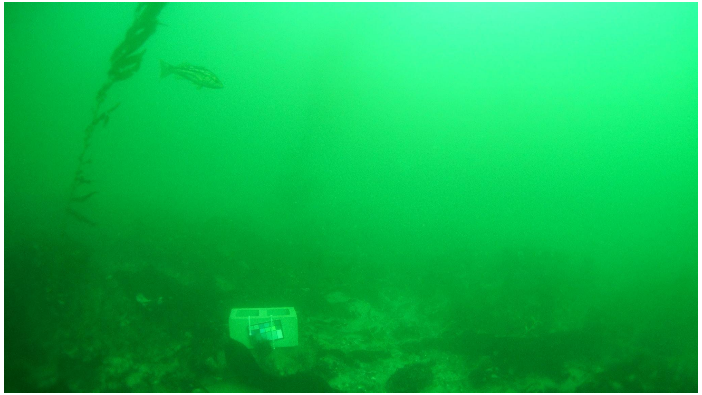
Object Detection with CNNs

Overview of key concepts & algorithms

Agenda

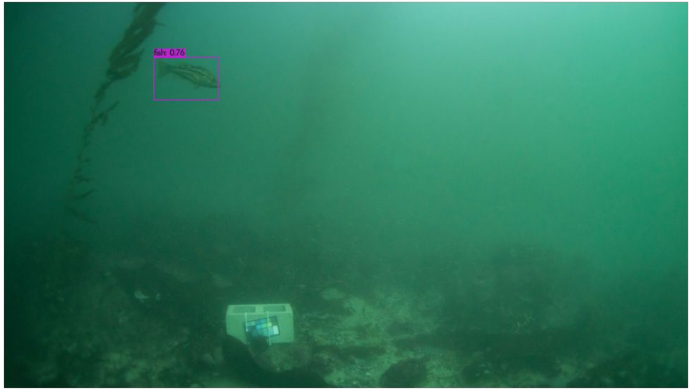
- Context
- Terminology
- CNN Building blocks
- Common approaches
- Deep-dive: YOLO
- Prior research

Context | We want to find fish in kelp forest images



FishOASIS imaging and acoustic system deployed in kelp forest MPAs

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



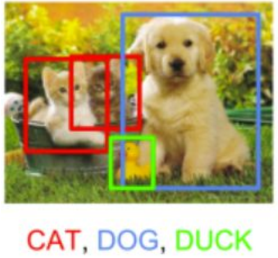
Incorporate on-board ML tool to automate detection and counting

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

Terminology | Key tasks for an object detector

Object classification

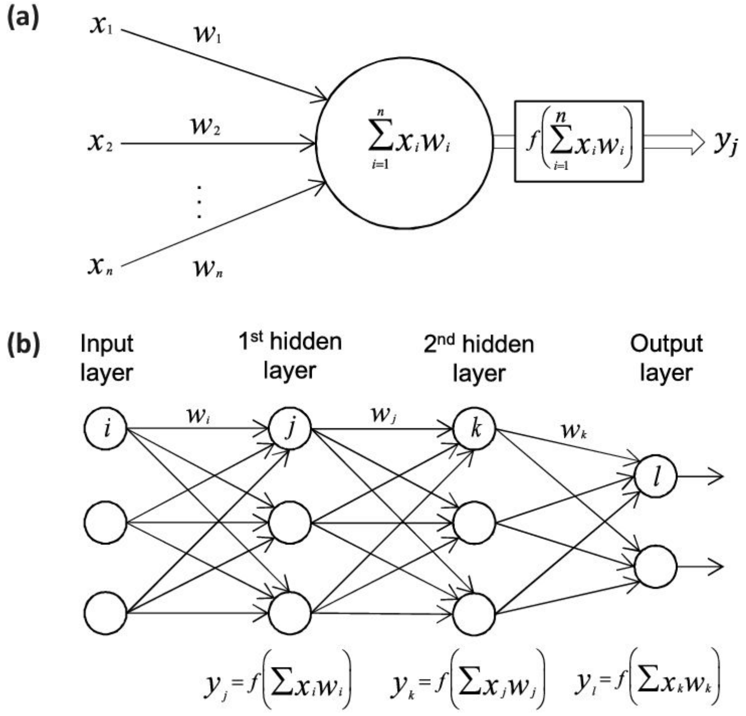
Object detection



Object localisation

Instance segmentation

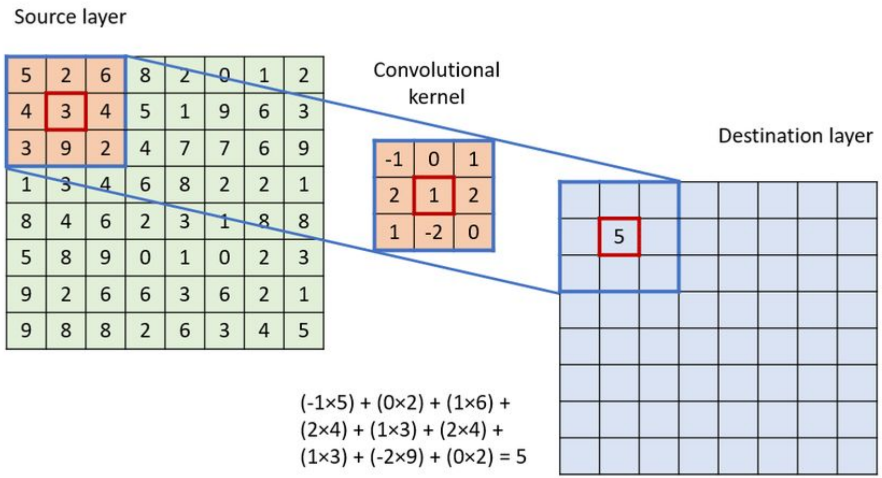
CNN building blocks | Fully connected layers (trained)



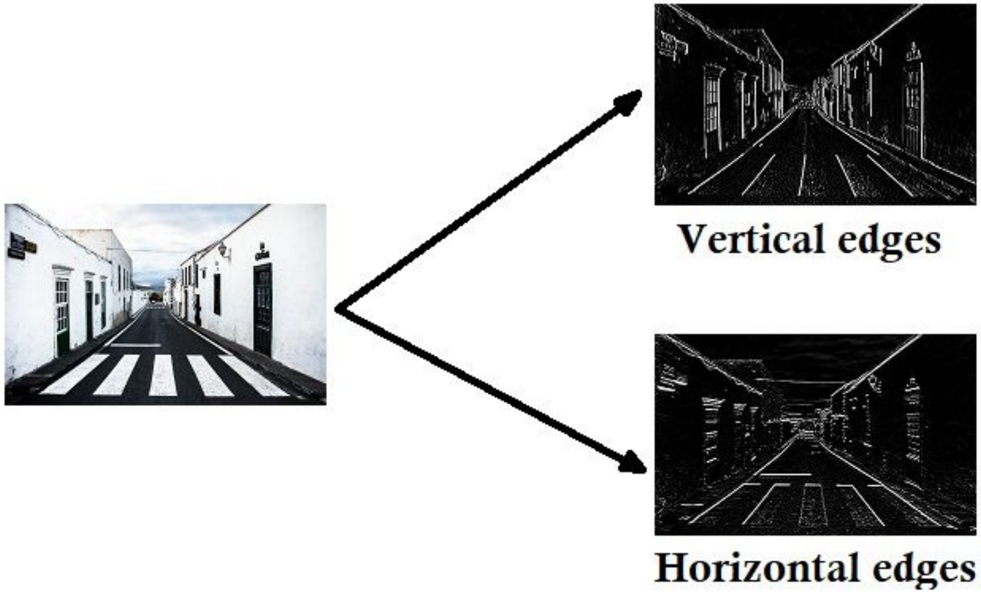
- Stacked regression models with a non-linear activation function
- 'Train' optimal weights
- Predict class of given input
- # parameters depends on image size

CNN building blocks |Convolutional layers (trained)

- Filter applied sequentially to blocks of pixels in an image, output single value per block
- 'Train' optimal filter elements
- Extract useful/differentiating features
- # parameters doesn't depend on image size



CNN building blocks | Examples of filters



1	0	-1
1	0	-1
1	0	-1

1	1	1
0	0	0
-1	-1	-1

CNN building blocks| Hyperparameters (tuned)

Pooling

- Apply filter to “summarise” input (usually considered part of conv layer)
- Defined by type of pooling (max pooling, average pooling)

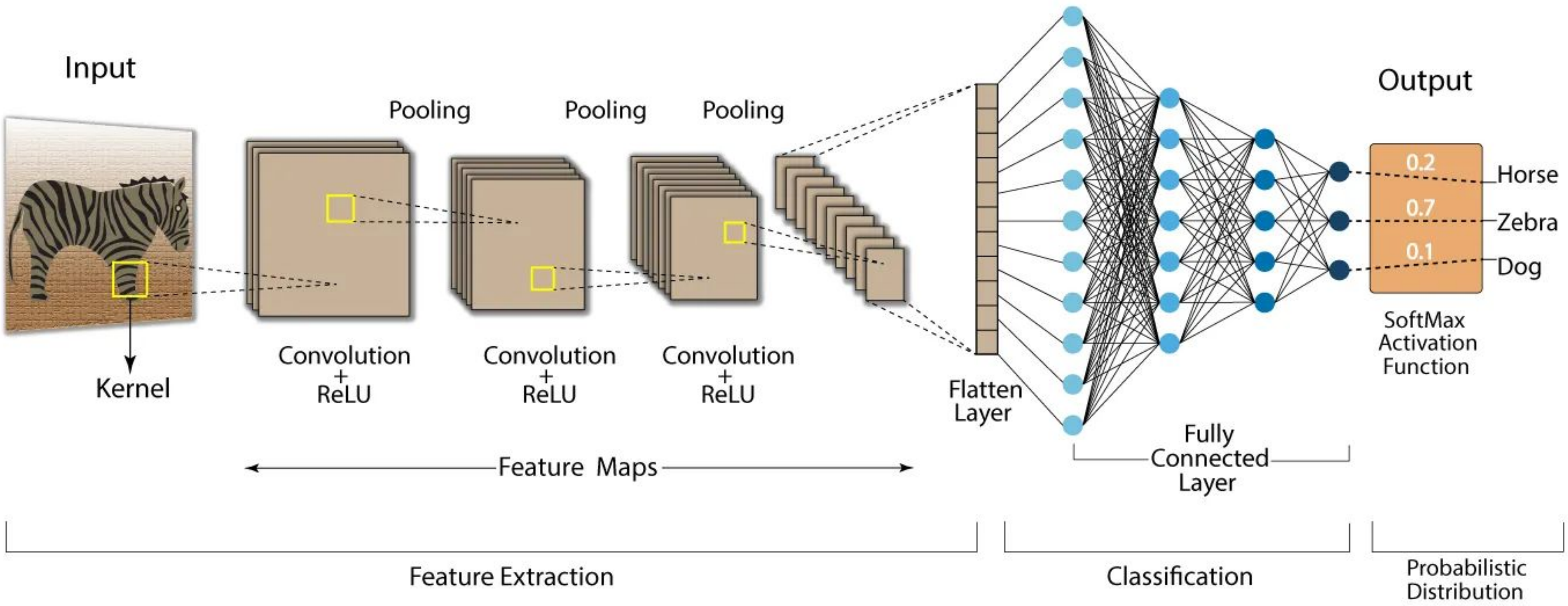
Stride

- Number of steps move when scanning image with filter

Padding

- Add with extra border of 0’s p elements wide
- Common types are “valid” (p=0) and “same” (input size = output size)

Convolution Neural Network (CNN)



Common approaches | Classification vs. regression

Classification-based approach

Implemented in two stages:

- First, select ROI in an image
- Second, classify ROI using CNNs

Slower as prediction runs in two stages and run predictions on every ROI, commonly used for once-off detection

Popular algorithms:

- R-CNNs (Fast, Faster, Mask)
- RetinaNet

Regression-based approach

Predict classes and bounding boxes for the whole image in one run of algorithm

Trade accuracy for large improvements in speed, commonly used for real-time object detection

Popular algorithms:

- You Only Look Once (YOLO)
- Single Shot Detector (SSD)

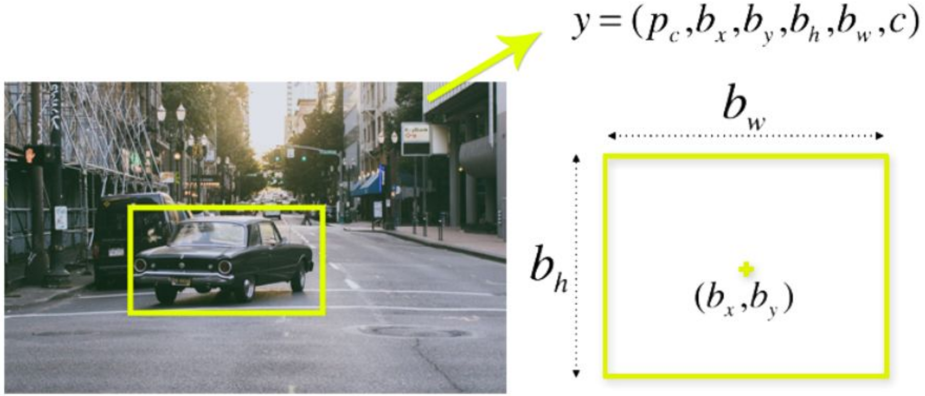
Deep-dive | You Only Look Once (YOLO)

What are we trying to predict?

We want to predict object class (c), prediction confidence (p_C), and object location, which is specified by:

- b_x : x-coord of center point
- b_y : y-coord of center point
- b_h: height of bounding box
- b_w : width of bounding box

If *m* objects are detected, output is a list of *m* vectors each with length *5 + # classes*



Deep-dive | You Only Look Once (YOLO)

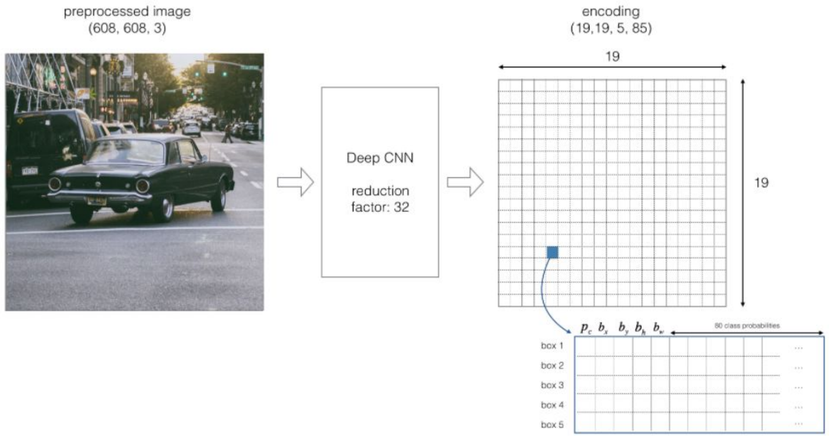
How does YOLO do this?

Pass image through conv layers, and then generate predictions:

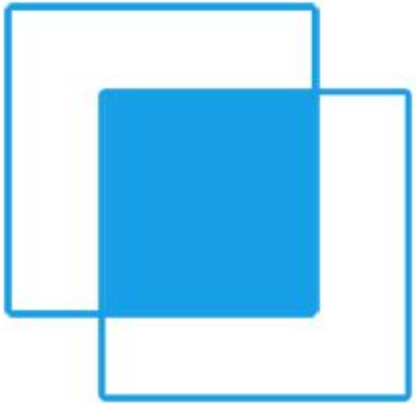
- Split image into cells (19 x 19)
- Predict *z* boxes for each cell (z = 5)
- ∴ *19 x 19 x z* output vectors

Remove empty boxes using non-max suppression:

- Drop all boxes with $p_c < 0.5$
- Sequentially select box with highest p_c and drop all overlapping boxes with IoU > 0.5



$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



```
   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
```

Prior work | We're doing something new :)

- 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, 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