Deep Learning

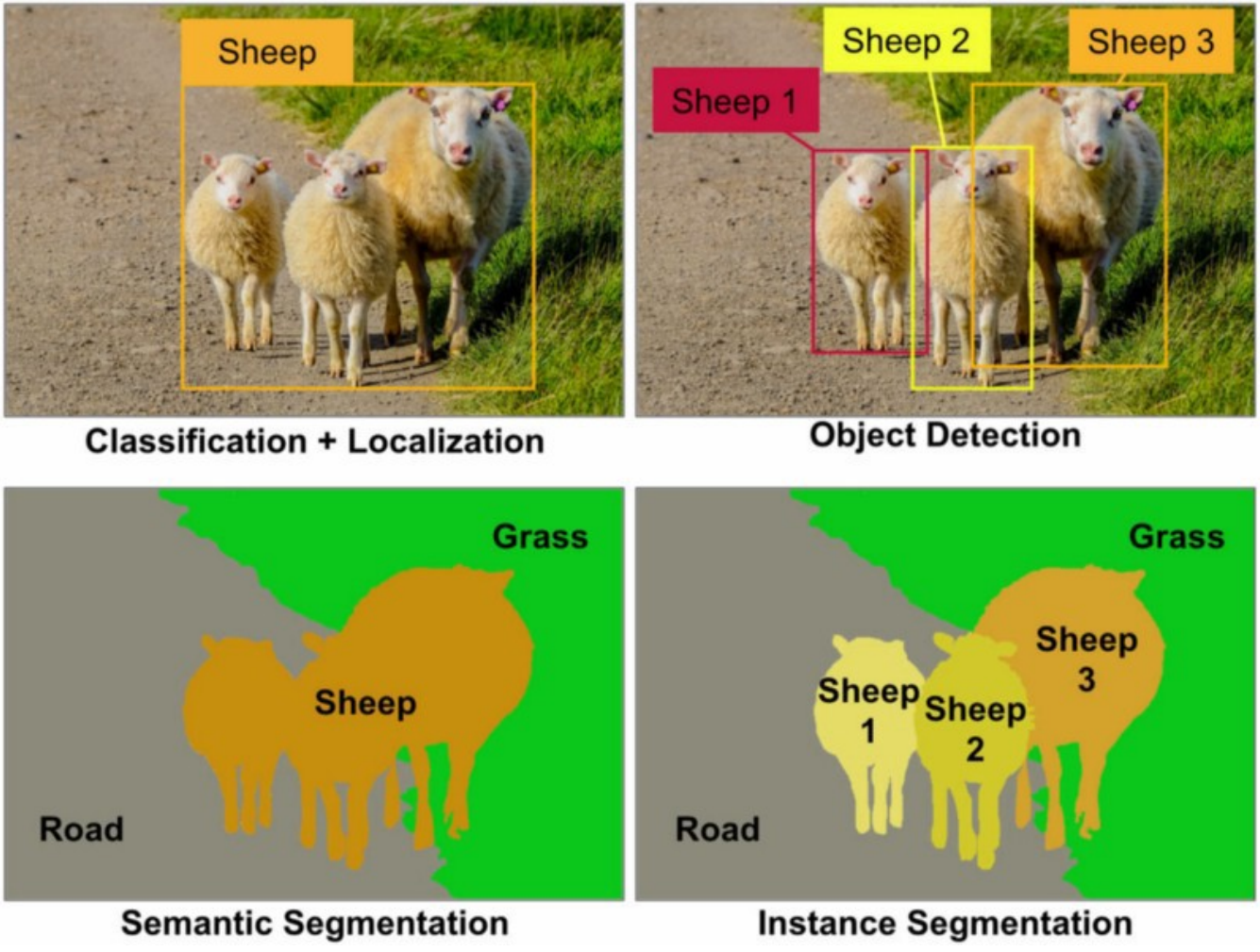
Lecture 6: Computer vision

Prof. Gilles Louppe
g.louppe@uliege.be

Today

How to build neural networks for (some) advanced computer vision tasks.

- Classification
- Object detection
- Segmentation



—

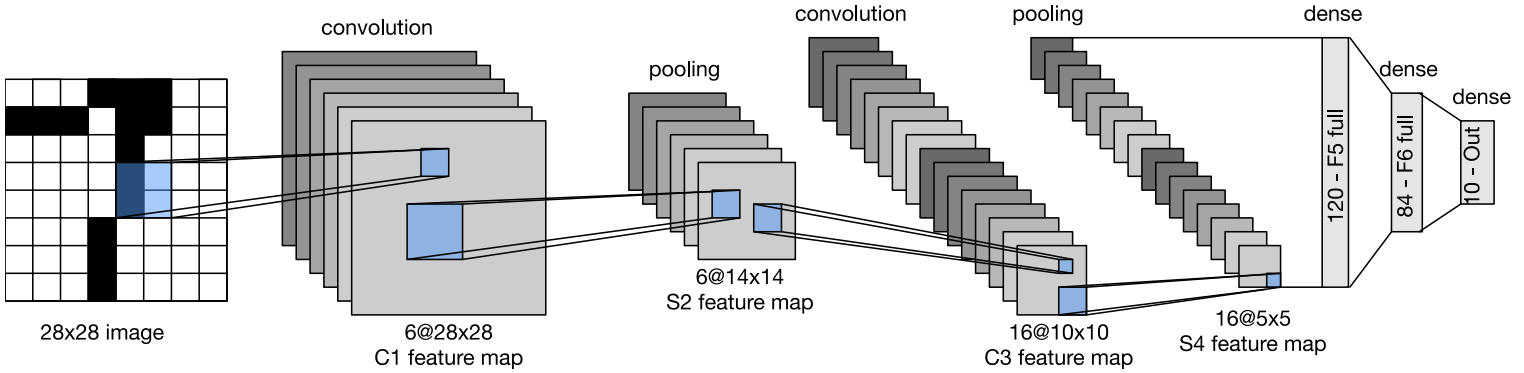
Credits: Aurélien Géron, 2018.

Classification

A few tips when using convnets for classifying images.

Convolutional neural networks

- Convolutional neural networks combine convolution, pooling and fully connected layers.
- They achieve state-of-the-art results for spatially structured data, such as images, sound or text.



—

Credits: Dive Into Deep Learning, 2020.

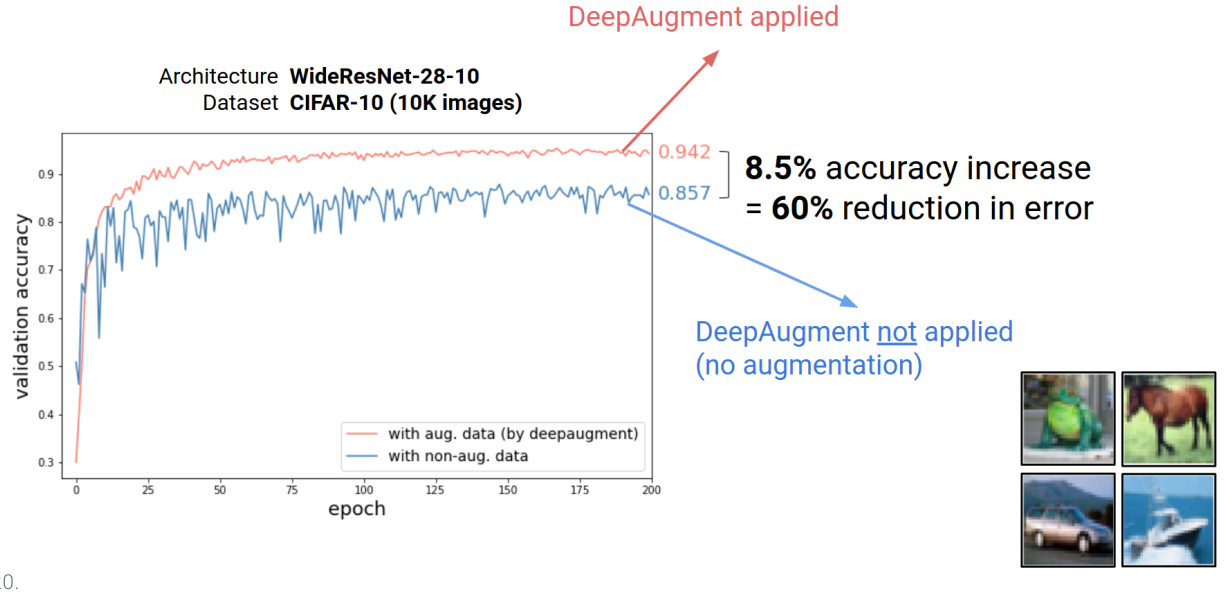
For classification,

- the activation in the output layer is a Softmax activation producing a vector $\mathbf{h} \in \triangle^C$ of probability estimates $P(Y = i|\mathbf{x})$, where C is the number of classes;
- the loss function is the cross-entropy loss.

Image augmentation

The lack of data is the biggest limit to the performance of deep learning models.

- Collecting more data is usually expensive and laborious.
- Synthesizing data is complicated and may not represent the true distribution.
- Augmenting the data with base transformations is simple and efficient.



—

Credits: DeepAugment, 2020.

Pre-trained models

- Training a model on natural images, from scratch, takes days or weeks.
- Many models pre-trained on large datasets are publicly available for download. These models can be used as feature extractors or for smart initialization.
- The models themselves should be considered as generic and re-usable assets.

Transfer learning

- Take a pre-trained network, remove the last layer(s) and then treat the rest of the network as a fixed feature extractor.
- Train a model from these features on a new task.
- Often better than handcrafted feature extraction for natural images, or better than training from data of the new task only.

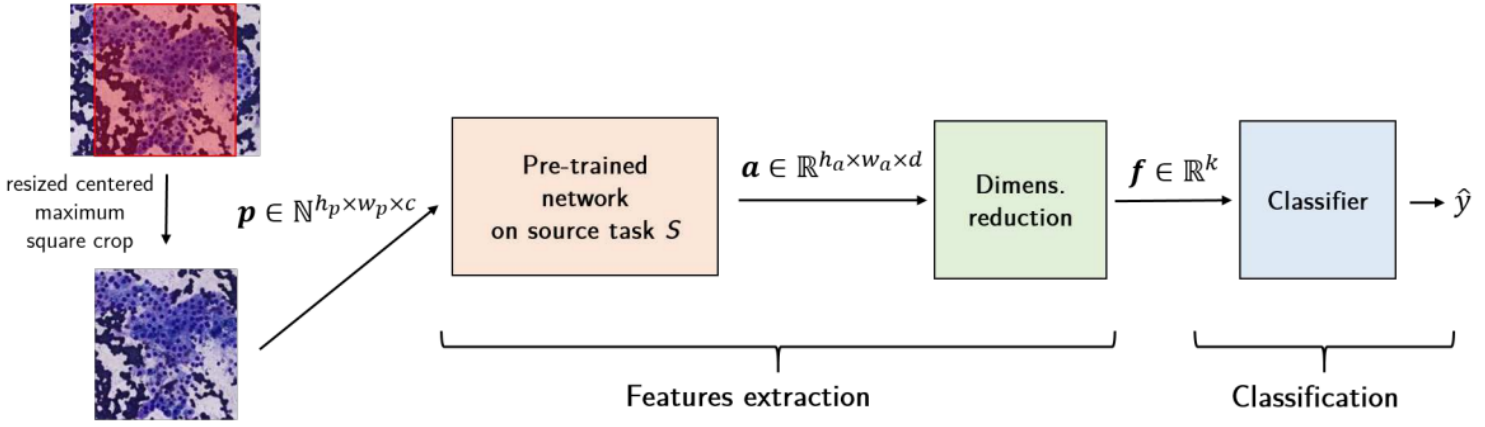
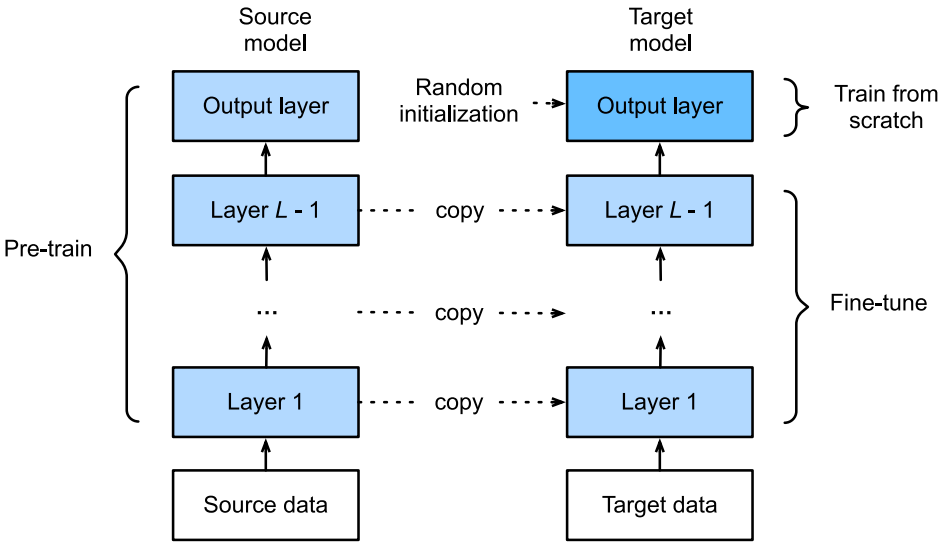


Figure 2. Feature extraction from pre-trained convolutional neural networks

—

Credits: Mormont et al, Comparison of deep transfer learning strategies for digital pathology, 2018.



Fine-tuning

Same as for transfer learning, but also fine-tune the weights of the pre-trained network by training the whole network on the new task.

—

Credits: Dive Into Deep Learning, 2020.

For models pre-trained on ImageNet, transferred/fine-tuned networks usually work even when the input images are from a different domain (e.g., biomedical images, satellite images or paintings).

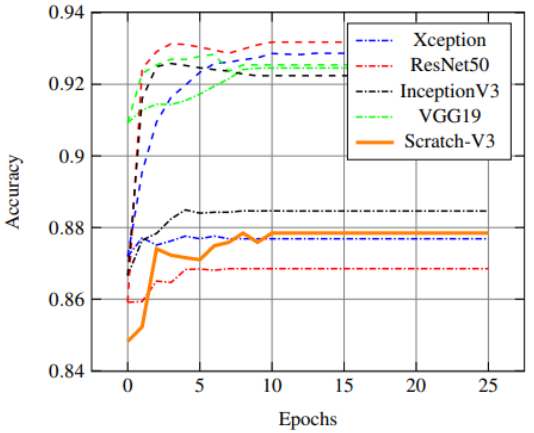


Fig. 2: Comparison between the fine tuning approach versus the off the shelf one when classifying the material of the heritage objects of the Rijksmuseum dataset. We observe how the first approach (as reported by the the dashed lines) leads to significant improvements when compared to the latter one (reported by the dash-dotted lines) for three out of four neural architectures. Furthermore, we can also observe how training a DCNN from scratch leads to worse results when compared to fine-tuned architectures which have been pre-trained on ImageNet (solid orange line).

—

Credits: Matthia Sabatelli et al, Deep Transfer Learning for Art Classification Problems, 2018.

Object detection

The simplest strategy to move from image classification to object detection is to classify local regions, at multiple scales and locations.

Parsing at fixed scale

Final list of detections

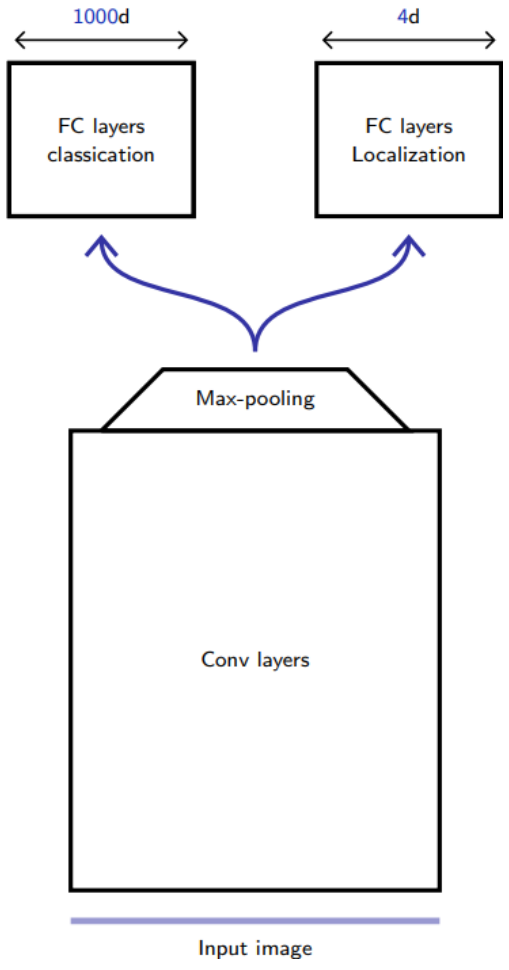
—

The sliding window approach evaluates a classifier at a large number of locations and scales.

This approach is usually **computationally expensive** as performance directly depends on the resolution and number of the windows fed to the classifier (the more the better, but also the more costly).

OverFeat

The complexity of the sliding window approach was mitigated in the pioneer OverFeat network (Sermanet et al, 2013) by adding a regression head to predict the object bounding box (x, y, w, h).



For each location and scale pre-defined from a **coarse** grid,

- the classifier head outputs a class and a confidence (left);
- the regression head predicts the location of the object (right).

—

Credits: Sermanet et al, 2013.

These bounding boxes are finally merged by **Non-Maximum Suppression** to produce the final predictions over a small number of objects.

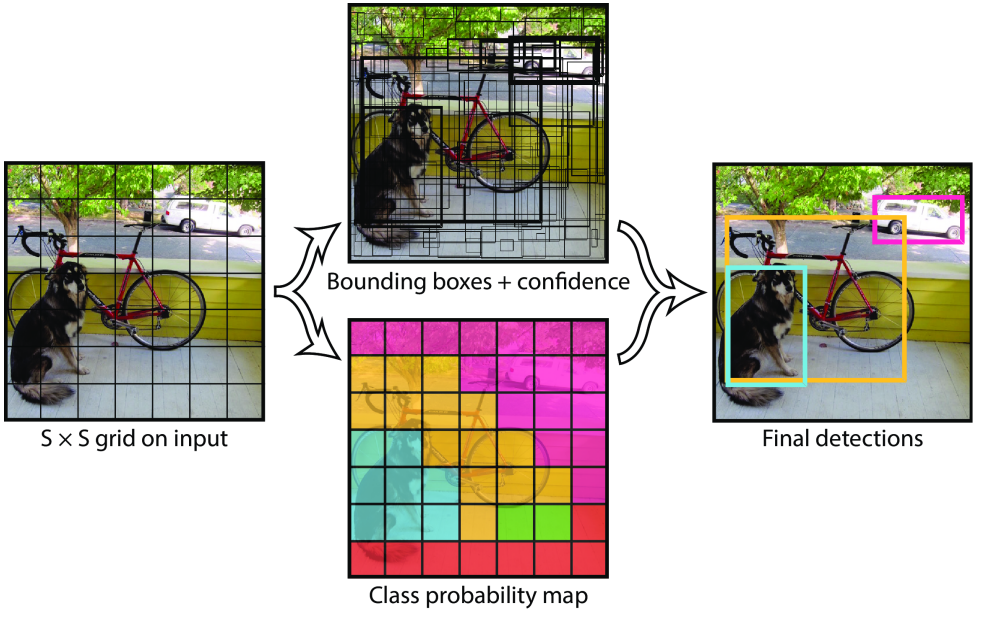
—

Credits: Sermanet et al, 2013.

The OverFeat architecture comes with several drawbacks:

- it is a disjoint system (2 disjoint heads with their respective losses, ad-hoc merging procedure);
- it optimizes for localization rather than detection;
- it cannot reason about global context and thus requires significant post-processing to produce coherent detections.

YOLO



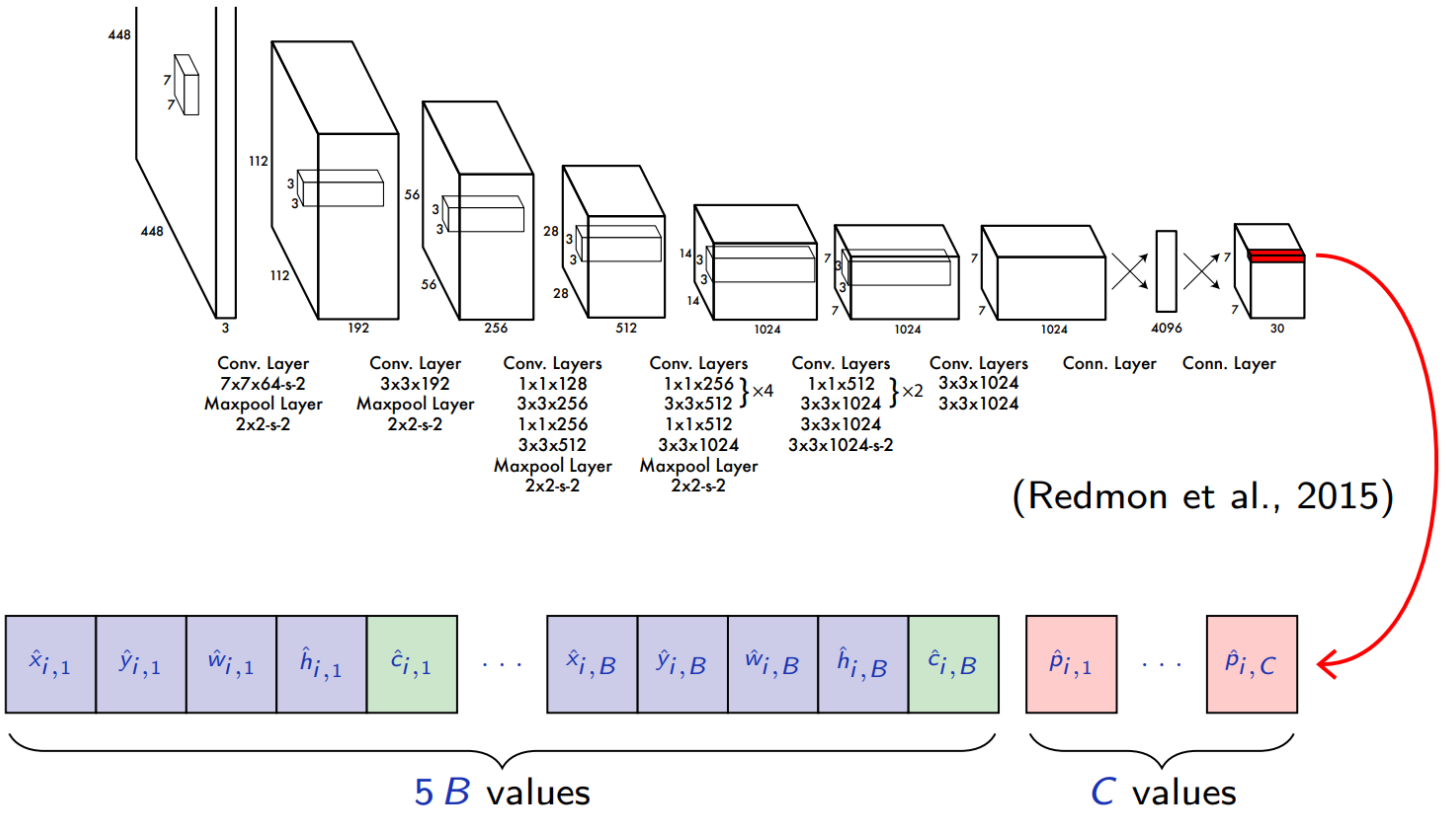
YOLO (Redmon et al, 2015) models detection as a regression problem.

The image is divided into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (5B + C)$ tensor.

—

Credits: Redmon et al, 2015.

For $S = 7, B = 2, C = 20$, the network predicts a vector of size 30 for each cell.



The network predicts class scores and bounding-box regressions, and **although the output comes from fully connected layers, it has a 2D structure**.

- Unlike sliding window techniques, YOLO is therefore capable of reasoning globally about the image when making predictions.
- It sees the entire image during training and test time, so it implicitly encodes contextual information about classes as well as their appearance.

During training, YOLO makes the assumptions that any of the $S \times S$ cells contains at most (the center of) a single object. We define for every image, cell index $i = 1, ..., S \times S$, predicted box $j = 1, ..., B$ and class index $c = 1, ..., C$,

- 1_i^{obj} is 1 if there is an object in cell i, and 0 otherwise;
- $1_{i,j}^{\text{obj}}$ is 1 if there is an object in cell i and predicted box j is the most fitting one, and 0 otherwise;
- $p_{i,c}$ is 1 if there is an object of class c in cell i, and 0 and otherwise;
- x_i, y_i, w_i, h_i the annoted bouding box (defined only if $1_i^{\text{obj}} = 1$, and relative in location and scale to the cell);
- $c_{i,j}$ is the IoU between the predicted box and the ground truth target.

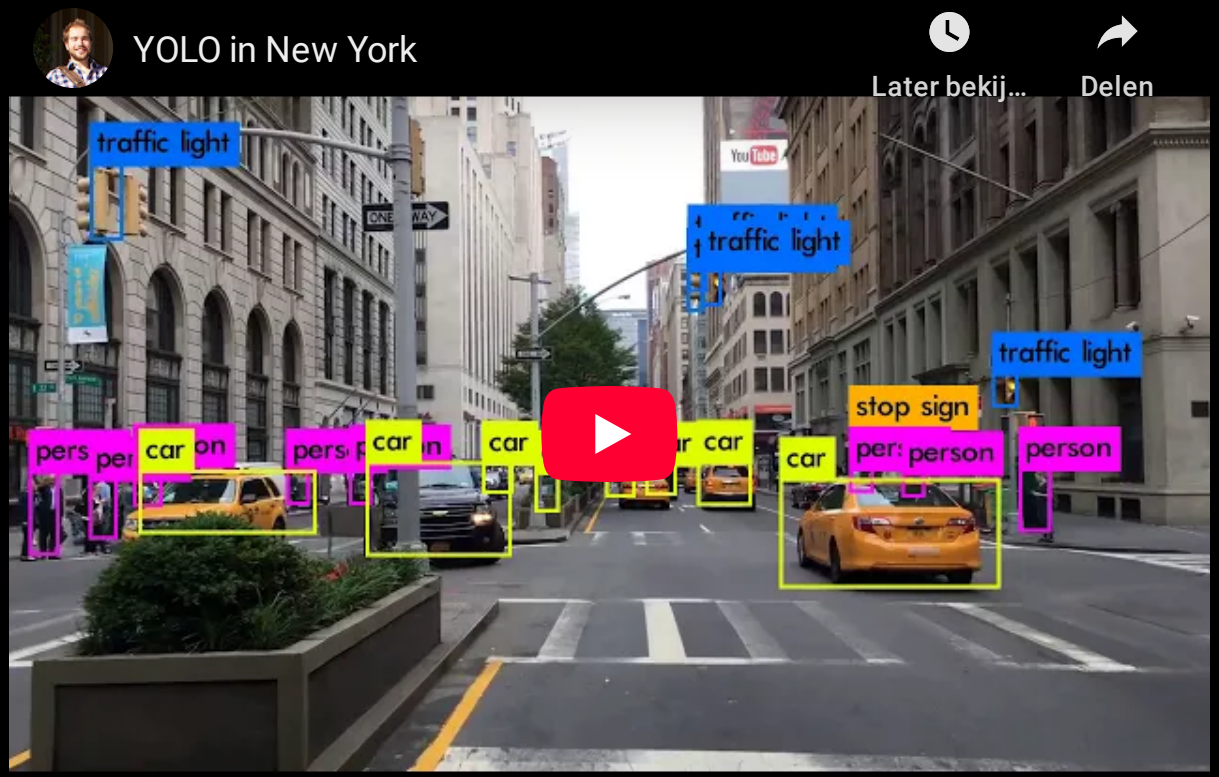
The training procedure first computes on each image the value of the $1_{i,j}^{\text{obj}}$'s and $c_{i,j}$, and then does one step to minimize the multi-part loss function

$$\lambda_{\text{coord}} \sum_{i=1}^{S \times S} \sum_{j=1}^{B} 1_{i,j}^{\text{obj}} \left((x_i - \hat{x}_{i,j})^2 + (y_i - \hat{y}_{i,j})^2 + (\sqrt{w_i} - \sqrt{\hat{w}_{i,j}})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_{i,j}})^2 \right)$$
$$+ \lambda_{\text{obj}} \sum_{i=1}^{S \times S} \sum_{j=1}^{B} 1_{i,j}^{\text{obj}} (c_{i,j} - \hat{c}_{i,j})^2 + \lambda_{\text{noobj}} \sum_{i=1}^{S \times S} \sum_{j=1}^{B} (1 - 1_{i,j}^{\text{obj}}) \, \hat{c}_{i,j}^2$$
$$+ \lambda_{\text{classes}} \sum_{i=1}^{S \times S} 1_{i}^{\text{obj}} \sum_{c=1}^{C} (p_{i,c} - \hat{p}_{i,c})^2$$

where $\hat{p}_{i,c}$, $\hat{x}_{i,j}$, $\hat{y}_{i,j}$, $\hat{w}_{i,j}$, $\hat{h}_{i,j}$ and $\hat{c}_{i,j}$ are the network outputs.

Training YOLO relies on **many engineering choices** that illustrate well how involved is deep learning in practice:

- pre-train the 20 first convolutional layers on ImageNet classification;
- use 448×448 input for detection, instead of 224×224;
- use Leaky ReLUs for all layers;
- dropout after the first convolutional layer;
- normalize bounding boxes parameters in $[0, 1]$;
- use a quadratic loss not only for the bounding box coordinates, but also for the confidence and the class scores;
- reduce weight of large bounding boxes by using the square roots of the size in the loss;
- reduce the importance of empty cells by weighting less the confidence-related loss on them;
- data augmentation with scaling, translation and HSV transformation.



YOLO (Redmon, 2017).

Region-based CNNs

An alternative strategy to having a huge predefined set of box proposals is to rely on region proposals first extracted from the image.

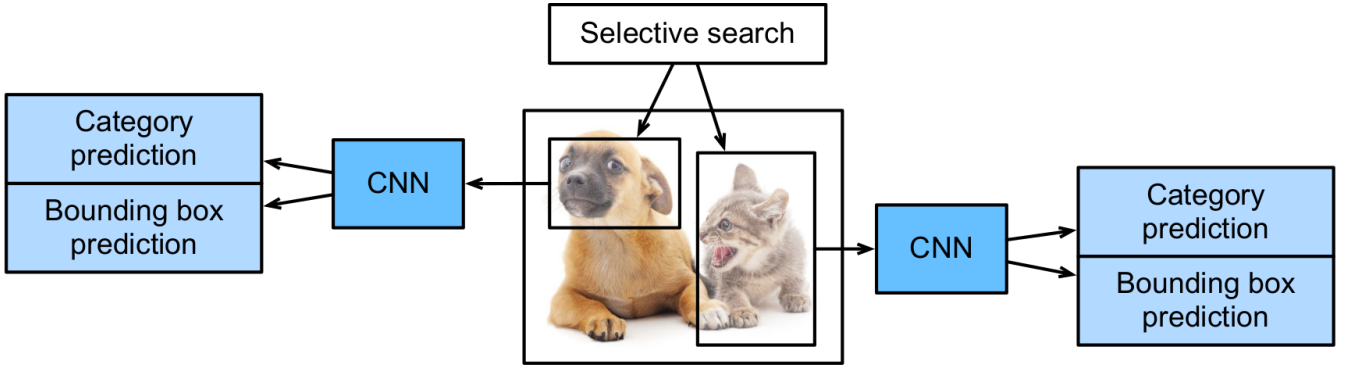
The main family of architectures following this principle are region-based convolutional neural networks:

- (Slow) R-CNN (Girshick et al, 2014)
- Fast R-CNN (Girshick et al, 2015)
- Faster R-CNN (Ren et al, 2015)
- Mask R-CNN (He et al, 2017)

R-CNN

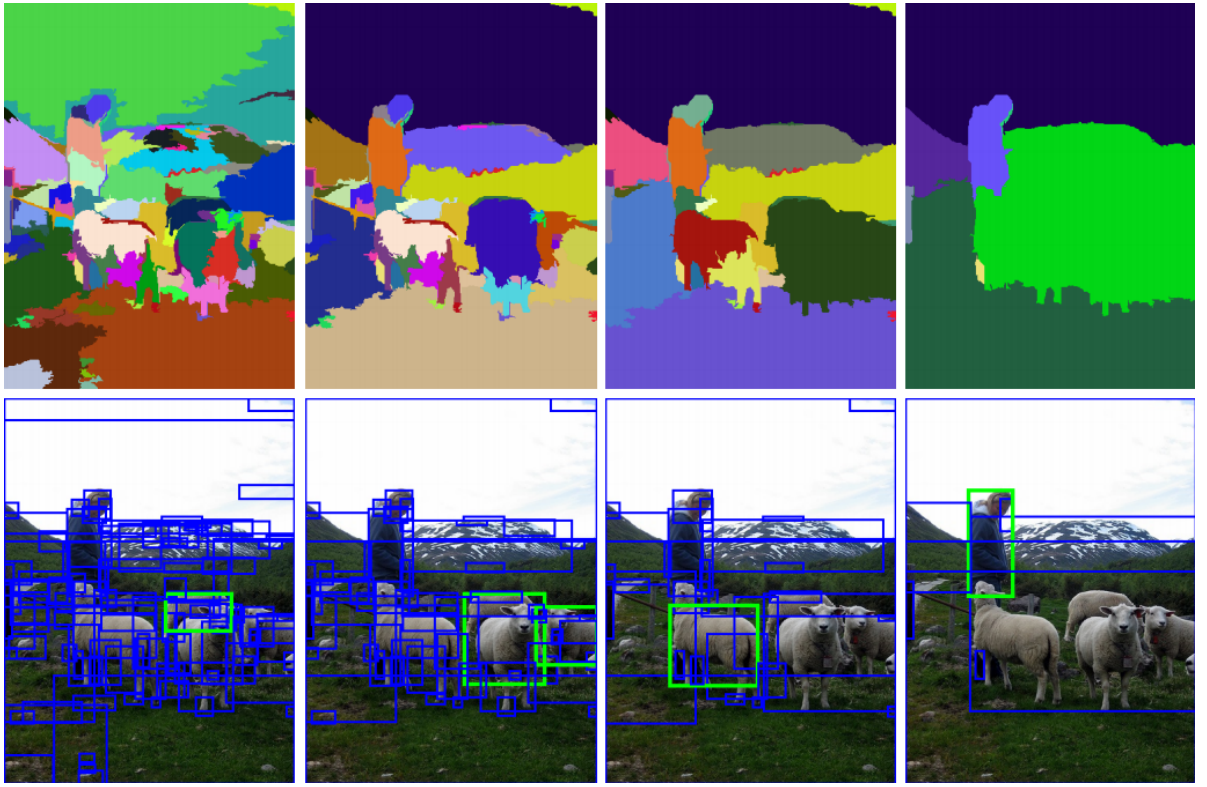
This architecture is made of four parts:

1. Selective search is performed on the input image to select multiple high-quality region proposals.
2. A pre-trained CNN (the backbone) is selected and put before the output layer. It resizes each proposed region into the input dimensions required by the network and uses a forward pass to output features for the proposals.
3. The features are fed to an SVM for predicting the class.
4. The features are fed to a linear regression model for predicting the bounding-box.



—

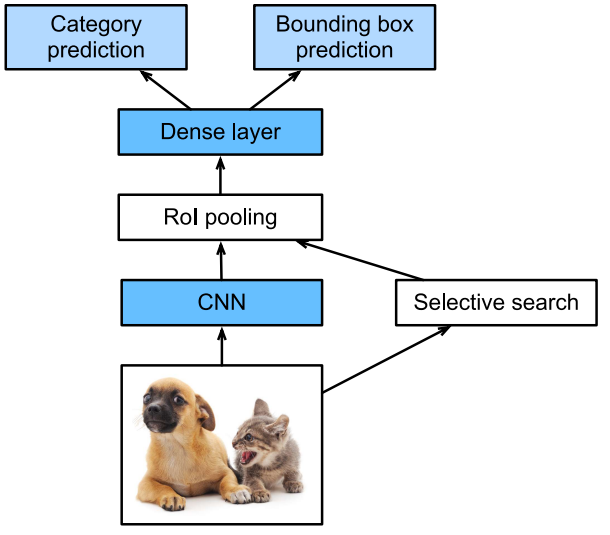
Credits: Dive Into Deep Learning, 2020.



Selective search (Uijlings et al, 2013) groups adjacent pixels of similar texture, color, or intensity by analyzing windows of different sizes in the image.

Fast R-CNN

- The main performance bottleneck of R-CNN is the need to independently extract features for each proposed region.
- Fast R-CNN uses the entire image as input to the CNN for feature extraction, rather than each proposed region.
- Fast R-CNN introduces RoI pooling for producing feature vectors of fixed size from region proposals of different sizes.



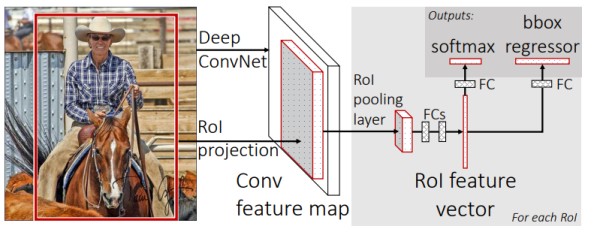
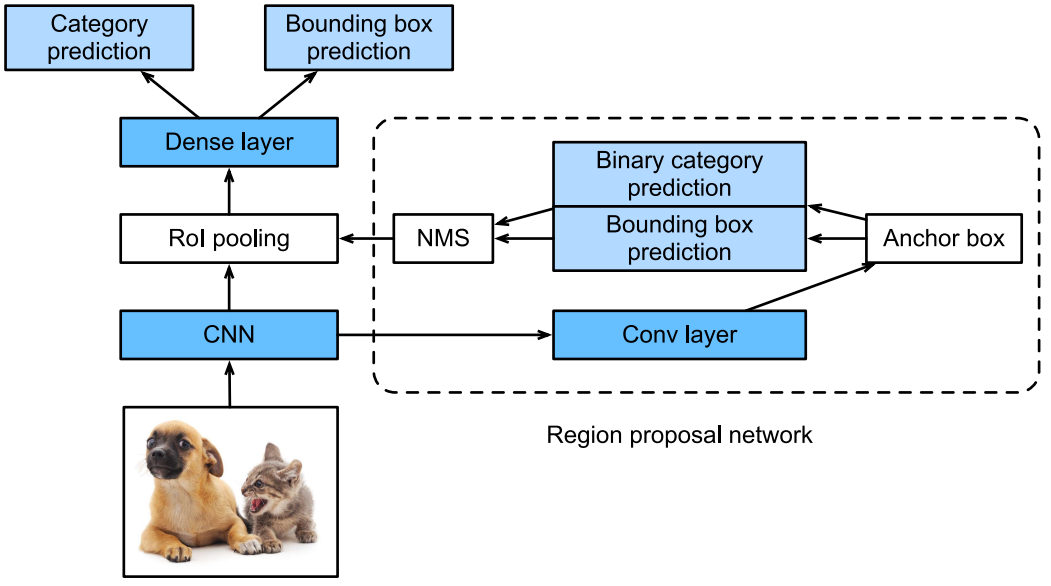


Figure 1. Fast R-CNN architecture. An input image and multiple regions of interest (RoIs) are input into a fully convolutional network. Each RoI is pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers (FCs). The network has two output vectors per RoI: softmax probabilities and per-class bounding-box regression offsets. The architecture is trained end-to-end with a multi-task loss.

—

Credits: Dive Into Deep Learning, 2020.



Faster R-CNN

- The performance of both R-CNN and Fast R-CNN is tied to the quality of the region proposals from selective search.
- Faster R-CNN replaces selective search with a region proposal network.
- This network reduces the number of proposed regions generated, while ensuring precise object detection.

—

Credits: Dive Into Deep Learning, 2020.

YOLO (v2) vs YOLO 9000 vs SSD vs Faster RCNN

Takeaways

- One-stage detectors (YOLO, SSD, RetinaNet, etc) are fast for inference but are usually not the most accurate object detectors.
- Two-stage detectors (Fast R-CNN, Faster R-CNN, R-FCN, Light head R-CNN, etc) are usually slower but are often more accurate.
- All networks depend on lots of engineering decisions.

(demo)

Segmentation

Segmentation is the task of partitioning an image, at the pixel level, into regions:

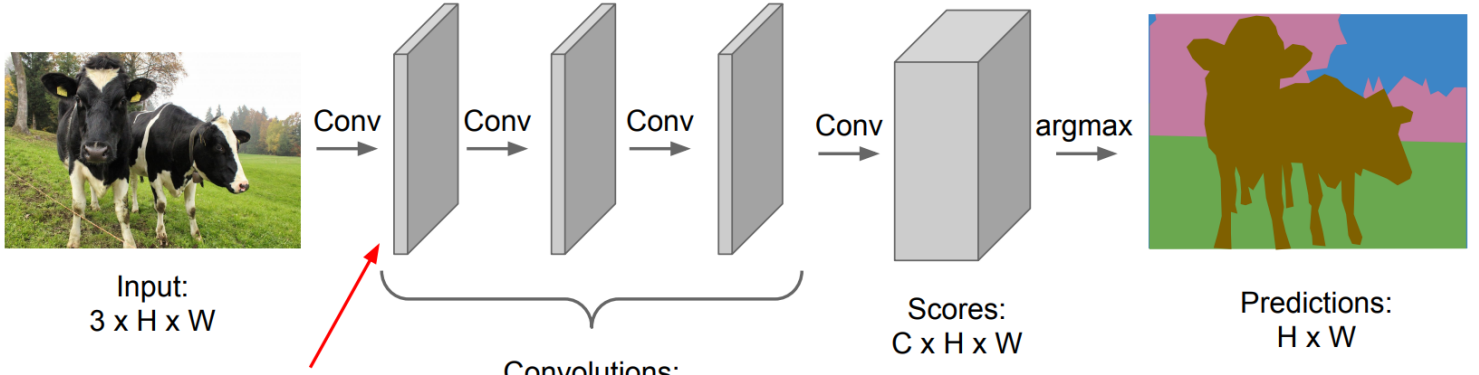
- **Semantic segmentation**: All pixels in an image are labeled with their class (e.g., car, pedestrian, road).
- **Instance segmentation**: Pixels of detected objects are labeled with an instance ID (e.g., car 1, car 2, pedestrian 1).
- Panoptic segmentation: Combines semantic and instance segmentation. All pixels in an image are labeled with a class and an instance ID (if applicable).

—

Credits: Dive Into Deep Learning, 2020.

The deep learning approach casts semantic segmentation as pixel classification. Convolutional networks can be used for that purpose, but with a few adaptations.

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!



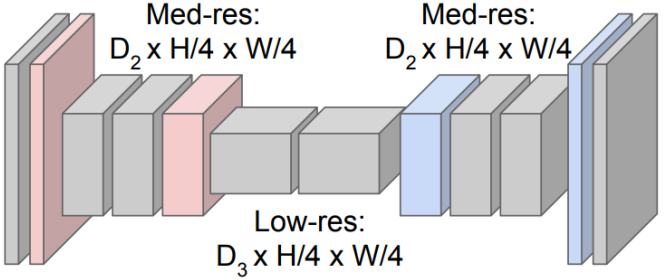
—

Downsampling:
Pooling, strided convolution

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

Upsampling:
???

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

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Transposed convolution

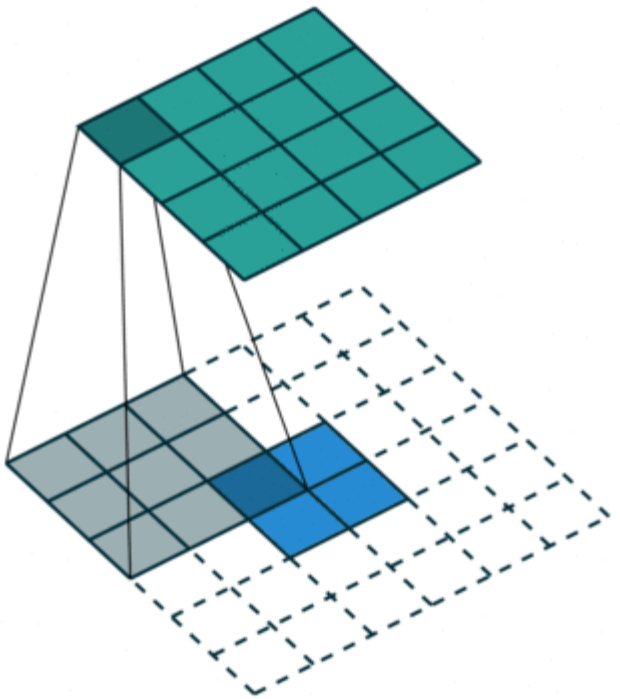
A transposed convolution is a convolution where the implementation of the forward and backward passes are swapped.

Given a convolutional kernel $\mathbf{u}$,

- the forward pass is implemented as $v(\mathbf{h}) = \mathbf{U}^T v(\mathbf{x})$ with appropriate reshaping, thereby effectively up-sampling an input $v(\mathbf{x})$ into a larger one;
- the backward pass is computed by multiplying the loss by $\mathbf{U}$ instead of $\mathbf{U}^T$.

$$\mathbf{U}^T v(\mathbf{x}) = v(\mathbf{h})$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 4 & 1 & 0 & 0 \\ 1 & 4 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 4 & 1 & 4 & 1 \\ 3 & 4 & 1 & 4 \\ 0 & 3 & 0 & 1 \\ 3 & 0 & 1 & 0 \\ 3 & 3 & 4 & 1 \\ 1 & 3 & 3 & 4 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 3 & 3 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \\ 4 \\ 4 \end{pmatrix} = \begin{pmatrix} 2 \\ 9 \\ 6 \\ 1 \\ 6 \\ 29 \\ 30 \\ 7 \\ 10 \\ 29 \\ 33 \\ 13 \\ 12 \\ 24 \\ 16 \\ 4 \end{pmatrix}$$



—

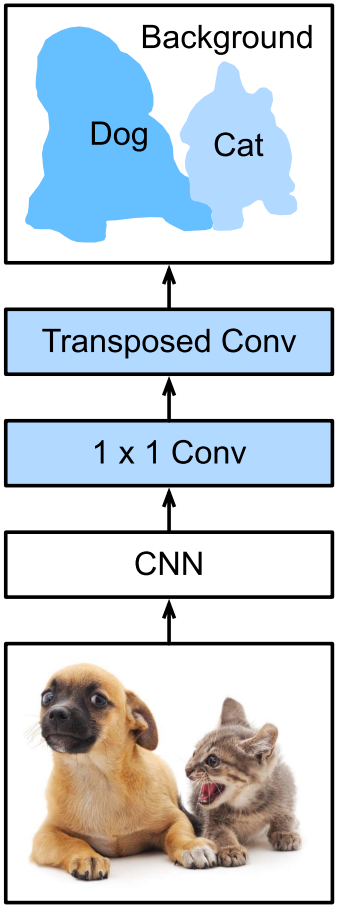
Credits: Dumoulin and Visin, A guide to convolution arithmetic for deep learning, 2016.

Fully convolutional networks (FCNs)

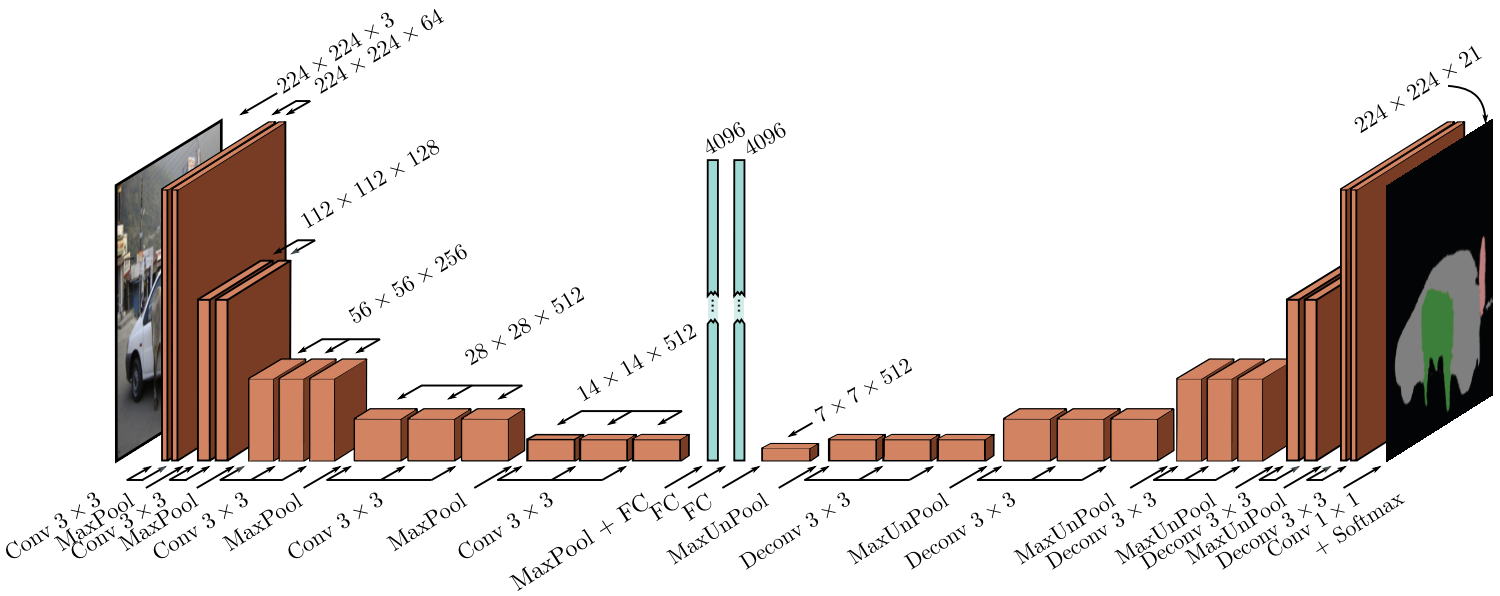
A fully convolutional network (FCN) is a convolutional network that replaces the fully connected layers with convolutional layers and transposed convolutional layers.

For semantic segmentation, the simplest design of a fully convolutional network consists in:

- using a (pre-trained) convolutional network for downsampling and extracting image features;
- replacing the dense layers with a 1×1 convolution layer to transform the number of channels into the number of categories;
- upsampling the feature map to the size of the input image by using one (or several) transposed convolution layer(s).



Contrary to fully connected networks, the dimensions of the output of a fully convolutional network is not fixed. It directly depends on the dimensions of the input, which can be images of arbitrary sizes.



The previous **encoder-decoder architecture** is a simple and effective way to perform semantic segmentation.

However, the low-resolution representation in the middle of the network can be a bottleneck for the segmentation performance, as it must retain enough information to reconstruct the high-resolution segmentation map.

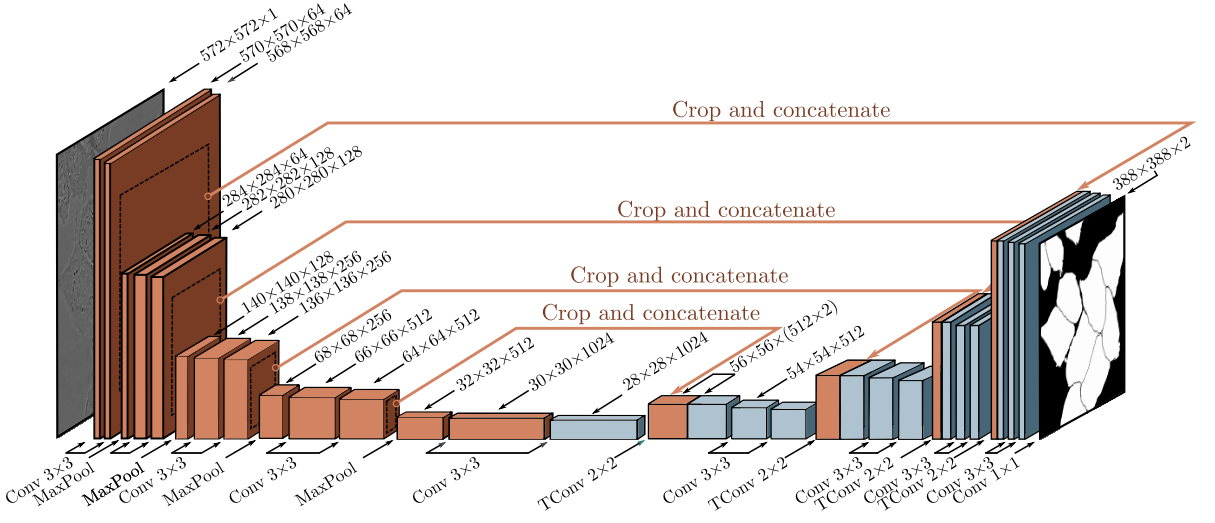
—

Credits: Simon J.D. Prince, Understanding Deep Learning, 2023.

UNet

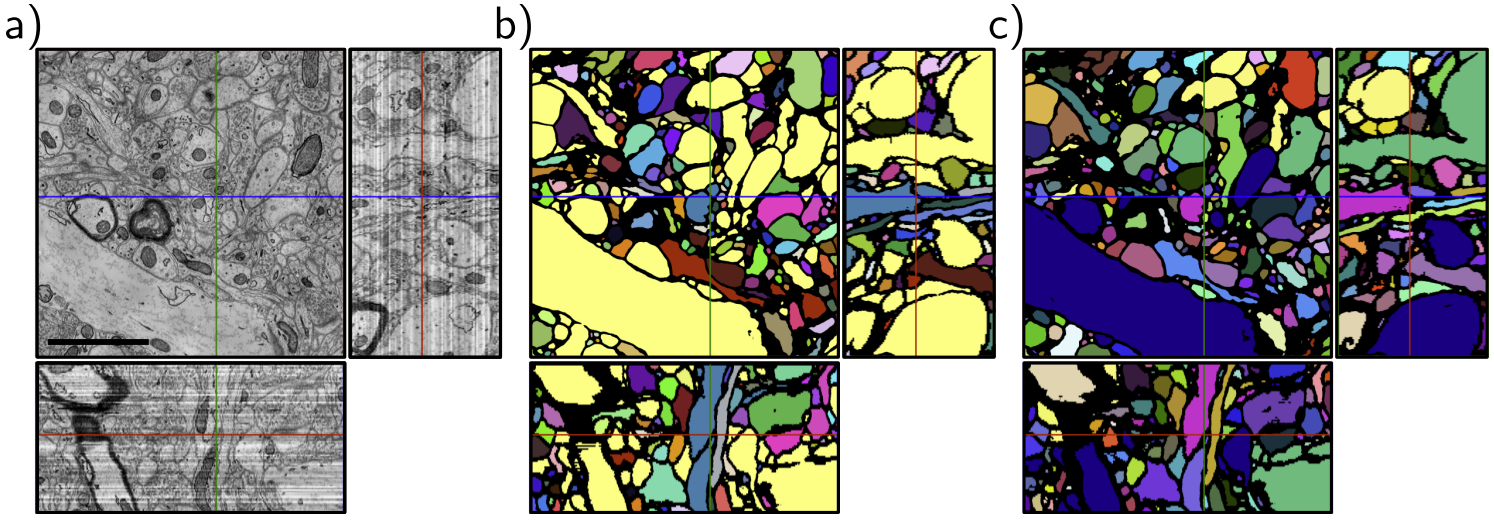
The **UNet** architecture is an encoder-decoder architecture with skip connections (usually concatenations) that directly connect the encoder and decoder layers at the same resolution. In this way, the decoder can use both

- the corresponding high-resolution features from the encoder, and
- the lower-resolution features from the previous layers.



—

Credits: Simon J.D. Prince, Understanding Deep Learning, 2023.



3d segmentation results using a UNet architecture.
(a) Slices of a 3d volume of a mouse cortex, (b) A UNet is used to classify voxels as either inside or outside neutrites. Connected regions are shown with different colors, (c) 5-member ensemble of UNets.

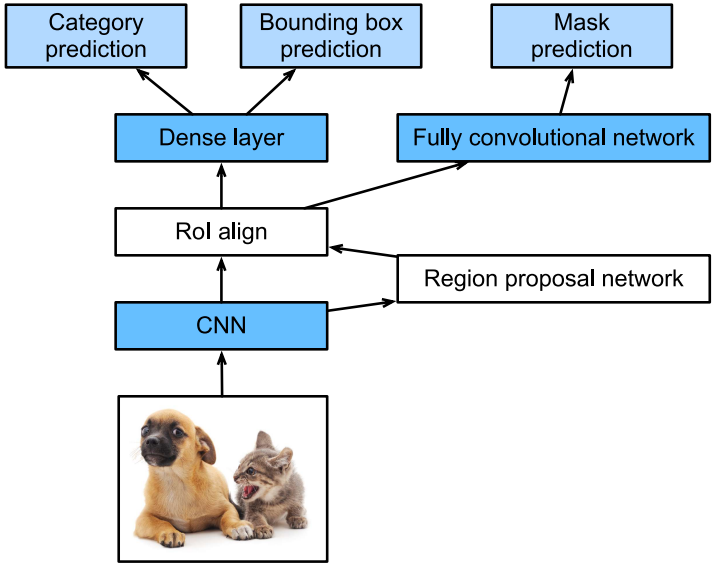
—

(demo)

Mask R-CNN

Segmentation is a natural extension of object detection. For example, Mask R-CNN extends the Faster R-CNN model for semantic segmentation:

- The RoI pooling layer is replaced with an RoI alignment layer.
- It branches off to an FCN for predicting a semantic segmentation mask.
- Object detection combined with mask prediction enables instance segmentation.

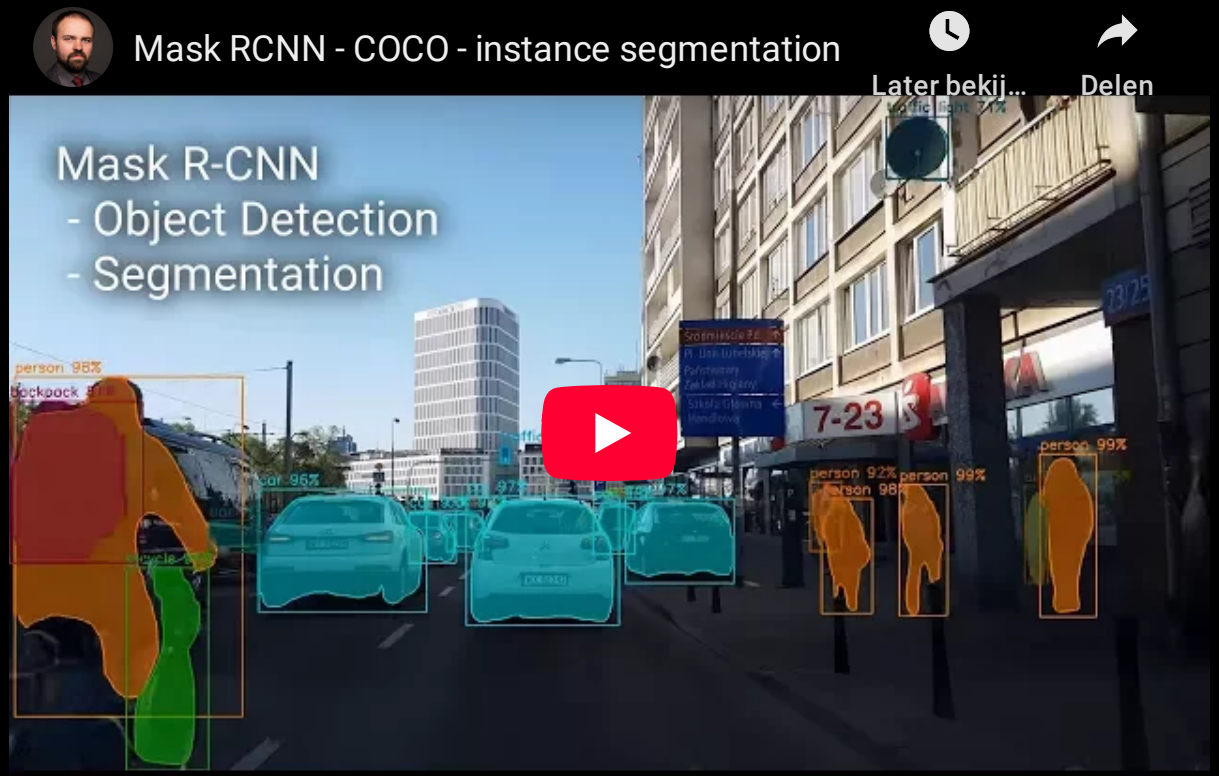


—

Credits: Dive Into Deep Learning, 2020.

—

Credits: He et al, 2017.


Mask RCNN - COCO - instance segmentation
Later bekij...
Delen
Mask R-CNN
- Object Detection
- Segmentation
person 98%
person 92%
person 99%
person 99%
7-23

It is noteworthy that for detection and segmentation, there is an heavy re-use of large networks trained for classification.

The models themselves, as much as the source code of the algorithm that produced them, or the training data, are generic and re-usable assets.

The end.