CSE428: Image Processing

Lecture 17

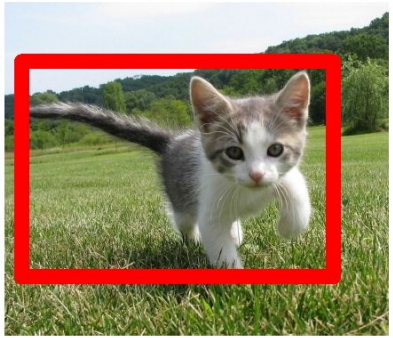
Object Detection (Cont.)

Image Recognition Problems

Classification

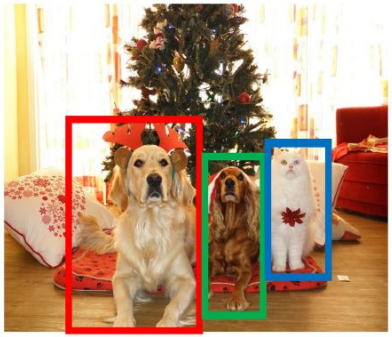
[{CAT}]

Classification + Localization



[{CAT, (x, y, h, w)}]

Object Detection

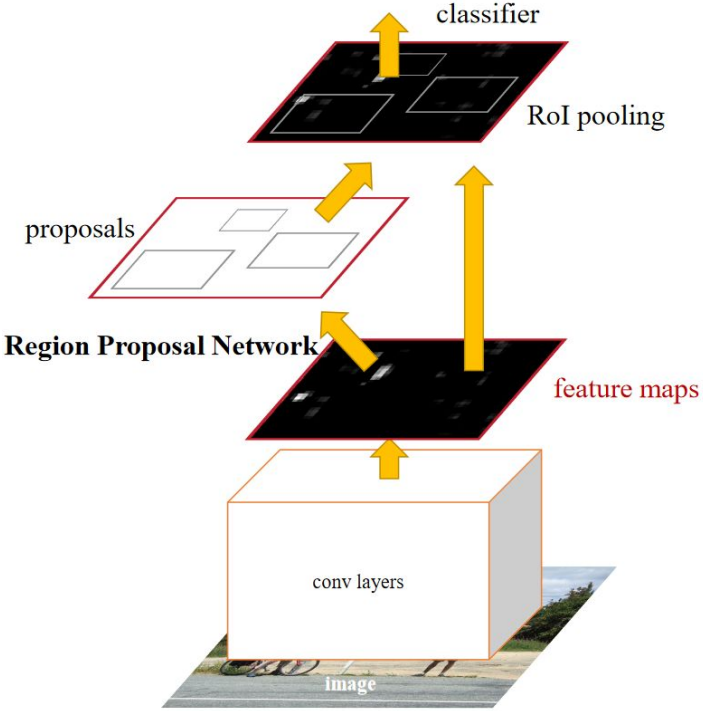


[{DOG, (x, y, h, w)},
{DOG, (x, y, h, w)},
{CAT, (x, y, h, w)}]

Faster R-CNN

Faster R-CNN

1. Regions of interest are generated using the region proposal network (RPN). To generate RoIs, the RPN uses convolutional layers.
2. The second part of Faster R-CNN is classification. It outputs the final bounding boxes and accepts two inputs—the list of RoIs from the previous step (RPN), and a feature volume computed from the input image.



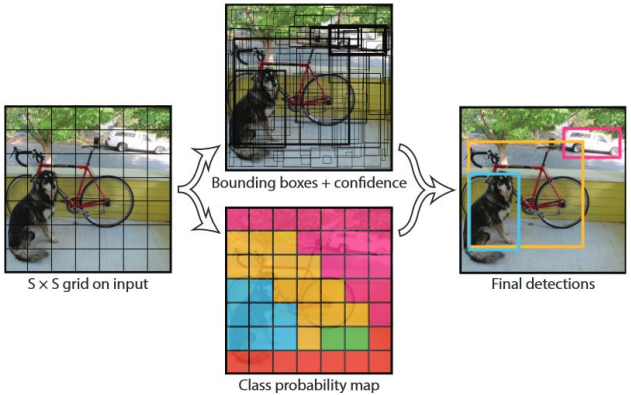
https://arxiv.org/abs/1506.01497

You Only Look Once (YOLO)

The core idea of YOLO is reframing object detection as a single regression problem.

We will divide the input into a S × S grid, as represented in this diagram, and for each part of the grid, we will define B bounding boxes. Then, our only task will be to predict the following for each bounding box:

- The center of the box
- The width and height of the box
- The probability that this box contains an object
- The class of said object



https://arxiv.org/abs/1506.02640

Inferring with YOLO

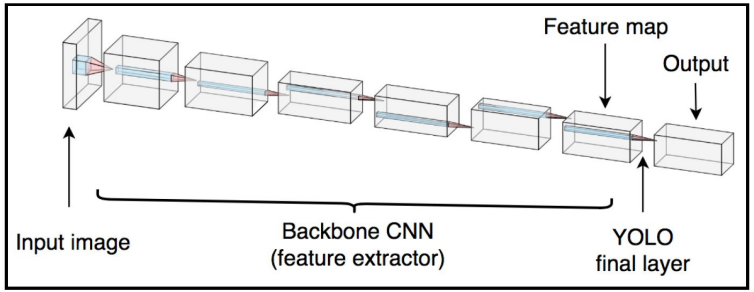
Split the model into two parts—inference and training.

Inference is the process of taking an image input and computing results.

Training is the process of learning the weights of the model. When implementing a model from scratch, inference cannot be used before the model is trained. But, for the sake of simplicity, we are going to start with inference.

The YOLO Backbone

YOLO is based on a backbone model. The role of this model is to extract meaningful features from the image that will be used by the final layers.

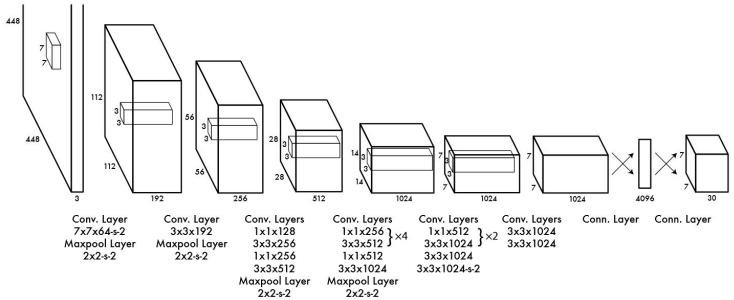


The final layer of the backbone outputs a feature volume of size w × h × D, where w × h is the size of the grid and D is the depth of the feature volume. For instance, for VGG-16, D = 512.

The YOLO Backbone

YOLO's final layer accepts the feature volume as an input. It is composed of convolutional filters of size 1 × 1. YOLO's final output is a **w × h × M** tensor, where **w × h** is the size of the grid, and **M** corresponds to the formula **B × (C + 5)**, where the following applies:

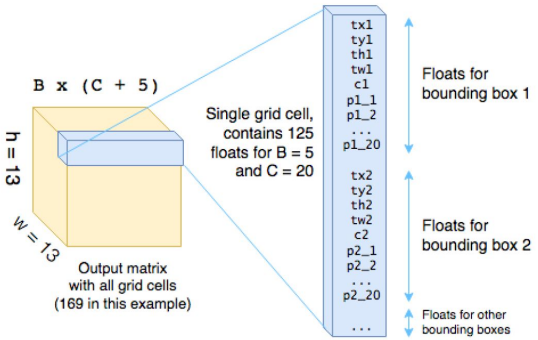
- **B** is the number of bounding boxes per grid cell.
- **C** is the number of classes (in our example, we will use 20 classes).



The YOLO Backbone Predictions

For each bounding box, we need to predict **(C + 5)** numbers:

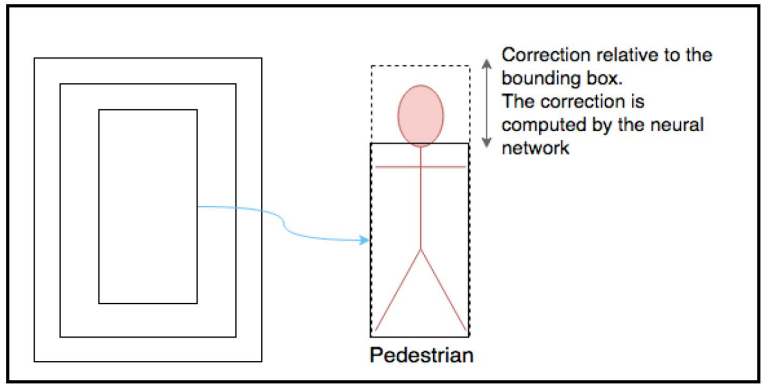
- **tx** and **ty** will be used to compute the coordinates of the center of the bounding box.
- **tw** and **th** will be used to compute the width and height of the bounding box.
- **c** is the confidence that an object is in the bounding box.
- **p1, p2, ..., and pC** are the probability that the bounding box contains an object of class **1, 2, ..., C**.



Concept: Anchor Boxes

Anchor boxes (also called **priors**) are a set of bounding box sizes that are decided upon before training the network.

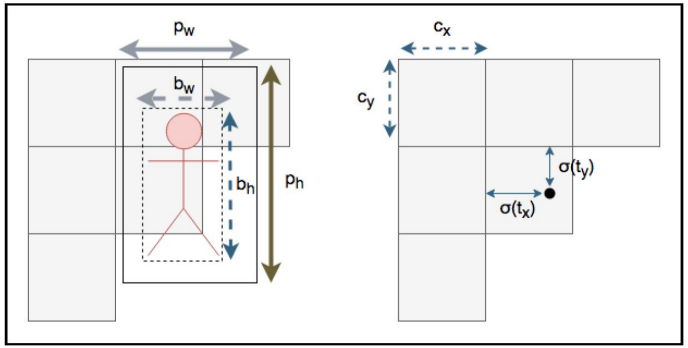
A set of anchor boxes is usually small—from 3 to 25 different sizes in practice. As those boxes cannot exactly match all the objects, the network is used to refine the closest anchor box.



Refining Anchor Boxes with Predictions

- **tx**, **ty**, **tw**, **th** are the outputs from the last layer.
- **bx**, **by**, **bw**, **bh** are the position and size of the predicted bounding box, respectively.
- **pw**, **ph** represent the original size of the anchor box.
- **cx** and **cy** are the coordinates of the current grid cell (they will be (0,0) for the top left box, (w - 1,0) for the top-right box, and (0, h - 1) for the bottom-left box).

$$
\begin{aligned}
b_x &= \text{sigmoid}(t_x) + c_x \\
b_y &= \text{sigmoid}(t_y) + c_y \\
b_w &= p_w \exp(t_w) \\
b_h &= p_h \exp(t_h)
\end{aligned}
$$



Making Sense of The Neural Network Output

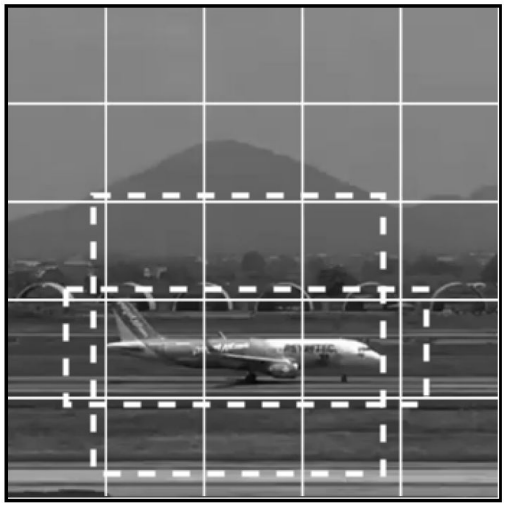
The output of the neural network, a matrix with raw numbers, needs to be transformed into a list of bounding boxes. A simplified version of the code would look like this:

```
boxes = []
for row in range(grid_height):
    for col in range(grid_width):
        for b in range(num_box):
            tx, ty, tw, th = network_output[row, col, b, :4]
            box_confidence = network_output[row, col, b, 4]
            classes_scores = network_output[row, col, b, 5:]

            bx = sigmoid(tx) + col
            by = sigmoid(ty) + row

            # anchor_boxes is a list of dictionaries containing the size of
each anchor
            bw = anchor_boxes[b]['w'] * np.exp(tw)
            bh = anchors_boxes[b]['h'] * np.exp(th)

            boxes.append((bx, by, bw, bh, box_confidence, classes_scores))
```

Post-processing the Boxes (Thresholding)

Multiply the confidence by the class probabilities and threshold them in order to only keep high probabilities

Here is an example of this operation with a simple sample, with a threshold of 0.3 and a box confidence (for this specific box) of 0.5:

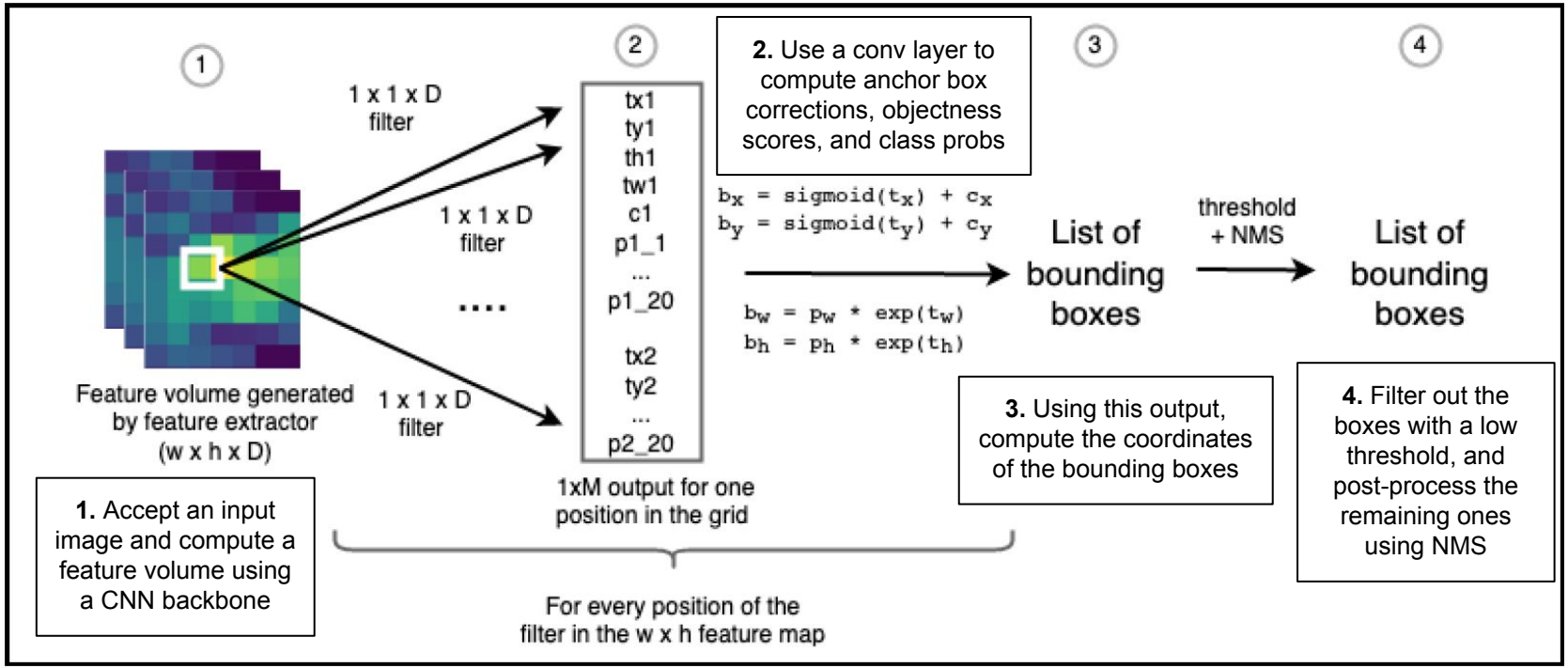


CLASS_LABELS	*dog*	*airplane*	*bird*	*elephant*
classes_scores	0.7	0.8	0.001	0.1
final_scores	0.35	0.4	0.0005	0.05
filtered_scores	0.35	0.4	0	0

Post-processing the Boxes (NMS)

Apply NMS to get rid of redundant boxes

Inferring with YOLO Summary



Training YOLO: Loss Function

Bounding box loss
The first part of the loss helps the network learn the weights to predict the bounding box coordinates and size

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

Object confidence loss
The second part of the loss teaches the network to learn the weights to predict whether a bounding box contains an object

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

Classification loss
The final part of the loss, the classification loss, ensures that the network learns to predict the proper class for each bounding box

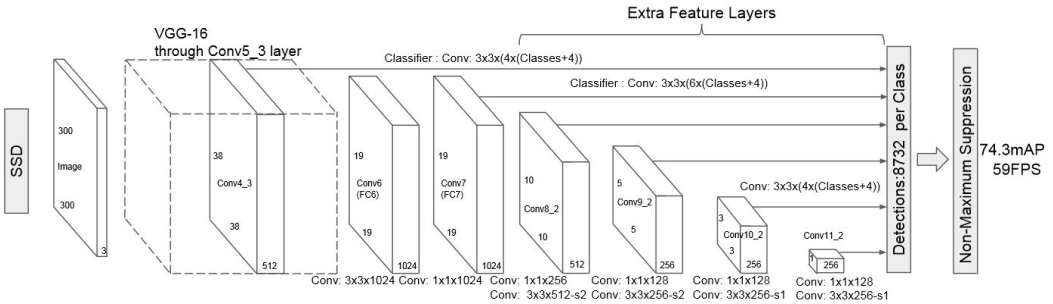
$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

Performance

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

Single Shot MultiBox Detector (SSD)

The core of SSD is predicting category scores and box offsets for a fixed set of default bounding boxes using small convolutional filters applied to feature maps.



Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	$\sim 1000 \times 600$
Fast YOLO	52.7	155	1	98	448×448
YOLO (VGG16)	66.4	21	1	98	448×448
SSD300	74.3	46	1	8732	300×300
SSD512	76.8	19	1	24564	512×512
SSD300	74.3	59	8	8732	300×300
SSD512	76.8	22	8	24564	512×512

https://arxiv.org/abs/1512.02325

Comparison

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [5]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [8]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [6]	Inception-ResNet-v2 [21]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [20]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [15]	DarkNet-19 [15]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [11, 3]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [3]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [9]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [9]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608×608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

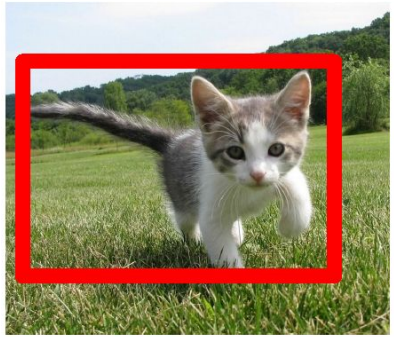
Segmentation

Image Recognition Problems

Classification

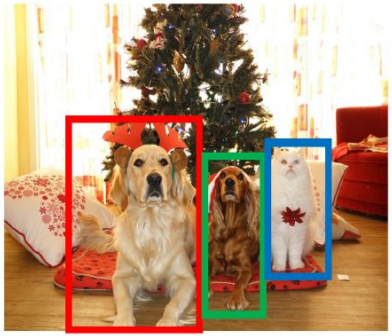
[{CAT}]

Classification + Localization



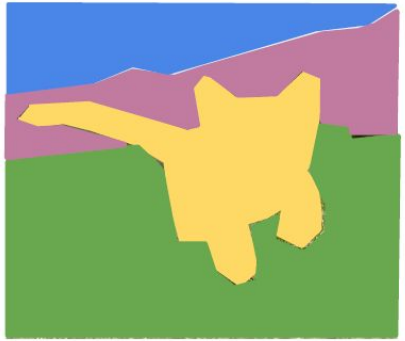
[{CAT, (x, y, h, w)}]

Object Detection



[{DOG, (x, y, h, w)},
{DOG, (x, y, h, w)},
{CAT, (x, y, h, w)}]

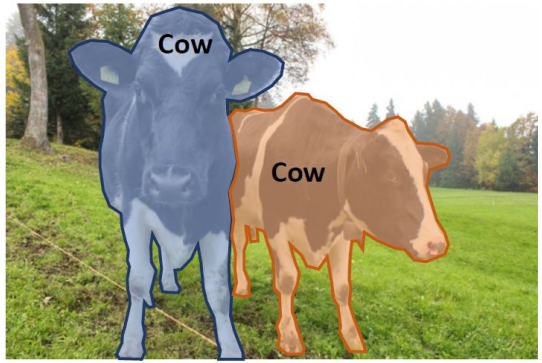
Segmentation

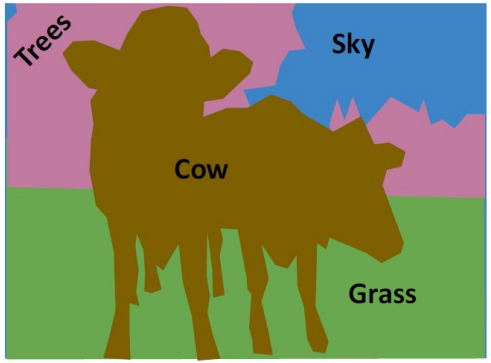


[{(0, 0), SKY},
{(M/2, N/2), CAT},
...,
{(M-1, N-1), GRASS}]

Groups of image segmentation

1. **Semantic segmentation** is an approach detecting, for every pixel, belonging class of the object.
2. **Instance segmentation** is an approach that identifies, for every pixel, a belonging instance of the object. It detects each distinct object of interest in the image.





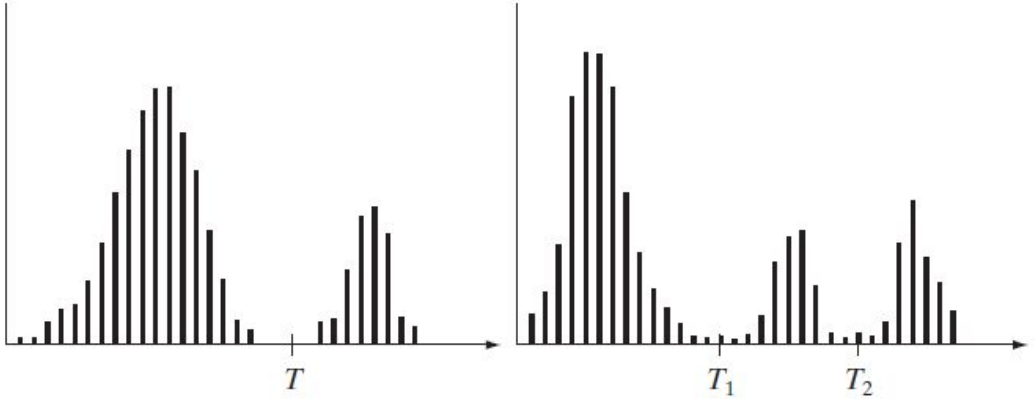
Classes of Segmentation Techniques

There are two classes of segmentation techniques:

1. Classical computer vision approaches
- Thresholding
- Watershed transformation
2. AI based techniques
- FCN
- UNet
- Mask R-CNN

Intensity Thresholding

Extract the objects from the background: select a threshold, that separates these modes. Then, any point in the image at which is called an object point; otherwise, the point is called a background point. Threshold could be global or local.

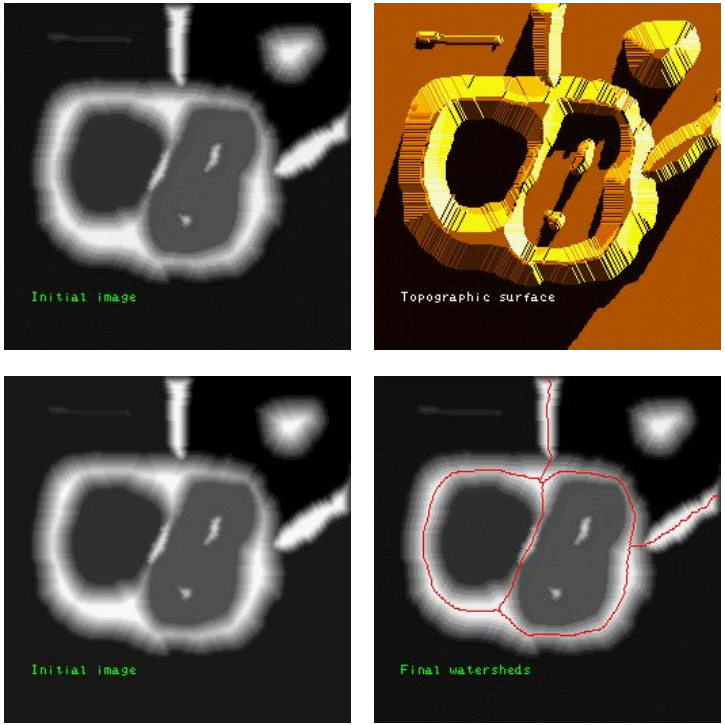


$$g(x,y) = \begin{cases} 1 & \text{if } f(x,y) > T \\ 0 & \text{if } f(x,y) \le T \end{cases} \qquad g(x,y) = \begin{cases} a & \text{if } f(x,y) > T_2 \\ b & \text{if } T_1 < f(x,y) \le T_2 \\ c & \text{if } f(x,y) \le T_1 \end{cases}$$

The Watershed Transformation

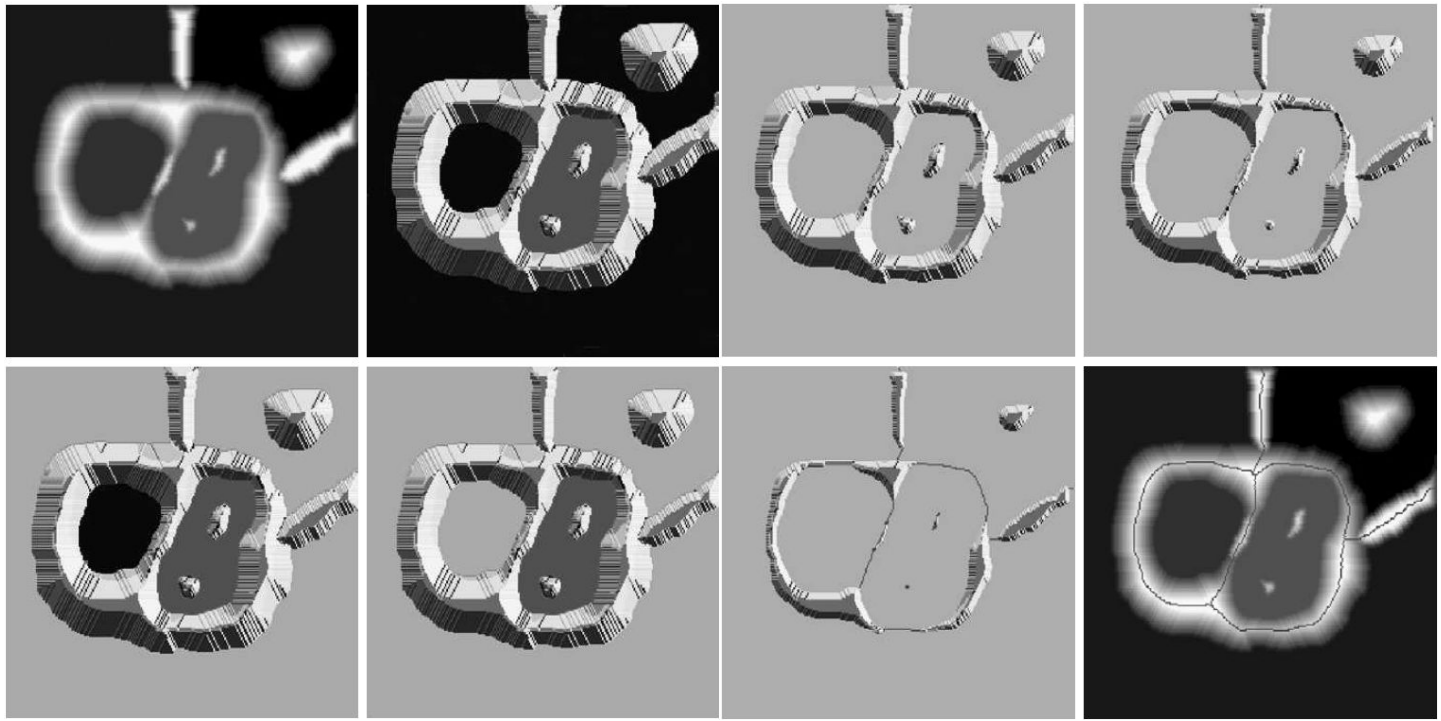
Any grayscale image can be considered as a topographic surface, where the intensity is referred to as the height at the spatial pixel point.

If we flood this surface from its minima and, if we prevent the merging of the waters coming from different sources, we partition the image into two different sets: the catchment basins and the watershed lines.



https://people.cmm.minesparis.psl.eu/users/beucher/wtshed.html

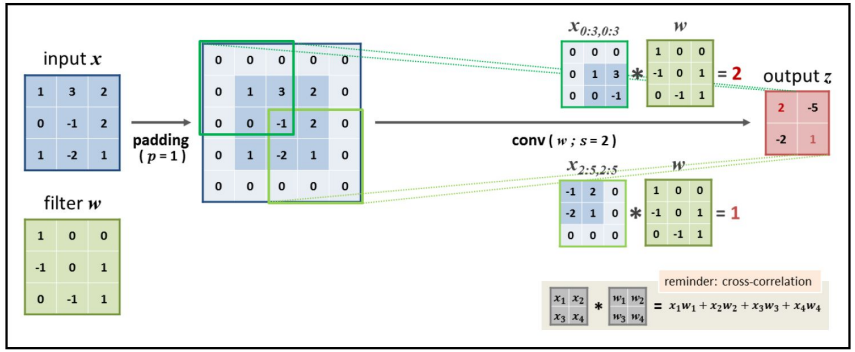
The Watershed Transformation



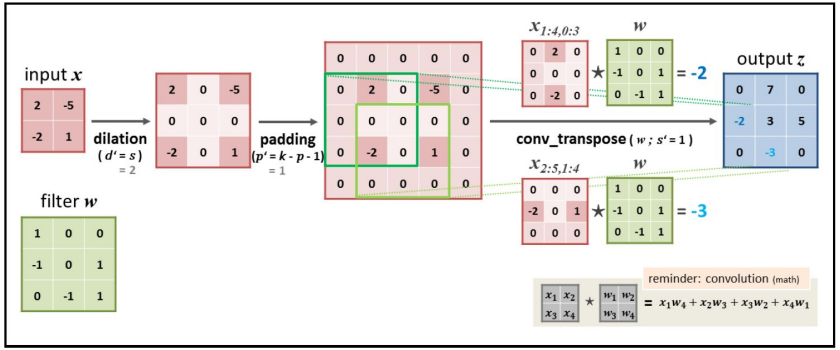
Concept: Transposed Convolution

Convolution Layer: decreasing spatial size (in a learnable way)

Transposed Convolution Layer: increasing spatial size (in a learnable way)



Convolution

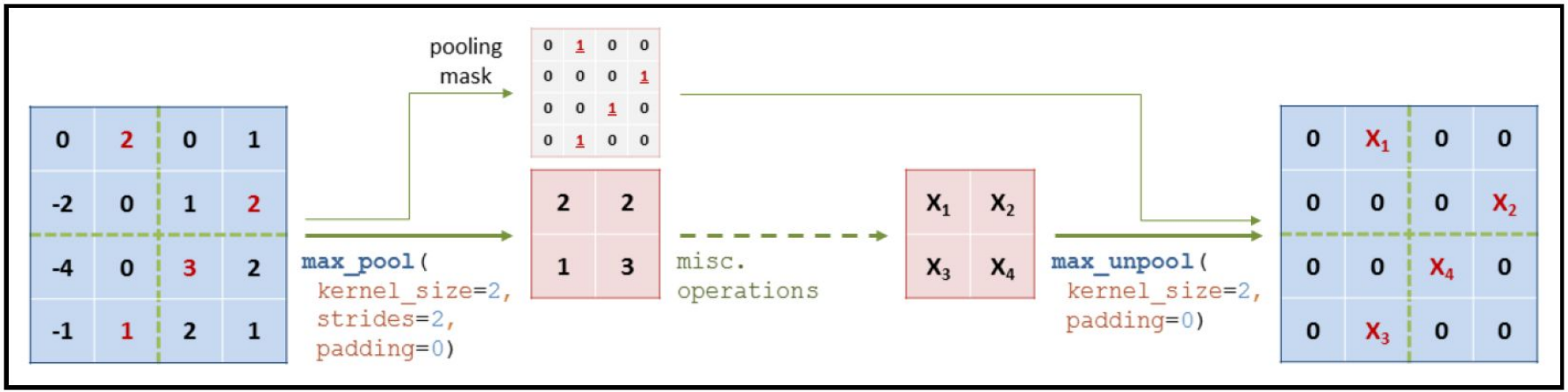


Transposed Convolution

Concept: Unpooling

Pooling Layer: decreasing spatial size

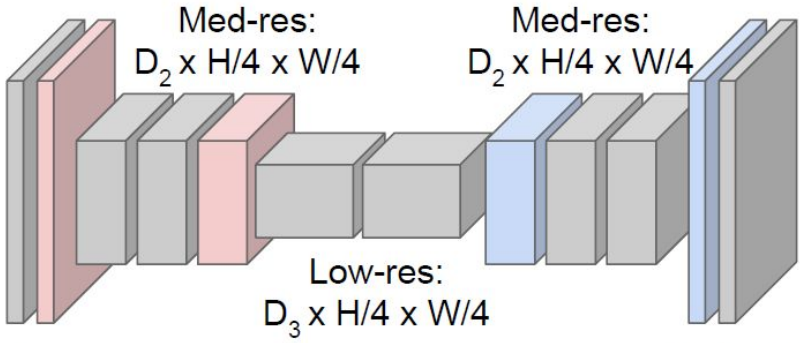
Unpooling Layer: increasing spatial size



Semantic Segmentation

Design network as a bunch of convolutional layers, with downsampling and upsampling inside the network

Input:
3 x H x W



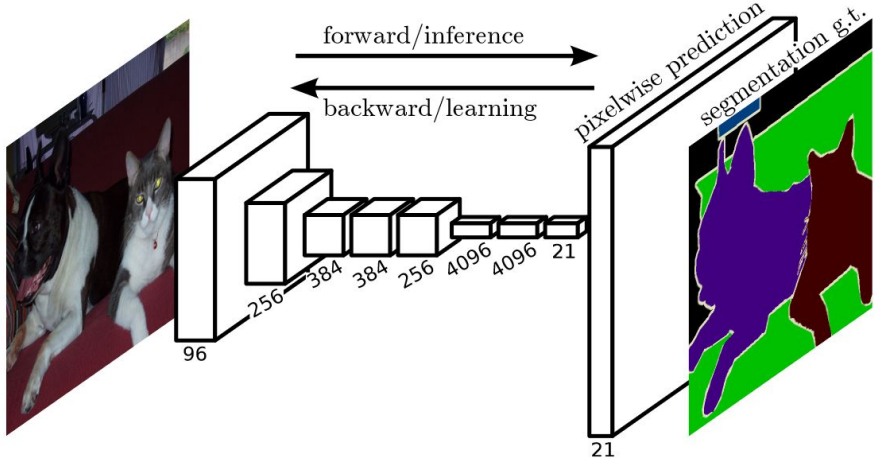
High-res:
D_1 x H/2 x W/2

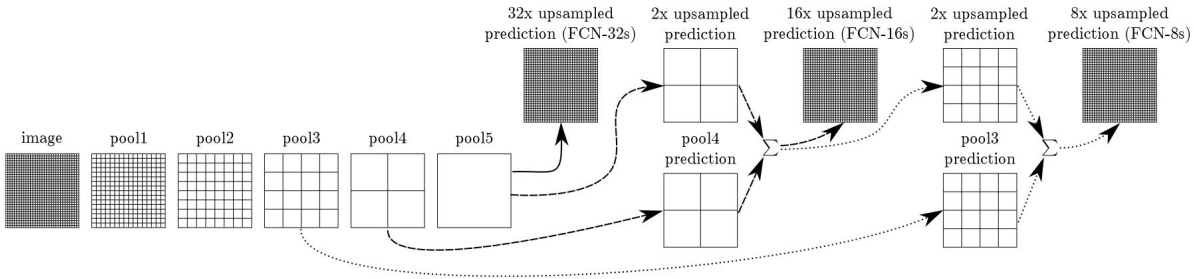
High-res:
D_1 x H/2 x W/2

Predictions:
H x W

Fully convolutional networks (FCN)

Fully convolutional network proposed by a team at UC Berkeley, in 2015, enables us to realize CNNs without any fully connected layers. This allows segmentation maps to be easily generated for input images of any size



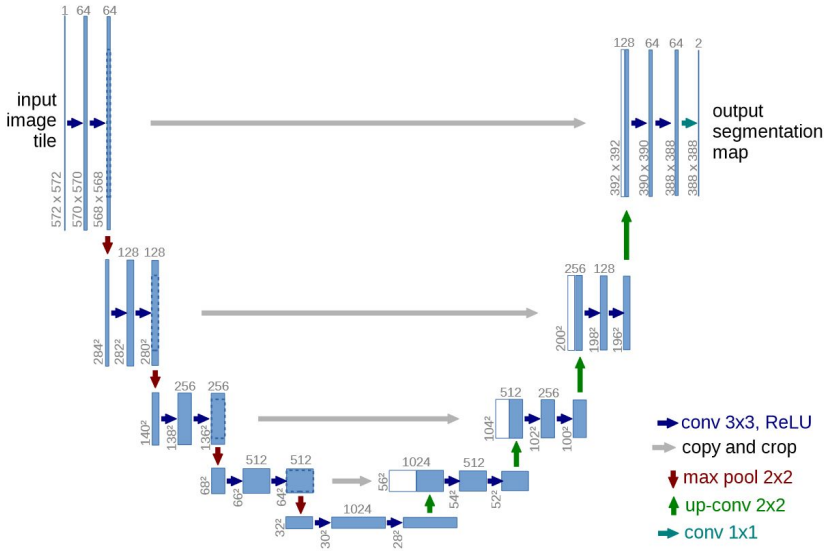


https://arxiv.org/abs/1411.4038

U-Net

A usual contracting network by successive layers, where pooling operators are replaced by upsampling operators. Hence, these layers increase the resolution of the output. In order to localize, high resolution features from the contracting path are combined with the upsampled output.

Works really well for biomedical image segmentation tasks.

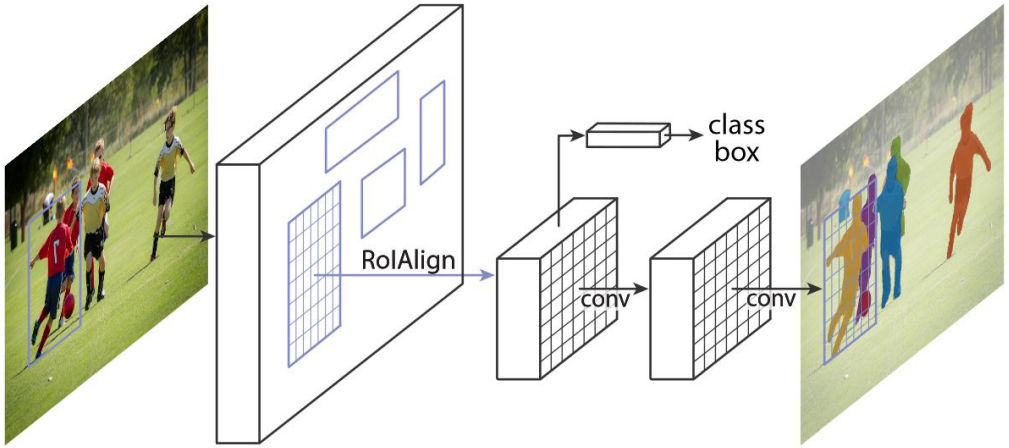


https://arxiv.org/abs/1505.04597

Mask R-CNN

Mask R-CNN extends **Faster R-CNN** by adding a branch for predicting an object mask in parallel with the existing branch for bounding box recognition.

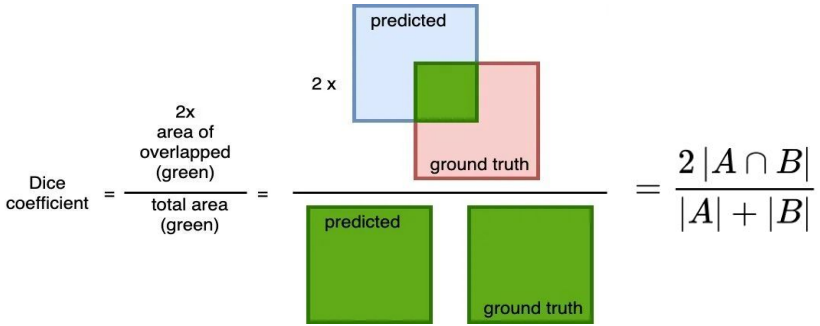
Moreover, Mask R-CNN is easy to generalize to other tasks such as to estimating human poses in the same framework.



https://arxiv.org/abs/1703.06870

Concept: Dice Coefficient

The index is known by several other names, especially Sørensen–Dice index, Sørensen index and Dice's coefficient. Other variations include the "similarity coefficient" or "index", such as Dice similarity coefficient (DSC).

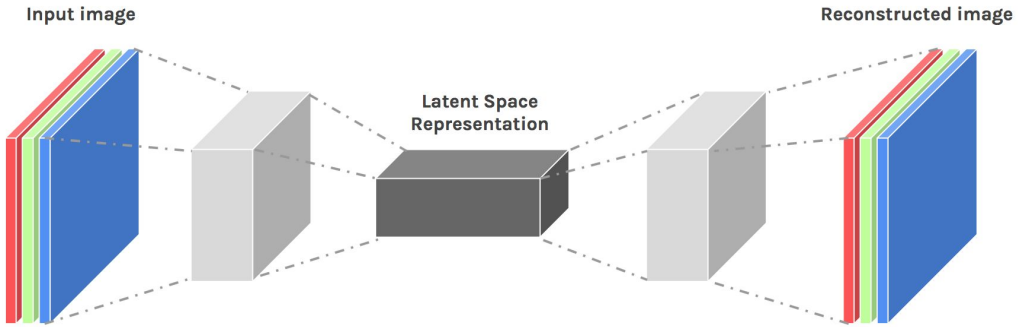


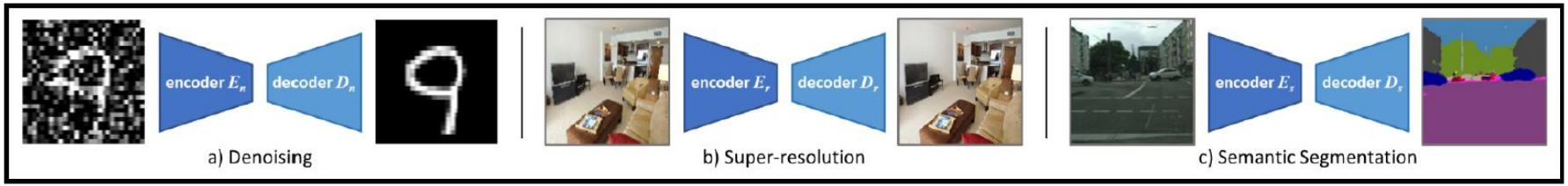
$$\mathrm{IoU}(A, B) = \frac{\mathrm{Dice}(A, B)}{2 - \mathrm{Dice}(A, B)} \quad ; \quad \mathrm{Dice}(A, B) = \frac{2 \times \mathrm{IoU}(A, B)}{1 + \mathrm{IoU}(A, B)}$$

Concept: Encoder-Decoder Architecture for Images

Encoder: reducing spatial size

Decoder: expanding spatial size





Resources

1. https://cs231n.github.io/convolutional-networks/
2. https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/
3. https://www.tensorflow.org/api_docs/python/tf/keras
4. Deep Learning with Python Book by François Chollet
5. https://www.deeplearningbook.org/
6. https://pjreddie.com/darknet/yolo/
7. https://blog.roboflow.com/guide-to-yolo-models/
8. **Hands-on Computer Vision with TensorFlow 2 by Eliot Andres & Benjamin Planche (Packt Pub.)**