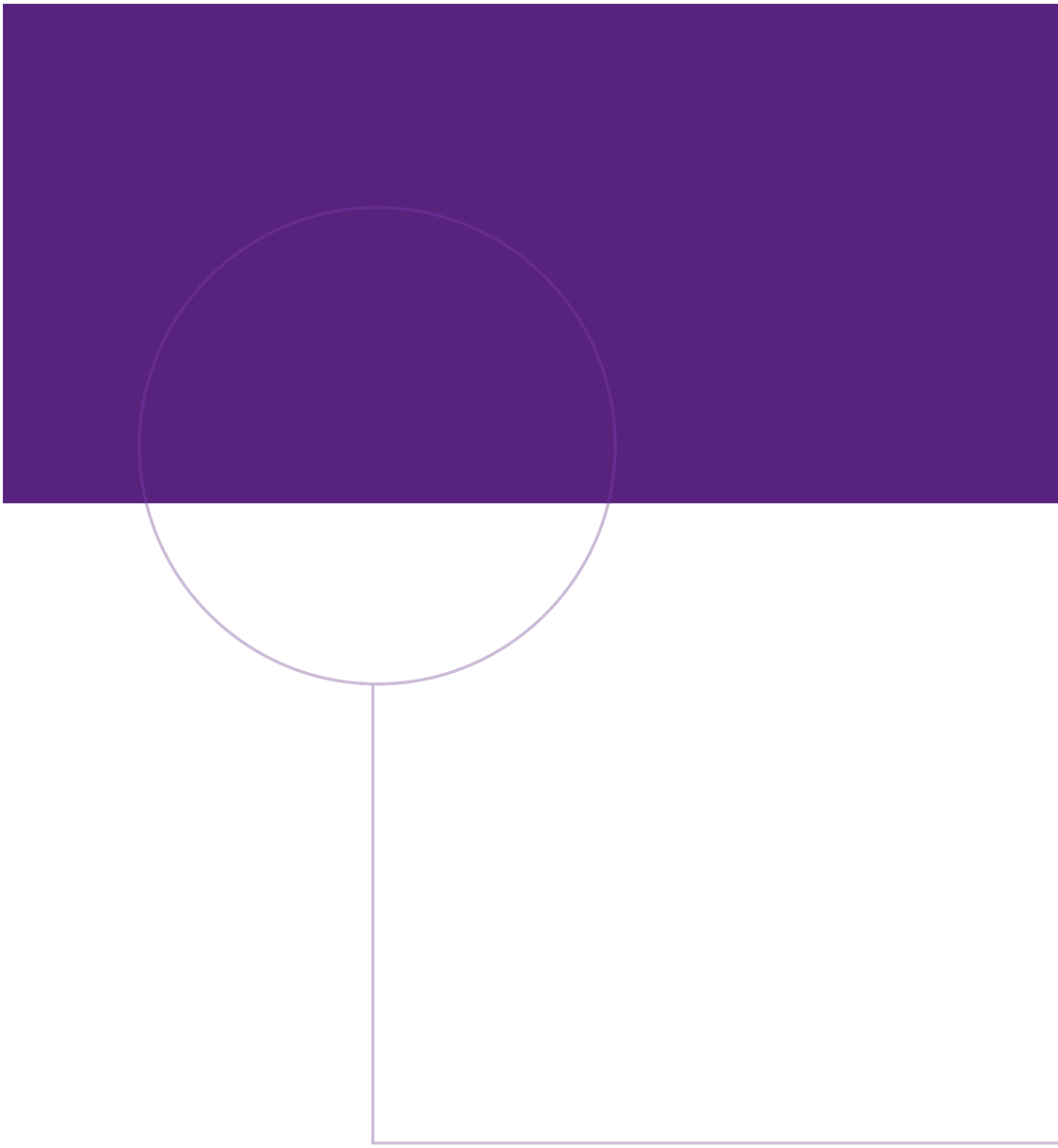
[16] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *arXiv e-prints*, page arXiv:1506.01497, June 2015.

[17] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *arXiv e-prints*, page arXiv:1409.0575, September 2014.

[18] Boris Sekachev, Nikita Manovich, zhiltsov max, Andrey Zhavoronkov, Dmitry Kalinin, Ben Hoff, TOsmanov, Artyom Zankevich, DmitriySidnev, Maksim Markelov, Johannes222, telenachos, Aleksandr Melnikov, Jijoong Kim, Nikita Glazov, Seungwon Jeong, a andre, Sebastian Yonekura, vugia truong, zliang7, Tritin Truong, idriss, Ajay Ramesh, Christian, Codacy Badger, DanVev, Dustin Dorroh, Eduardo, and Eric Jiang. opencv/cvat: Stable release of CVAT + https://cvat.org, May 2020.

[19] S. Sra, S. Nowozin, and S.J. Wright. *Optimization for Machine Learning*. Neural information processing series. MIT Press, 2012.

[20] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.



NTNU

Norwegian University of
Science and Technology