Object Detection on fish images using YOLOv2 network

Andreas Steinvik

April 25, 2019

1 Introduction

In this report I will present my results from using the YOLOv2 network architecture created by J.Redmon et al. [1]

1.1 Dataset

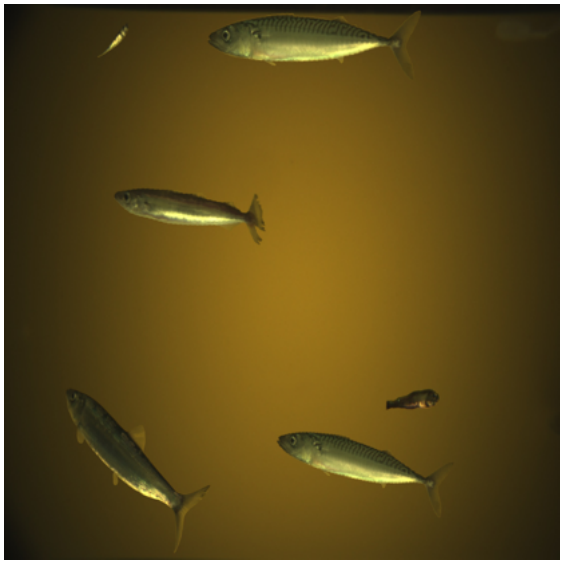

Figure 1: Example of training image[4]

The dataset used for training was underwater pictures of fish created by pasting cutouts of fish on a background. For this I used the image generator created by Ketil Malde [4]with some small modifications. I used a picture size of 416*416 and up to 6 objects per picture. The four classes in the dataset was:

- Herring

- Mackerel
- Benthosema
- BlueWhiting

Each scaled to get relative size differences in the picture.

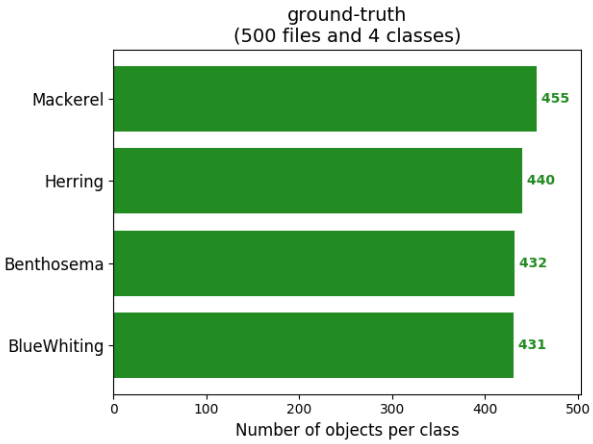

Figure 2: Number of occurences of each class in test set of 500 images

1.2 Code

All my code and the libraries used are available at my github `https://github.com/Asteinvik/INF368---YOLO`

running the code:

- **Calculating anchor boxes:** python3 YAD2K/boxing.py
- **calculating mAP:** python3 mAP/main.py -na
- **generating images:** python3 image-sim-modified/imagesim.py -d . -b Backgrounds -c source_data -n 700 -o images
- **predicting:** python3 retrain_yolo.py -d test.npz

2 Architecture

The YOLO architecture is is made up of 23 convolutional layers with the last having the shape 13,13,36. the (13,13) corresponds to a 13x13 grid of the original picture, while the 36 filters corresponds to 4 anchor boxes * (4 classes+4 box coordinates + 1 confidence). This makes a total of 50,584,836 trainable parameters for the network. For all layers except the last layer i used the pretrained weights from the original author [2]. For creation of the network and translating from the original architecture written in C and into Keras I used

the library YAD2K [3]. For creating mAP scores i used the library mAP by Cartucho [5]. See yolo.png in git repo for a plot of the architecture(the plot was too bit for this report).

The data handling of YAD2K was implemented using an externaly created .npz file containing images and boxes as serialized objects.

2.1 Anchor Boxes

In my network I used 4 anchor boxes. These were generated by taking 5000 sample boxes and using K-means to find the centroids most fitting. See 3 for plot of this process. My final anchors were:

- (78.83591005, 142.26317639)
- (31.0040678, 31.92474576)
- (113.12284407, 81.94561774)
- (169.93079662, 93.39007566)

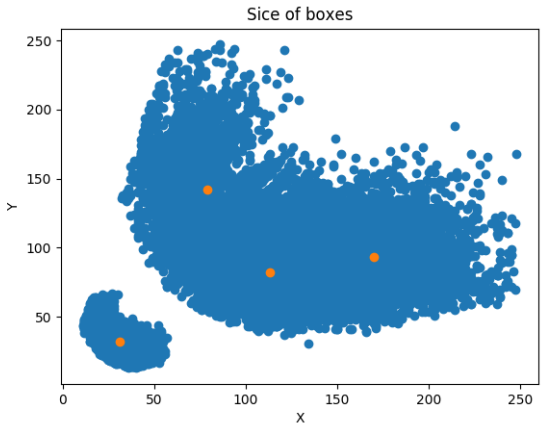

Figure 3: Plot of relative size of boxes. K-means centroids also shown

3 Training

For training I used 700 pictures for each run because the implementation in the YAD2K library was really lacking in a good way to read input. This was a major cause of problems working with this assignment and in hindsight should have been changed completely before I used too much time trying to fix it in other ways. Each run was split in 3. first 5 epochs with only last layer unfrozen, then 30 epochs with everything unfrozen, and lastly up to 30 layers with early stopping and saving the best epoch. after this I generated another new set of 700 images and repeated the process. One problem I had on some run was the

walidation error suddenly jumping to insane heights (largest was over $12 * 10^12$) which made some runs really bad and destroyed my plots. See 4, 5 and 6

4 Results

See 8, 14 and 15 for examples of predicted images. After some initial trouble i got quite good results, with mostly true positive predictions. See 15 for an example of a false positive, where a fin of a herring is misclasified as benthosema. The different classes had mostly the same scores, ranging between 80 and 90 % AP for each class on every test. I did not have any real images, so all these results are from simulated ones.

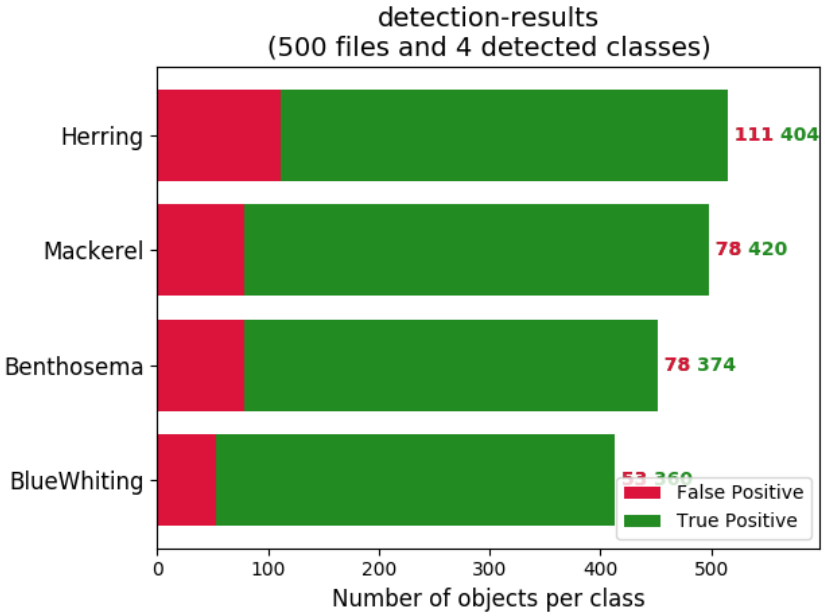

Figure 7: False Positive vs true positive for each class

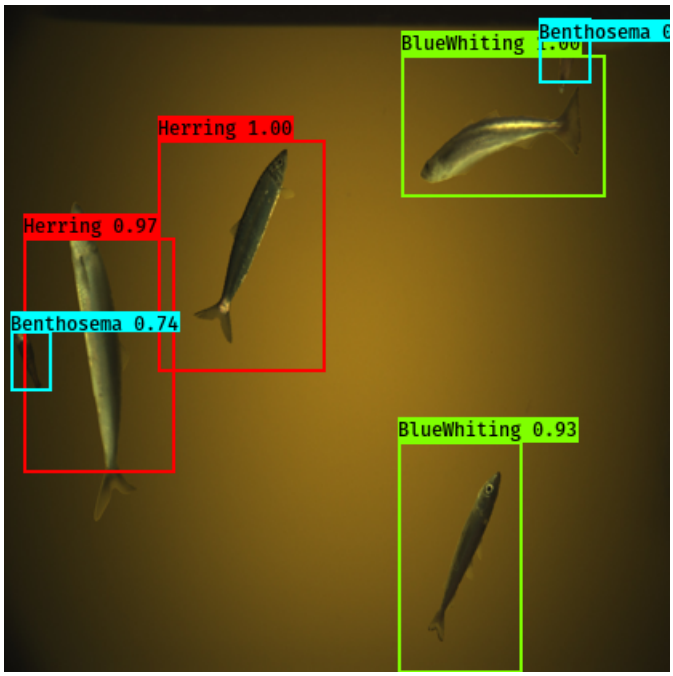

Figure 8: Example of predicted image with confidence and labels

4.1 mAP

My final ended up at 85.60% with an Average precision of each class at:

- **Benthosema:**85.75%
- **BlueWhiting:**78.48%
- **Herring:**87.94%
- **Mackerel:**90.24%

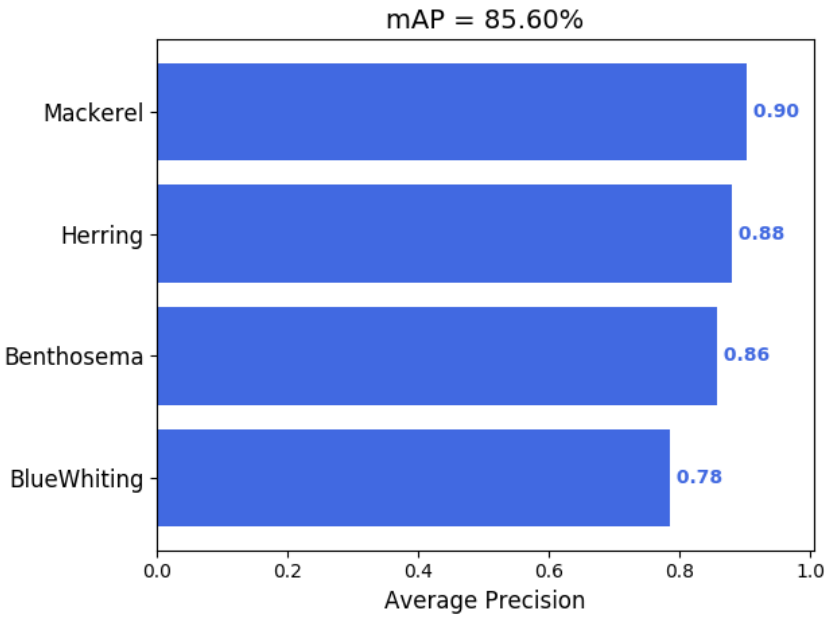

Figure 9: Average Precicion for each class

4.2 Precision-Recall

All plots of PR-curves looks very similar, and they all show behaviour of a good classifier: as strait as possible line with as high as posible precision.

4.2.1 Herring

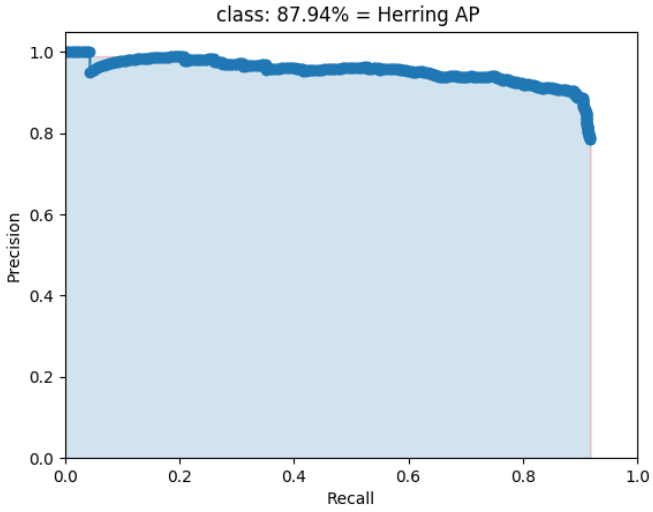

Figure 10: Precicion-recall curve for the Herring class

4.2.2 Benthosema

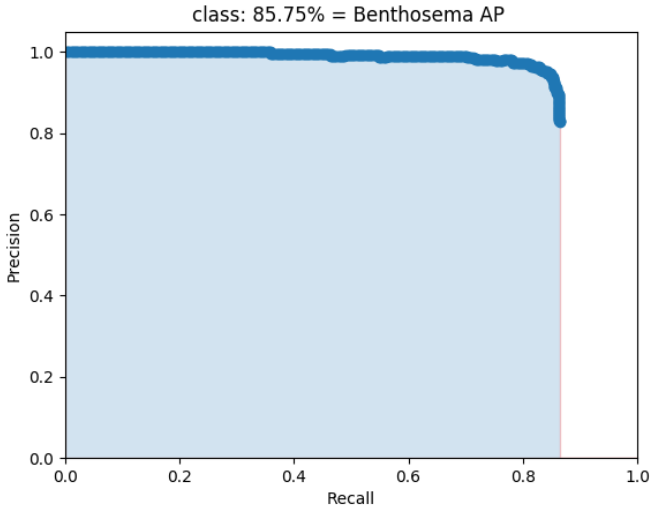

Figure 11: Precicion-recall curve for the Benthosema class

4.2.3 BlueWhiting

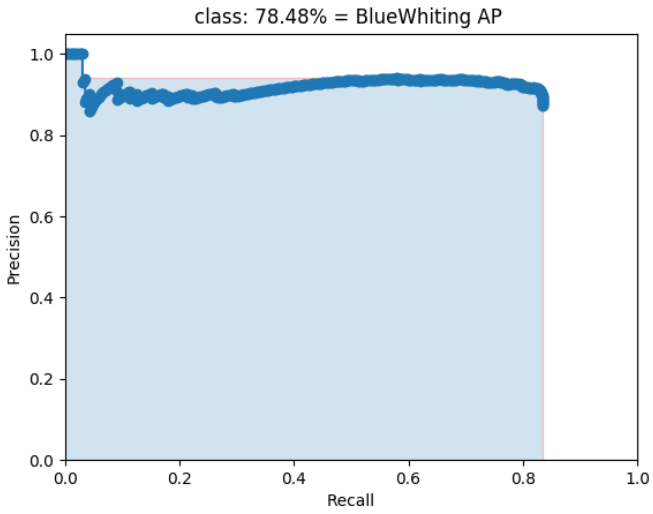

Figure 12: Precicion-recall curve for the BlueWhiting class

4.2.4 Mackerel

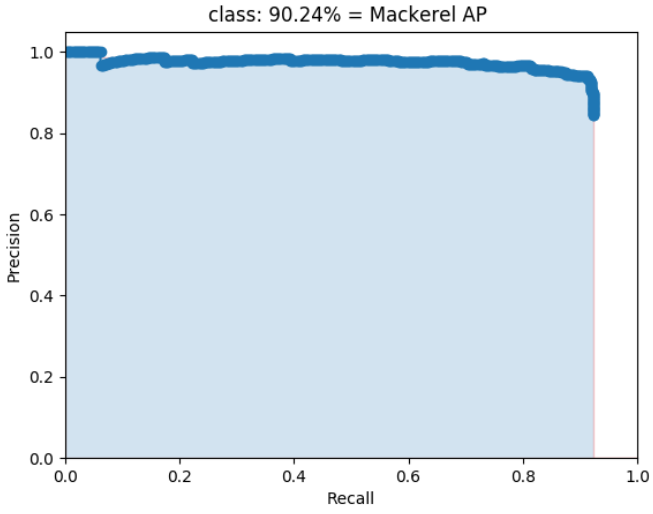

Figure 13: Precicion-recall curve for the Mackerel class

5 Discussion

After working out initial toubles with understanding how the different libraries worked the biggest difficulties was with how YAD2K demanded input. Some part of my pipeline from generating images to training on them would crash all the time until i got everything sorted out. The reason I did not change this when I started was that it took quite some time to understand the inner workings of YAD2K and when I finally did I had come up with a way around the problems. These limitations made me use atmost 700 images for each training run, something I suspect was the biggest limit on getting better accuracy in training. I first tried with the original achor boxes, but changed to customized after doing K-means. The biggest effect I saw on this was that the training was more stable, with fewer explosions in validation loss.

5.1 Further work

It would be fun to test on real images if I can find some.

I'm also planning on forking the YAD2K library for some improvements when I get more time on my hands.

References

[1] YOLO9000: Better, Faster, Stronger (2016)
Joseph Redmon et. al. `https://arxiv.org/abs/1612.08242`

[2] YOLO github
Joseph Redmon `https://github.com/pjreddie/darknet`

[3] YAD2K github
allanzelener `https://github.com/allanzelener/YAD2K/`

[4] Library used for creating images
Ketil Malde `https://github.com/ketil-malde/image-sim-modified`

[5] mAP github
Cartucho `https://github.com/Cartucho/mAP`

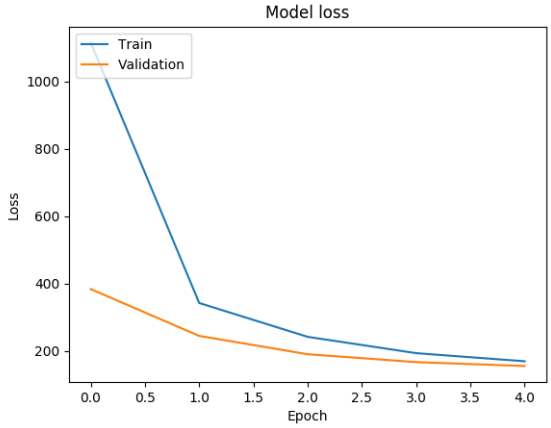

Figure 4: first step in training

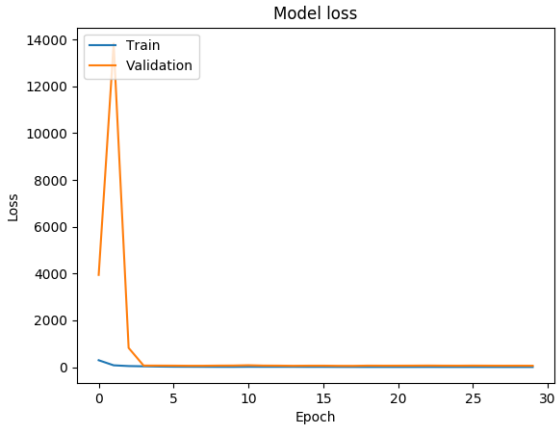

Figure 5: second step in training

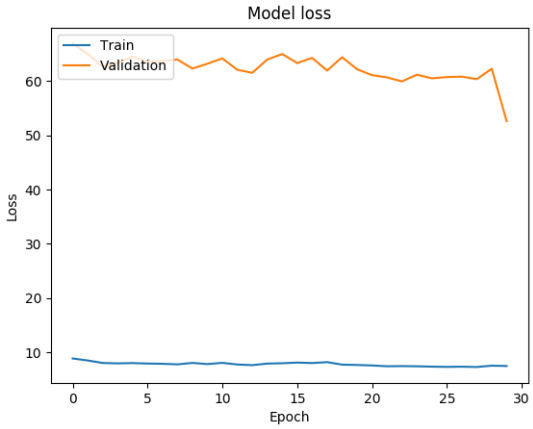

Figure 6: third step in training

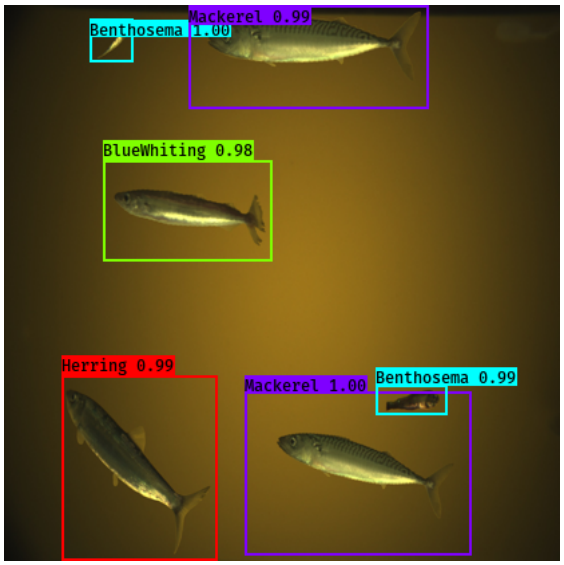

Figure 14: Example of predicted image with confidence and labels

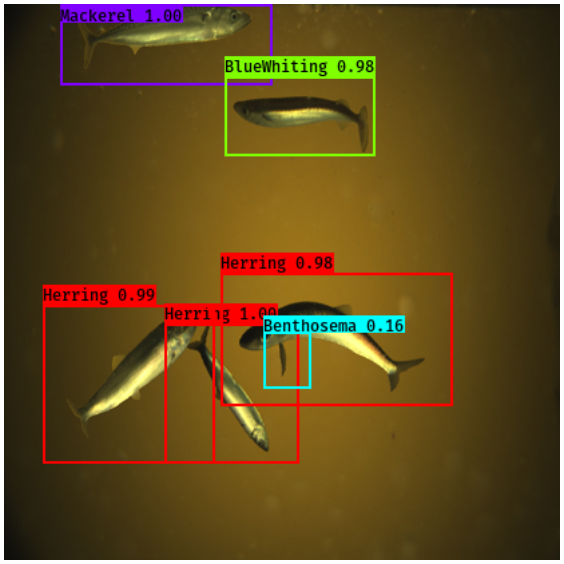

Figure 15: Example of predicted image with confidence and labels