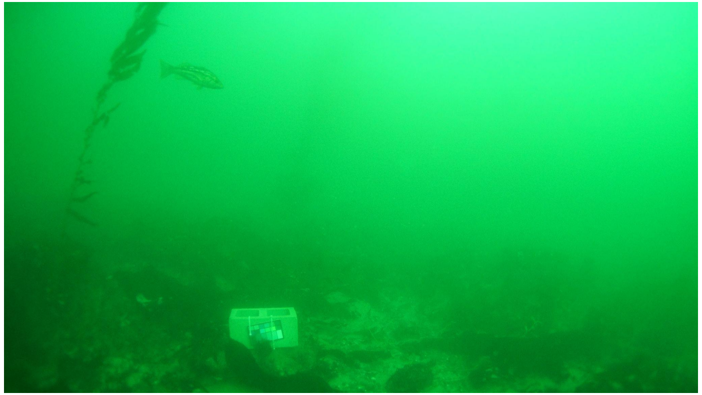
On-board fish detector using machine learning

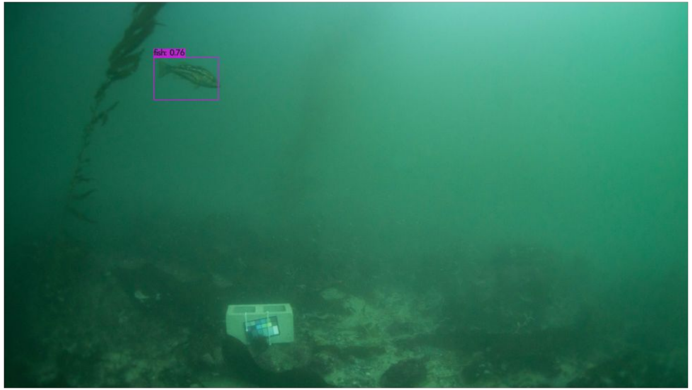
Clare Walker, August 2021

Aim: Automate fish detection for passive imaging systems



FishOASIS imaging and acoustic system deployed in kelp forest MPAs

- 1500 photos a day stored on R-Pi
- All fish counted by scientists



Incorporate on-board ML tool to automate detection and counting

- Only store photos with fish
- Only check sample of labels

Lit review

Observations from literature review

- Much of existing literature applying CNNs to images of fish focus on classification
- Object detection has predominantly been used for post-processing in labs
- Open-source tools exist for this context: VIAME, FathomNet, Fish4Knowledge
- Results confirm that YOLO is faster than region proposal techniques, but requires more training data (>1000 annotations of each class)
- Most results come from coral reefs and other shallow, relatively static environments — no evidence (so far) of application in kelp forests
- Tools for on-board, real-time detection have been developed for midwater ROVs rather than passive, low-cost imaging systems

Most similar paper by Coro, to be published July 2021

Similarities:

- On-board object detection for underwater applications
- Built as an ‘intelligent’ component of passive monitoring system
- Open-source code, deployable after specifying a few parameters
- Utilizes YOLO Tiny

Differences:

- Optimised for large fish, rather than detecting all fish in an image
- Tested in relatively static environments, rather than in dynamic kelp forests
- No re-training for use in new environments, harder to update based on experience
- Training set generated by animating fish, rather than on live data
- Utilizes YOLO Tiny v3 rather than v4, as well as options for two other models

Data & Methodology

Overview of key steps followed

1. Trained YOLO Tiny v4 on four different training sets:
    a. Excluded images before / after cut-off times (determined by brightness)
    b. Excluded images with no fish, i.e. downsampled majority class
    c. Incorporated simple batch image enhancements (edge enhance using PIL)
    d. Removed labels for small fish below YOLO recommended size (see analysis of results)
2. Calculated performance metrics to reflect keep/discard precision, recall, F1
3. Apply trained model to test sets
4. Create data visualisations for results (e.g. keep/discard heatmap, count time series)
5. Clean up notebooks and create GitHub repo

YOLO v4 Tiny architecture

```
   layer   filters  size/strd(dil)      input                output
   0 conv     32       3 x 3/ 2    608 x 608 x   3 ->  304 x 304 x  32 0.160 BF
   1 conv     64       3 x 3/ 2    304 x 304 x  32 ->  152 x 152 x  64 0.852 BF
   2 conv     64       3 x 3/ 1    152 x 152 x  64 ->  152 x 152 x  64 1.703 BF
   3 route  2                               1/2 ->  152 x 152 x  32
   4 conv     32       3 x 3/ 1    152 x 152 x  32 ->  152 x 152 x  32 0.426 BF
   5 conv     32       3 x 3/ 1    152 x 152 x  32 ->  152 x 152 x  32 0.426 BF
   6 route  5 4                                 ->  152 x 152 x  64
   7 conv     64       1 x 1/ 1    152 x 152 x  64 ->  152 x 152 x  64 0.189 BF
   8 route  2 7                                 ->  152 x 152 x 128
   9 max                2x 2/ 2    152 x 152 x 128 ->   76 x  76 x 128 0.003 BF
  10 conv    128       3 x 3/ 1     76 x  76 x 128 ->   76 x  76 x 128 1.703 BF
  11 route  10                              1/2 ->   76 x  76 x  64
  12 conv     64       3 x 3/ 1     76 x  76 x  64 ->   76 x  76 x  64 0.426 BF
  13 conv     64       3 x 3/ 1     76 x  76 x  64 ->   76 x  76 x  64 0.426 BF
  14 route  13 12                               ->   76 x  76 x 128
  15 conv    128       1 x 1/ 1     76 x  76 x 128 ->   76 x  76 x 128 0.189 BF
  16 route  10 15                               ->   76 x  76 x 256
  17 max                2x 2/ 2     76 x  76 x 256 ->   38 x  38 x 256 0.001 BF
  18 conv    256       3 x 3/ 1     38 x  38 x 256 ->   38 x  38 x 256 1.703 BF
  19 route  18                              1/2 ->   38 x  38 x 128
  20 conv    128       3 x 3/ 1     38 x  38 x 128 ->   38 x  38 x 128 0.426 BF
  21 conv    128       3 x 3/ 1     38 x  38 x 128 ->   38 x  38 x 128 0.426 BF
  22 route  21 20                               ->   38 x  38 x 256
  23 conv    256       1 x 1/ 1     38 x  38 x 256 ->   38 x  38 x 256 0.189 BF
  24 route  18 23                               ->   38 x  38 x 512
  25 max                2x 2/ 2     38 x  38 x 512 ->   19 x  19 x 512 0.001 BF
  26 conv    512       3 x 3/ 1     19 x  19 x 512 ->   19 x  19 x 512 1.703 BF
  27 conv    256       1 x 1/ 1     19 x  19 x 512 ->   19 x  19 x 256 0.095 BF
  28 conv    512       3 x 3/ 1     19 x  19 x 256 ->   19 x  19 x 512 0.852 BF
  29 conv     18       1 x 1/ 1     19 x  19 x 512 ->   19 x  19 x  18 0.007 BF
  30 yolo
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05
nms_kind: greedynms (1), beta = 0.600000
  31 route  27                                  ->   19 x  19 x 256
  32 conv    128       1 x 1/ 1     19 x  19 x 256 ->   19 x  19 x 128 0.024 BF
  33 upsample                 2x     19 x  19 x 128 ->   38 x  38 x 128
  34 route  33 23                               ->   38 x  38 x 384
  35 conv    256       3 x 3/ 1     38 x  38 x 384 ->   38 x  38 x 256 2.555 BF
  36 conv     18       1 x 1/ 1     38 x  38 x 256 ->   38 x  38 x  18 0.013 BF
  37 yolo
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05
nms_kind: greedynms (1), beta = 0.600000
```

More information on object detection using CNNs and further explanation of YOLO found here

Summary of data split

Date	Site C	Site C.2			
		1	2	3	4
25/05/18					
26/05/18	5				
10/07/18					
11/07/18					
12/07/18					
13/07/18					4
14/07/18					
15/07/18					
16/07/18					
17/07/18		1	3		
18/07/18					
19/07/18					
20/07/18					
21/07/18		2	6		
22/07/18					
23/07/18					

1 **Training & test**: Same site, same perspective, same days, different images

2 **Test**: Same site, same perspective, different days (temporal)

3 **Test**: Same site, same days, different perspective (spatial)

4 **Test**: Same site, different days, different perspective (temporal and spatial)

5 **Test**: Different site, different days (temporal and spatial)

6 **Proof of application**: Apply to unlabeled data, compare time series

Definition of performance metrics

Problem:

- We want to measure performance terms of binary classification: Keep or Discard
- The test data are imbalanced, typically **20% keep (at least one fish)** and **80% discard (no fish)**

Best practice:

- **Majority class** is typically referred to as the **negative outcome** (e.g. “no change” or “negative test result“)
- **Minority class** is typically referred to as the **positive outcome** (e.g. “change” or “positive test result”)
- This is because metrics measure ability to detect positive outcomes — if majority class were defined as positive outcome, a model that only predicted the majority class would look better than random

Solution:

- We will define **Keep as positive** outcome and **Discard as negative** outcome
- Note that due to above, results would look much better if defined in opposite direction

Definition of performance metrics

Measure in terms of binary classification: **Keep (Positive) / Discard (Negative)**

- True positive (TP) = found > 0 fish, image had > 0 fish
- False positive (FP) = found > 0 fish, image had no fish (error - okay)
- True negative (TN) = found no fish, image had no fish
- False negative (FN) = found no fish, image had > 0 fish (error - not okay)

Key metrics:

- Precision = TP / (TP + FP) : proportion of images kept that actually contained fish
- Recall = TP / (TP + FN) : proportion of images kept out all images that should have been kept
- F1 = 2 * Pr * Re / (Pr + Re) : harmonic mean of precision and recall

Example: Choice of positive outcome impacts results

Positive outcome: Keep

Confusion matrix

	Kept (+)	Discarded (-)	*Total*
Has fish (+)	**133** (TP)	**64** (FN)	*197*
Has no fish (-)	**41** (FP)	**762** (TN)	*803*
Total	174	826	*1000*

Accuracy = 895 / 1000 = **90%**

Precision = 133 / 174 = **76%**

Recall = 133 / 197 = **68%**

Positive outcome: Discard

Confusion matrix

	Discarded (+)	Kept (-)	*Total*
Has no fish (+)	**762** (TP)	**41** (FN)	*803*
Has fish (-)	**64** (FP)	**133** (TN)	*197*
Total	*826*	*174*	*1000*

Accuracy = 895 / 1000 = **90%**

Precision = 762 / 826 = **92%**

Recall = 762 / 803 = **95%**

Results

Preliminary results (imbalanced test sets)

		Model a *(n = 4171, % pos = 20)*			**Model b** *(n = 837, % pos = 100)*		
	% pos	Precision	Recall	F1	Precision	Recall	F1
Valid (1)	*20*	82	49	61	**76**	**68**	**72**
Valid (1) - no small fish	*9 (a)* *8 (b)*	*39*	*50*	*44*	***40***	***81***	***53***
Test (2)	*11*	57	46	51	48	63	54
Test (3)	*13*	48	43	45	31	66	42
Test (4)	*37*	61	55	58	51	70	59
Test (5)	*32*	32	70	44	32	80	46

Preliminary results (balanced test sets) - *better precision*

		Model a *(n = 4171, % pos = 20)*			**Model b** *(n = 837, % pos = 100)*		
	% pos	Precision	Recall	F1	Precision	Recall	F1
Valid (1)	*50*	95	49	65	**94**	**68**	**78**
Valid (1) - no small fish	*50*	*85*	*50*	*63*	***92***	***81***	***86***
Test (2)	*50*	95	46	62	97	63	77
Test (3)	*50*	95	43	59	80	66	72
Test (4)	*50*	74	55	63	66	70	68
Test (5)	*50*	47	70	56	48	80	60

Analysis of results: Where is the model going wrong?

False negatives

This occurs when the model fails to detect fish in an image, which is usually owing to:

1. Uncommon/rare fish not in training set
2. Labelled fish are **blurry/unclear ***
3. Labelled fish are **too small** **

False positives

This occurs when the model confuses something else for a fish, usually:

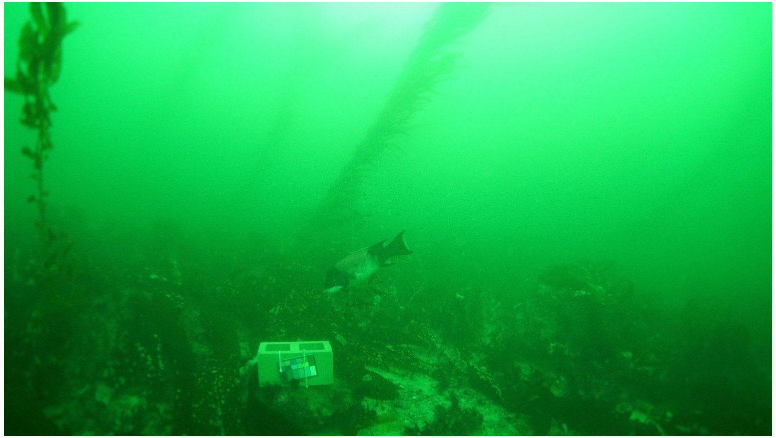
1. Slight variation in water color
2. Kelp

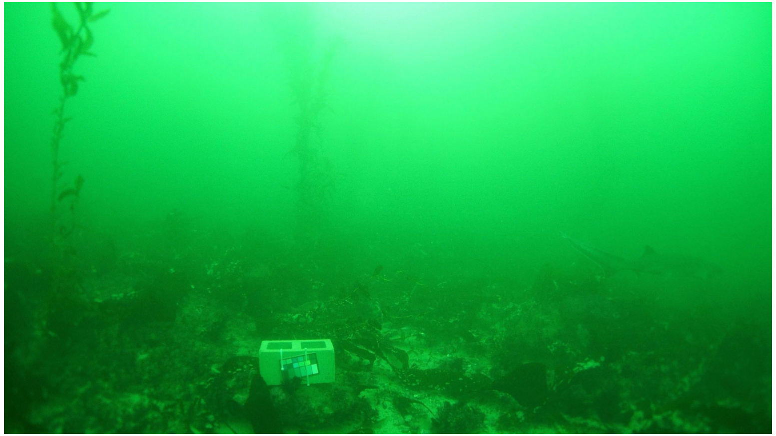
Conducted two further experiments:

c. Incorporated simple batch image enhancements from PIL (addresses *)

d. Removed labels for small fish below YOLO recommended size (addresses **)

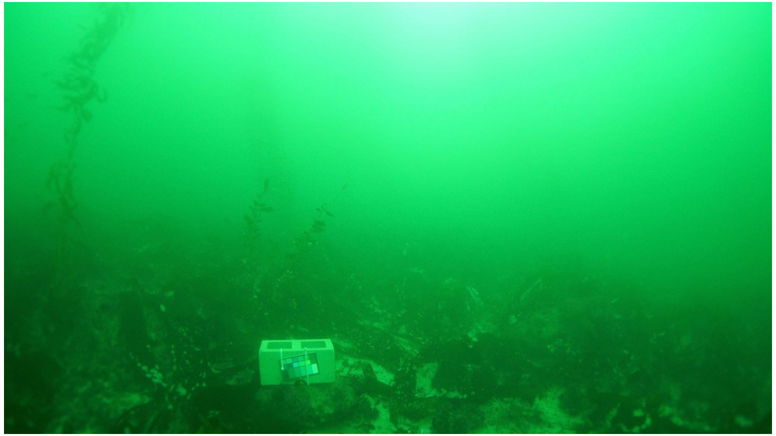
Analysis of results: Uncommon/rare fish (FN)





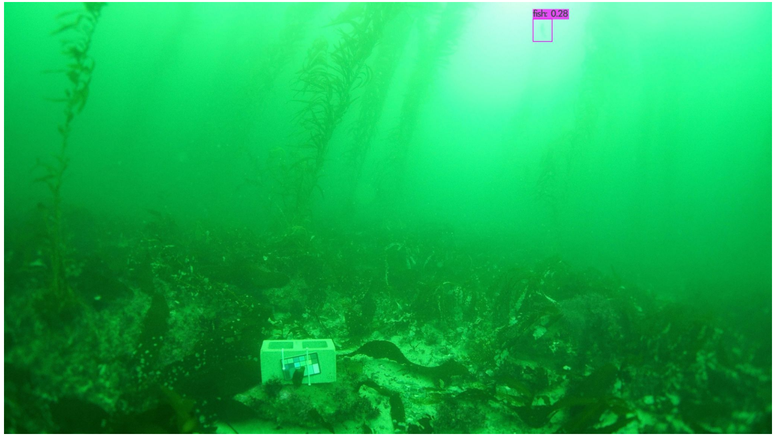
Annotated results sorted by TP, TN, FP, FN are saved here

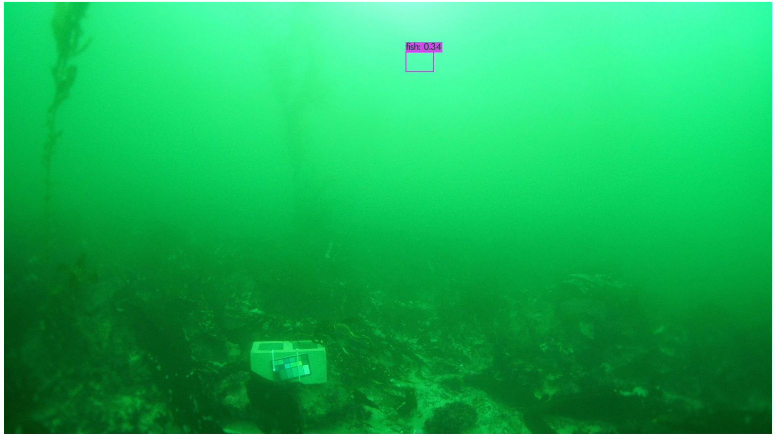
Analysis of results: Labelled fish too blurry (FN)

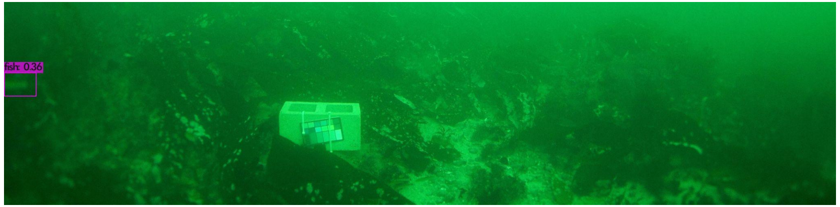


Annotated results sorted by TP, TN, FP, FN are saved [here]

Analysis of results: Labelled fish too blurry (FP)

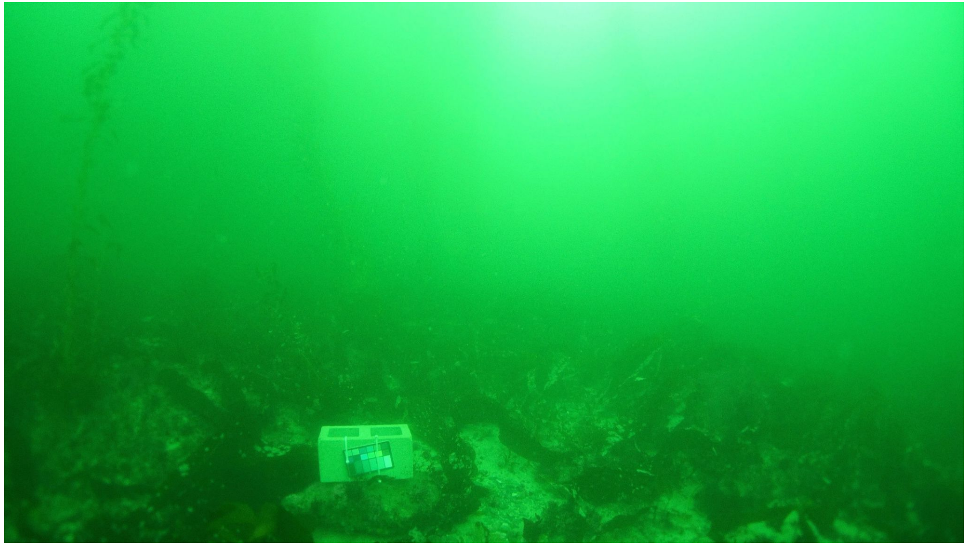


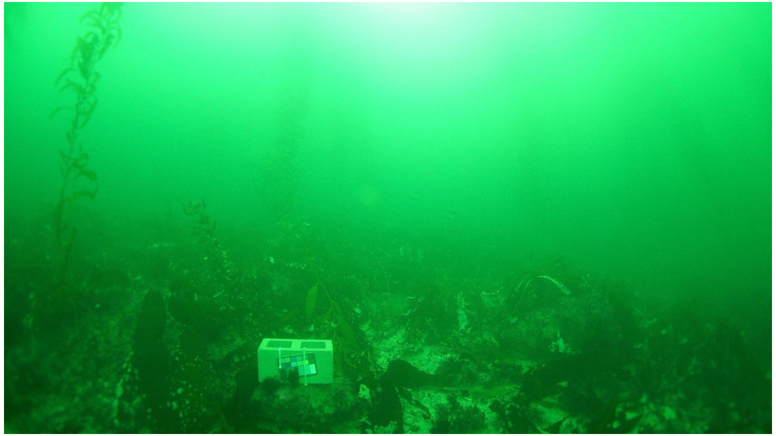




Annotated results sorted by TP, TN, FP, FN are saved here

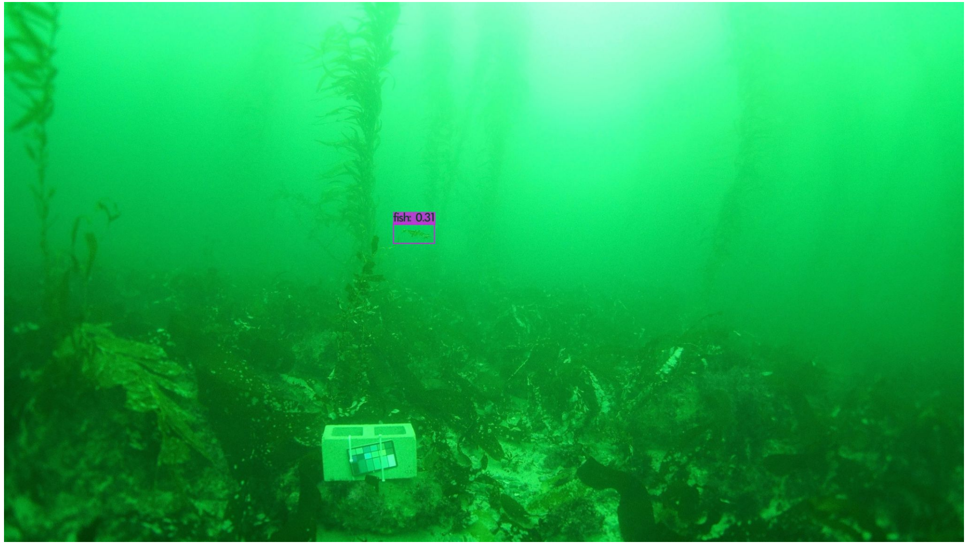
Analysis of results: Labelled fish too small (FN)

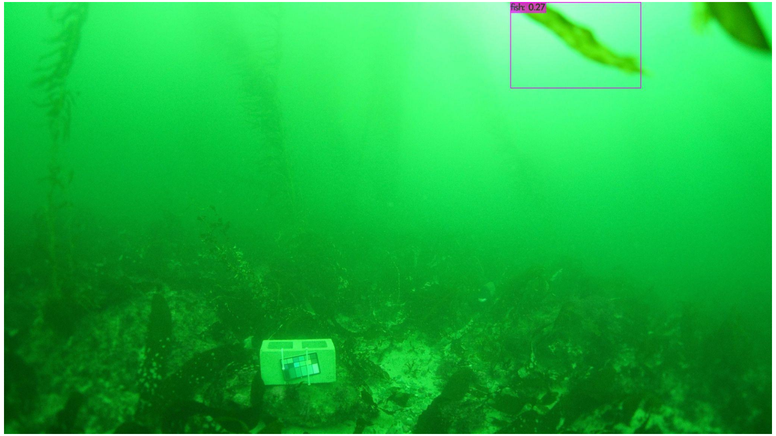




Annotated results sorted by TP, TN, FP, FN are saved here

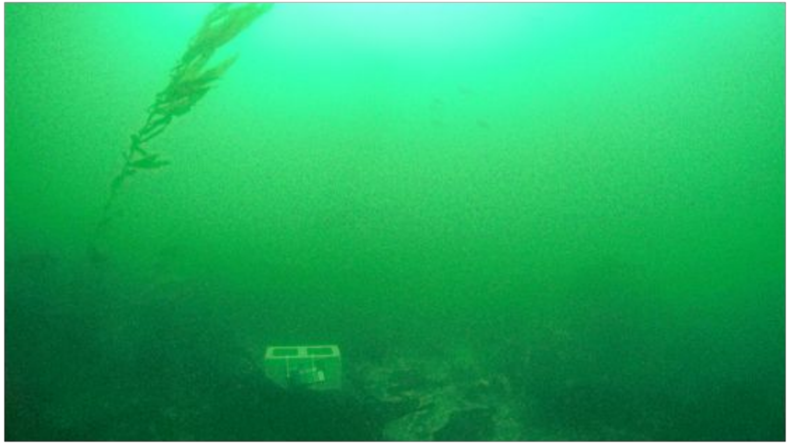
Analysis of results: Kelp confused with fish (FP)





Annotated results sorted by TP, TN, FP, FN are saved here

Example: Best image processing from PIL (Model c)



```
def enhance(path):
  im = Image.open(path)
  im = im.filter(EDGE_ENHANCE_MORE)
  enh = ImageEnhance.Contrast(im)
  return enh.enhance(1.2)
```

Preliminary results (imbalanced test sets)

		Model c *(n = 837, % pos = 100)*			**Model d** *(n = 837, % pos = 47)*		
	% pos	Precision	Recall	F1	Precision	Recall	F1
Valid (1)	*20* / *8*	59	46	52	69	44	53
Test (2)							
Test (3)							
Test (4)							
Test (5)							

Preliminary results (balanced test sets) - *better precision*

		Model c *(n = 837, % pos = 100)*			**Model d** *(n = 837, % pos = 47)*		
	% pos	Precision	Recall	F1	Precision	Recall	F1
Valid (1)	*50*	86	46	60	97	44	60
Test (2)							
Test (3)							
Test (4)							
Test (5)							

Conclusion

Conclusion: Further improvement limited by 5 challenges

Key takeaways:

- Best results achieved with **model b** (trained on only positive examples) and tested on set **without small fish**
- Given existing labels and basic image enhancement packages, this appears to be an **upper bound on performance**

Further improvements are limited by:

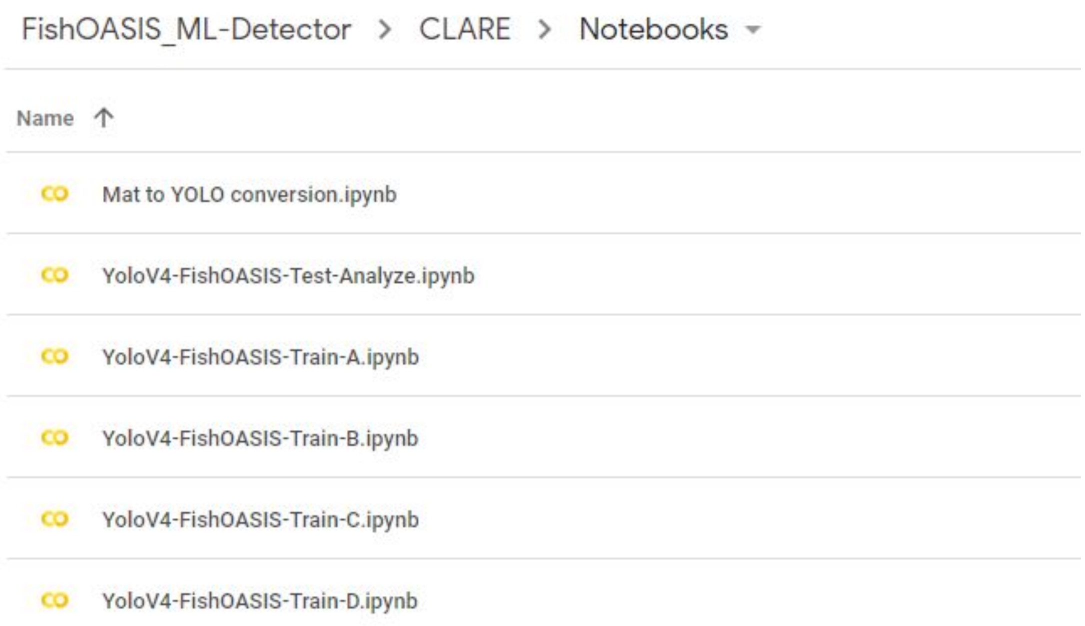
1. Many fish of interest are too small for basic YOLO v4 implementation
2. Some labels are too blurry to make out, especially without bespoke editing (e.g. in Adobe LightRoom)
3. Fish are too heterogenous to treat as one class
4. Real data is imbalanced (i.e. % images without fish > % images with fish)
5. Kelp can look like fish resulting in false positives (less severe)

These could be addressed by further research:

- Define and apply more sophisticated image processing algorithms (e.g. https://colorcorrection.firebaseapp.com/)
- Use another version of YOLO optimised for small objects
- Relabel images with only distinct fish

However, there is no obvious solution for class imbalance, fish heterogeneity and false positives due to kelp

Colab Notebooks will be shared on GitHub (w/o data)

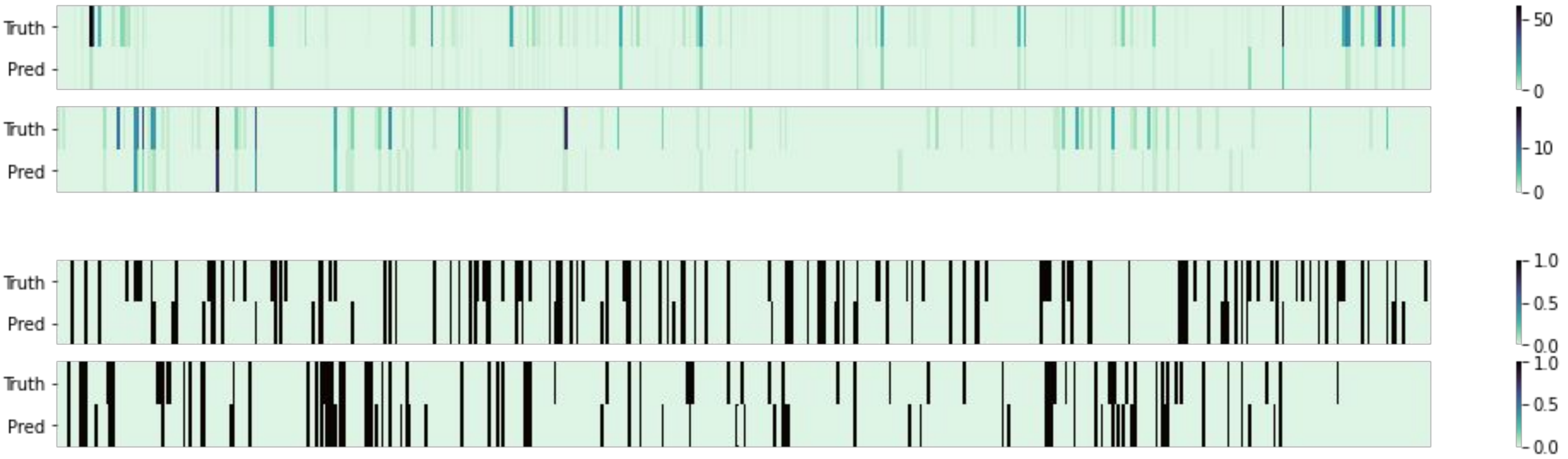


Next steps

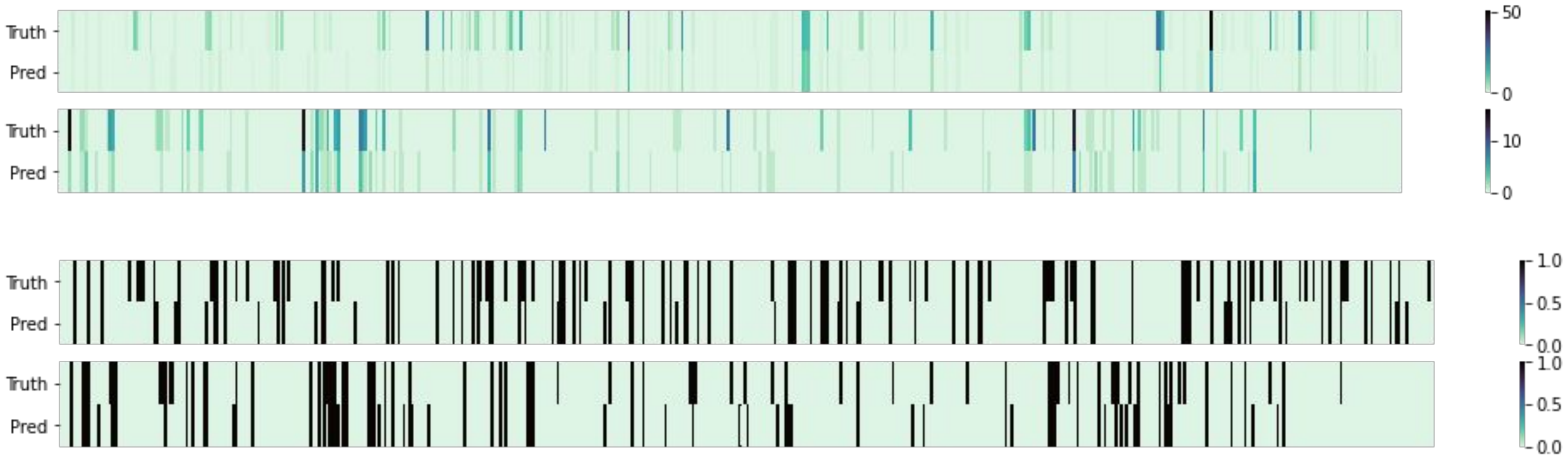
1. Try a few more experiments with YOLO Tiny v4:
    a. "Clean" labels (remove unclassified) and increase input image size
    b. "Cleaner" labels (remove unclassified, remove edge fish, remove small fish) and increase input image size
    c. Split image into e.g. 6 segments
    d. Apply Derya's image enhancement algorithm + other literature, using color palette
2. Treat as pure classification task, i.e. try experiment with VGG or ResNet

Heatmaps

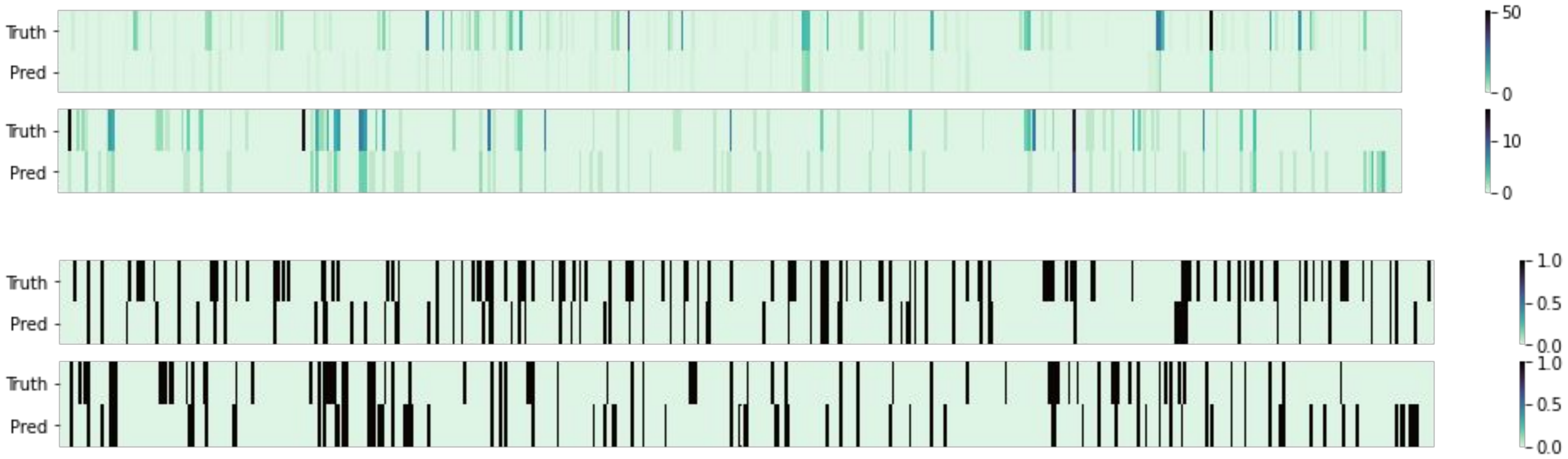
Heatmaps | Model a, Valid (1)



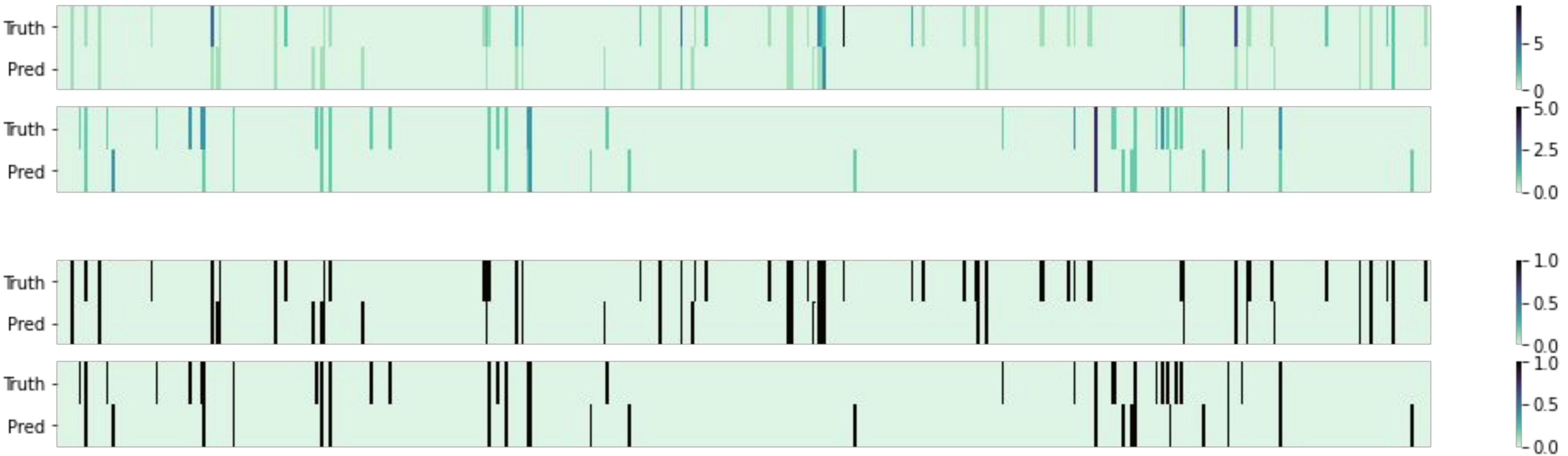
Heatmaps | Model b, Valid (1)



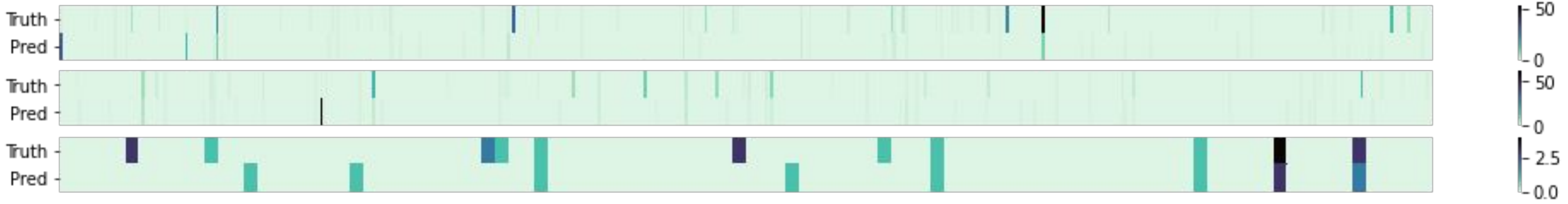
Heatmaps | Model c, Valid (1)



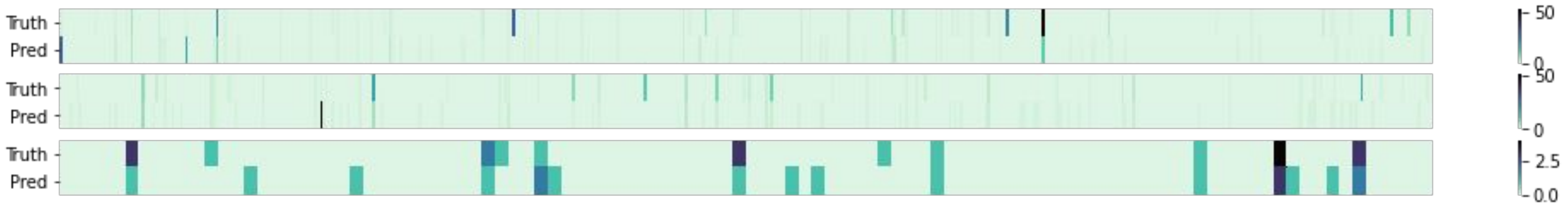
Heatmaps | Model d, Valid (1)

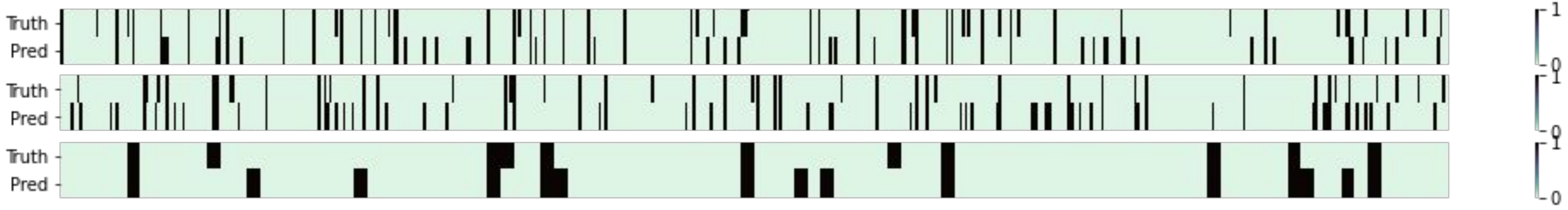


Heatmaps | Model a, Test (2)

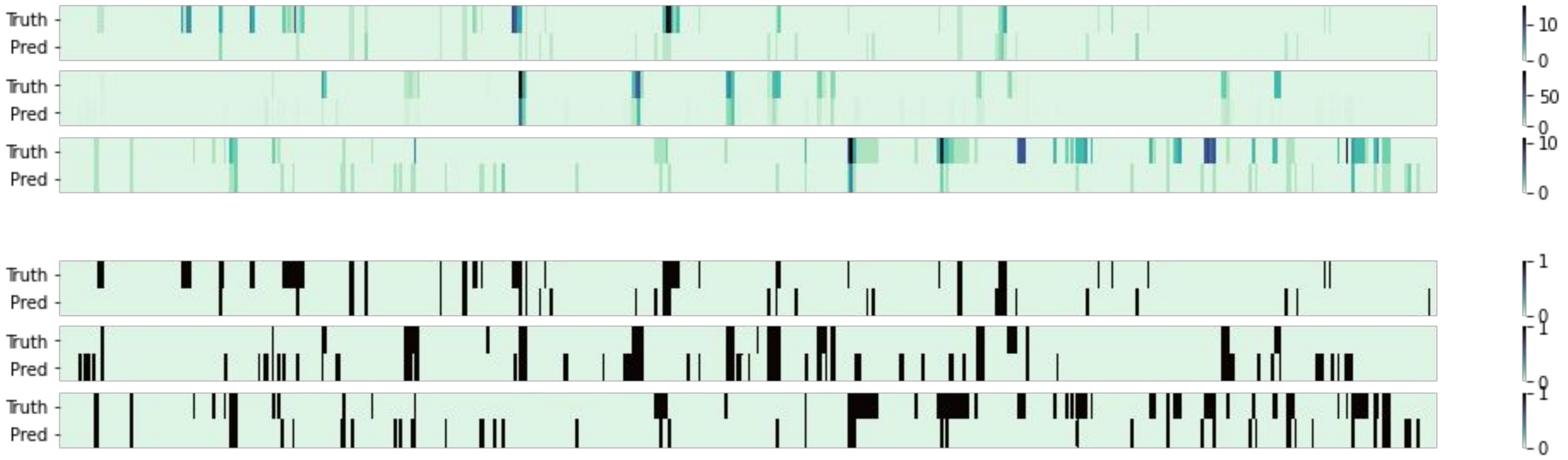


Heatmaps | Model b, Test (2)

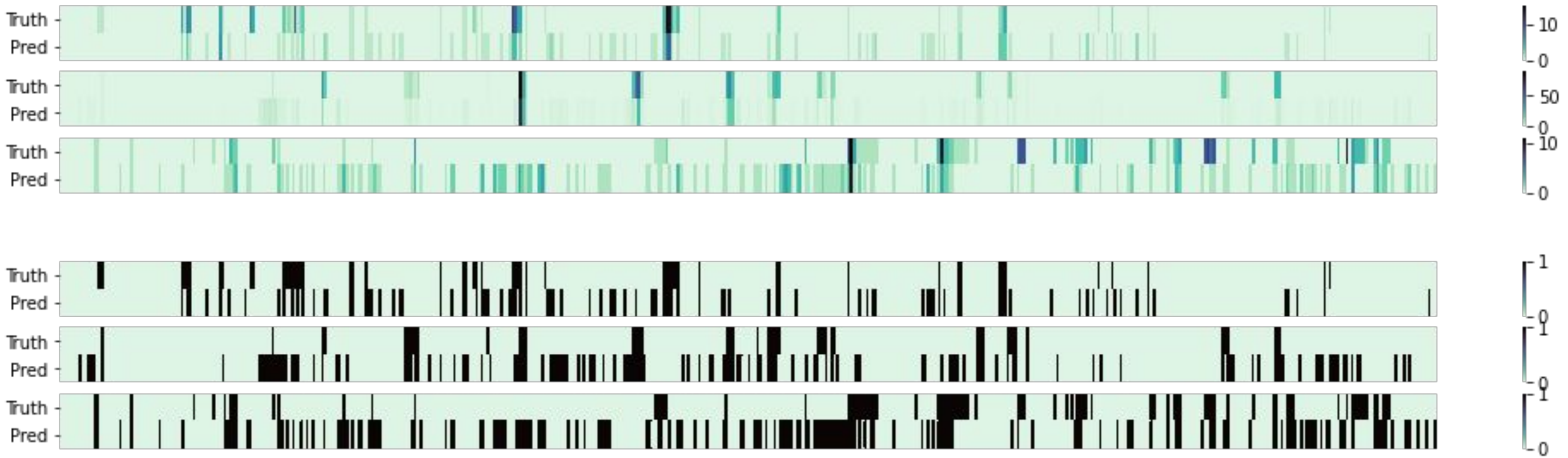




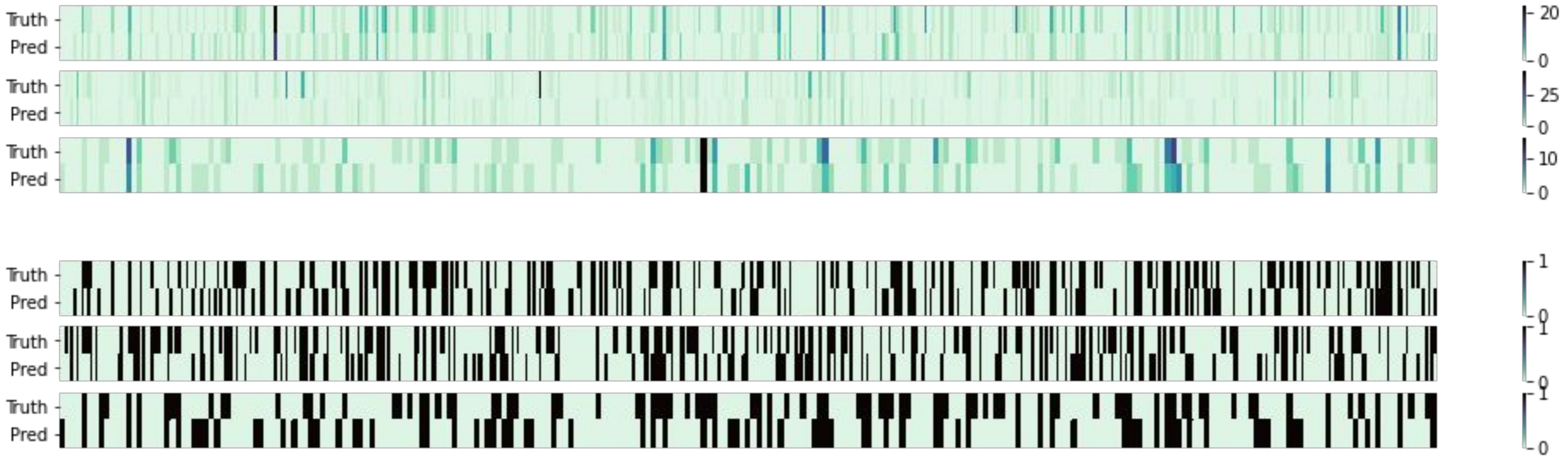
Heatmaps | Model a, Test (3)



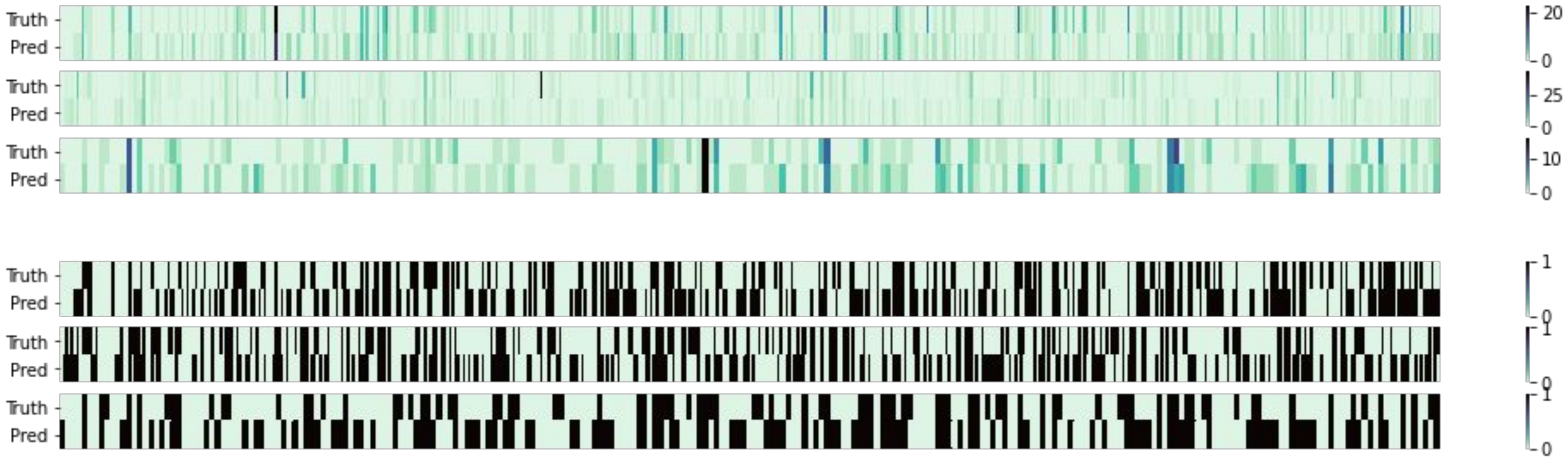
Heatmaps | Model b, Test (3)



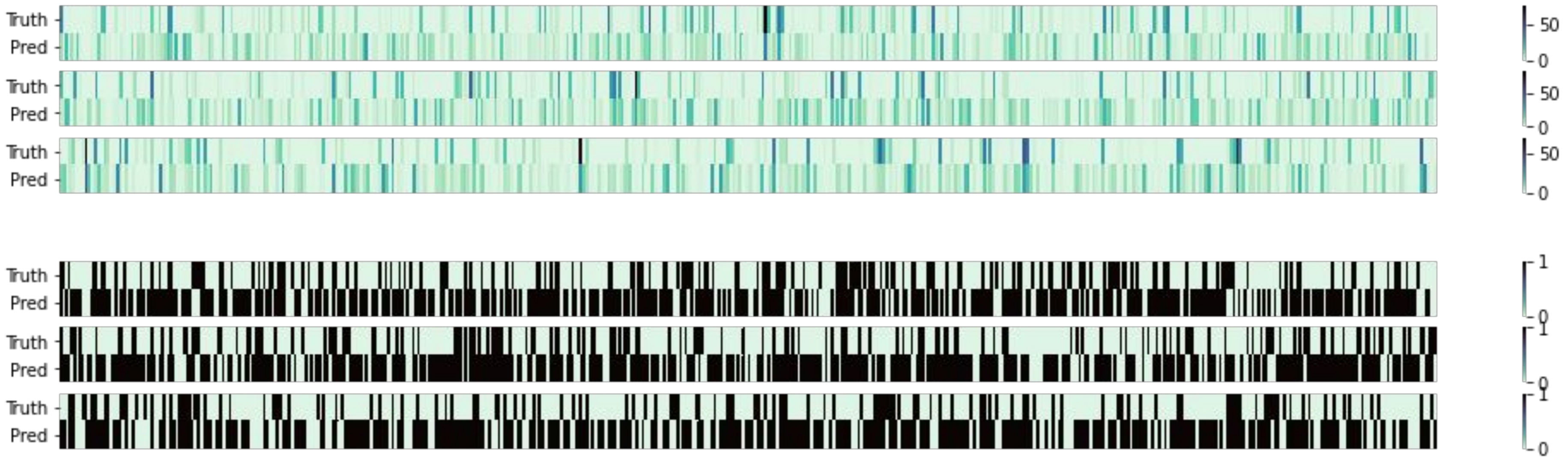
Heatmaps | Model a, Test (4)



Heatmaps | Model b, Test (4)



Heatmaps | Model a, Test (5)



Heatmaps | Model b, Test (5)