

Deep Learning

Lecture 6: Computer vision

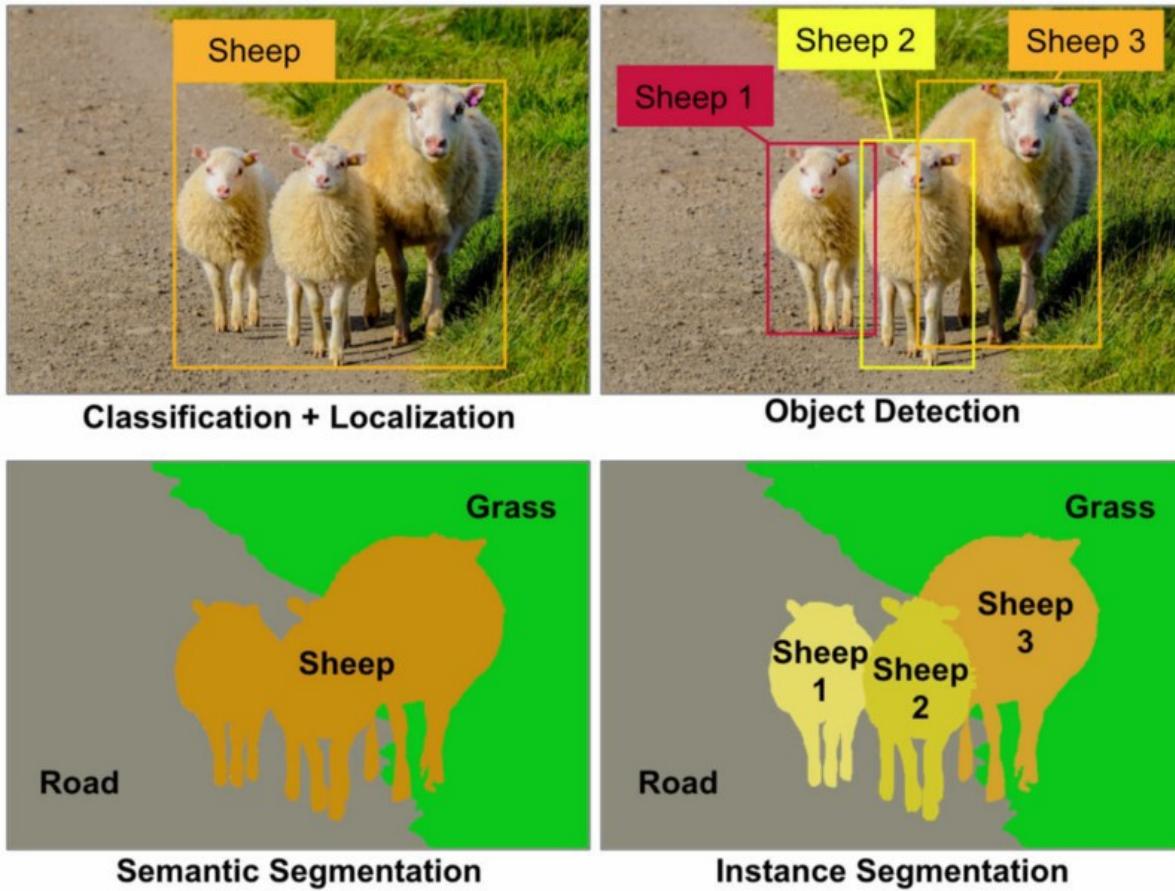
Prof. Gilles Louppe
g.louppe@uliege.be



Today

Advanced deep neural networks for computer vision:

- Classification
- Object detection
- Semantic segmentation



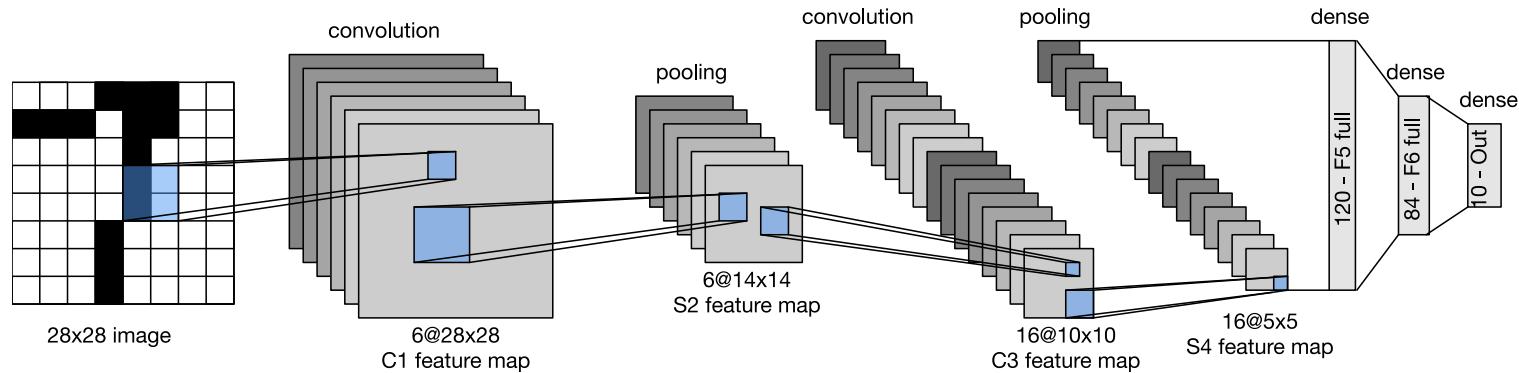
*Some of the main computer vision tasks.
Each of them requires a different neural network architecture.*

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.



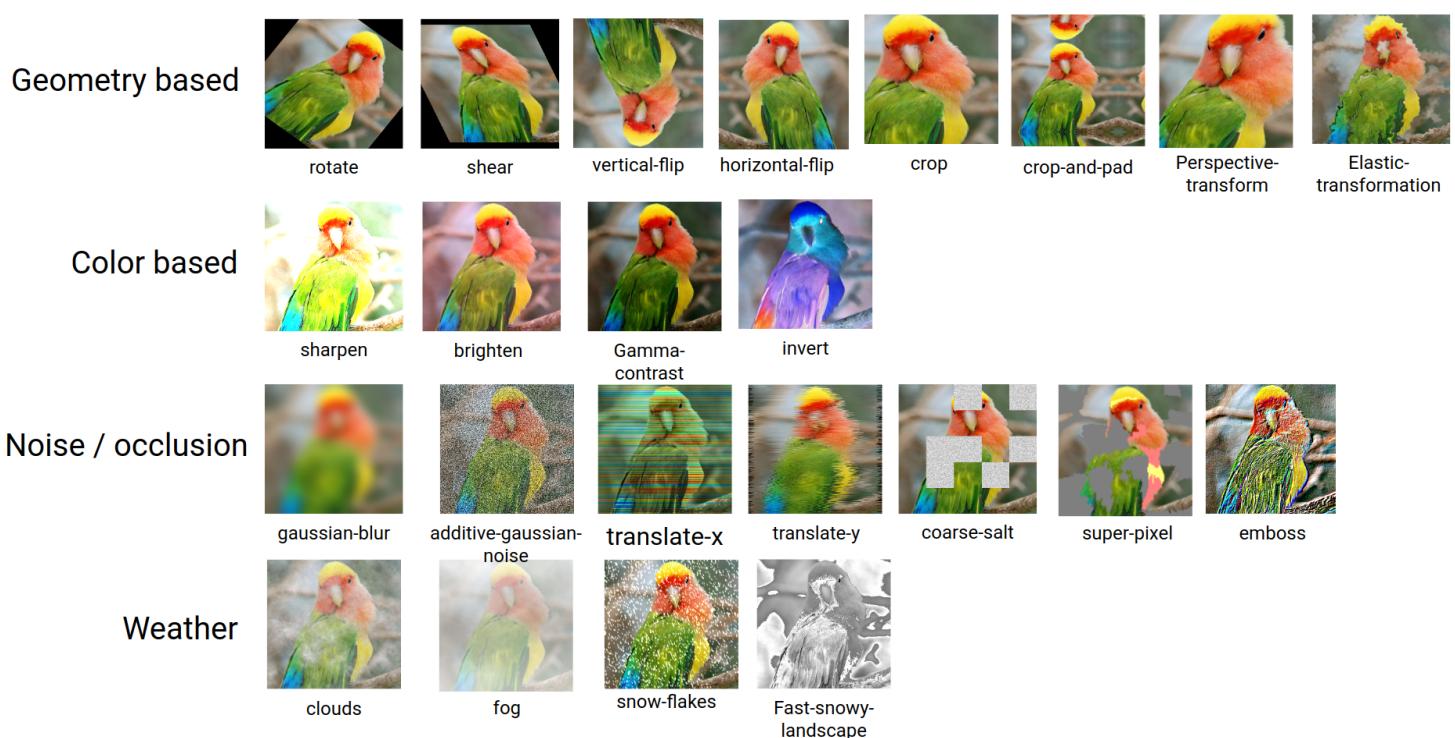
For classification,

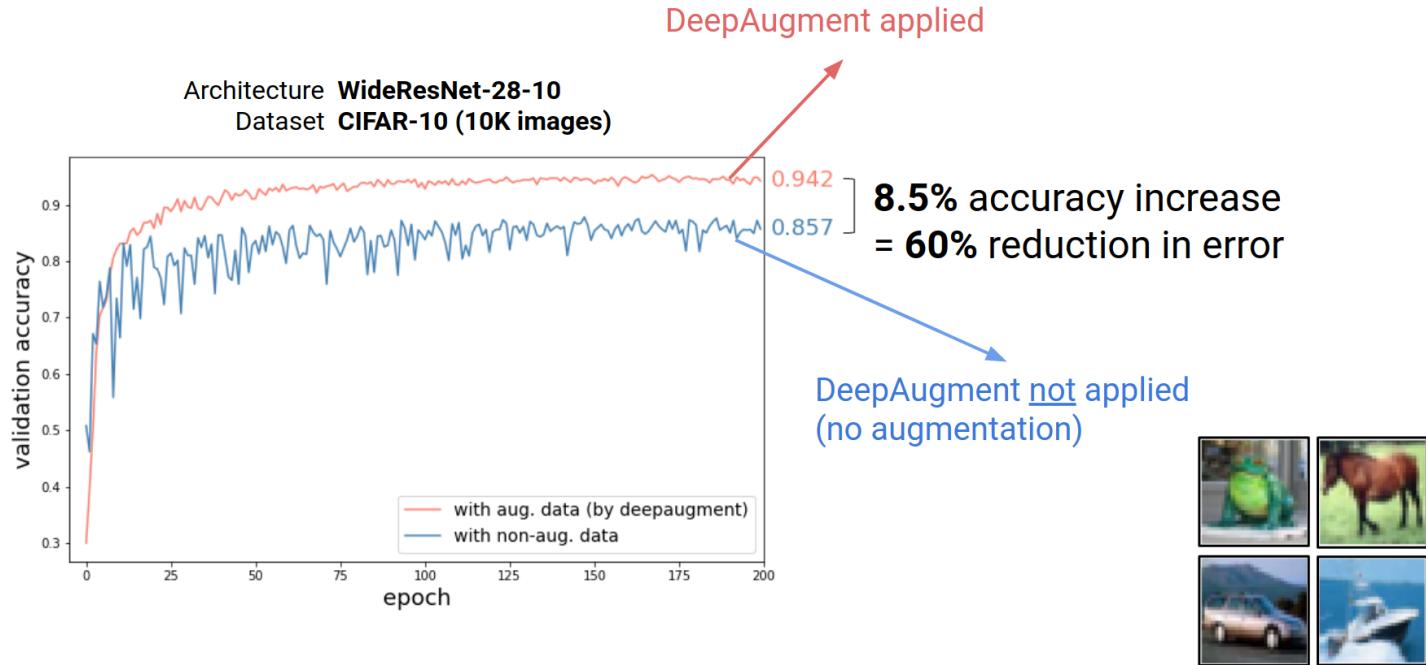
- the activation in the output layer is a Softmax activation producing a vector $\mathbf{h} \in \Delta^C$ of probability estimates $P(Y = i|\mathbf{x})$, where C is the number of classes;
- the loss function is the cross-entropy loss.

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 (e.g., as demonstrated with AlexNet).





Pre-trained models

- Training a model on natural images, from scratch, takes **days or weeks**.
- Many models trained on ImageNet 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.

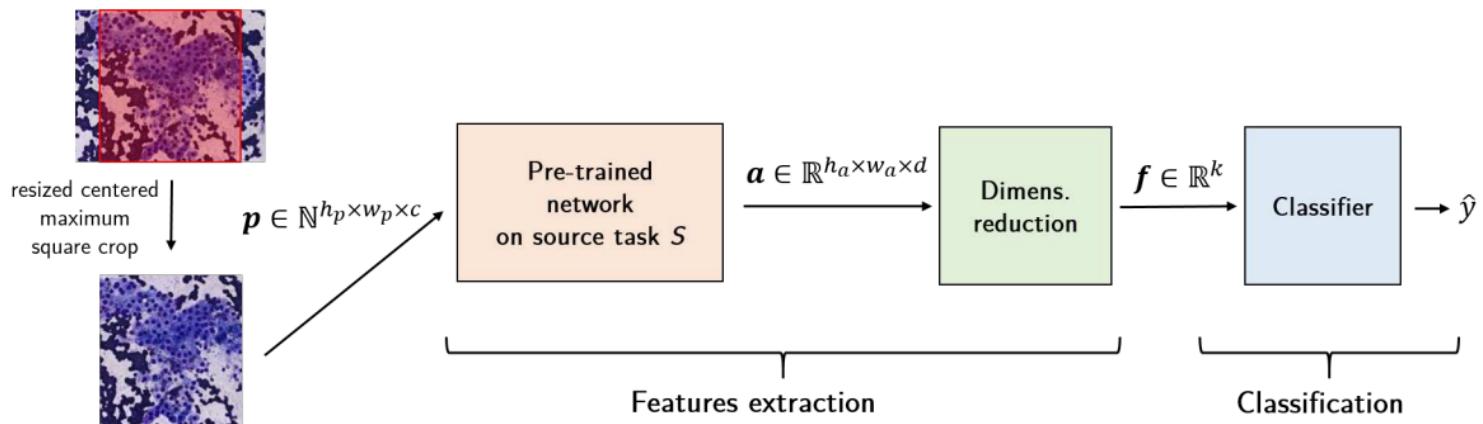
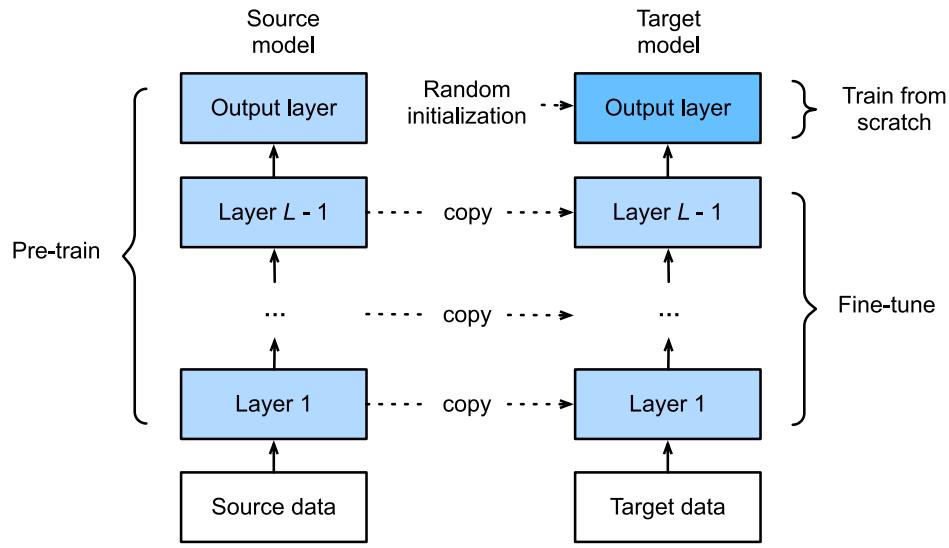


Figure 2. Feature extraction from pre-trained convolutional neural networks



Fine-tuning

- Same as for transfer learning, but also **fine-tune** the weights of the pre-trained network by continuing backpropagation.
- All or only some of the layers can be tuned.

In the case of models pre-trained on ImageNet, transferred/fine-tuned networks usually work even when the input images for the new task are not photographs of objects or animals, such as biomedical images, satellite images or paintings.

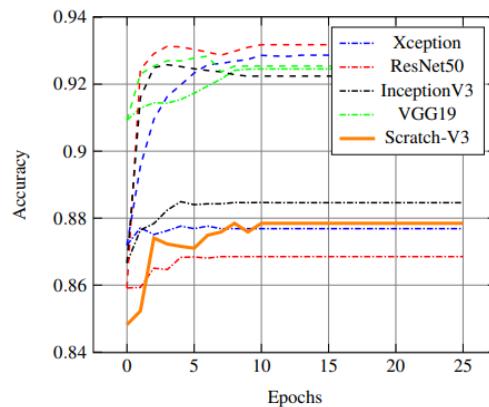


Fig. 2: Comparison between the fine tuning approach versus the off the shelf one when classifying the material of the heritage objects of the Rijksmuseum dataset. We observe how the first approach (as reported by the the dashed lines) leads to significant improvements when compared to the latter one (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).

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

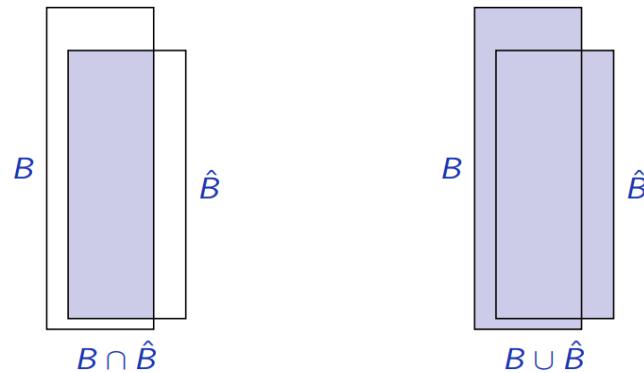


Final list of detections

Intersection over Union (IoU)

A standard performance indicator for object detection is to evaluate the **intersection over union** (IoU) between a predicted bounding box \hat{B} and an annotated bounding box B ,

$$\text{IoU}(B, \hat{B}) = \frac{\text{area}(B \cap \hat{B})}{\text{area}(B \cup \hat{B})}.$$

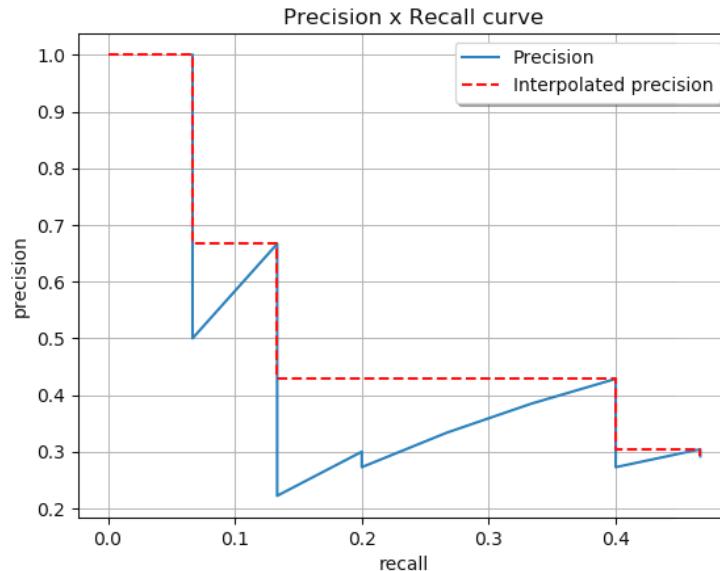


Mean Average Precision (mAP)

If $\text{IoU}(B, \hat{B})$ is larger than a fixed threshold (usually $\frac{1}{2}$), then the predicted bounding-box is valid (true positive) and wrong otherwise (false positive).

TP and FP values are accumulated for all thresholds on the predicted confidence. The area under the resulting precision-recall curve is the **average precision** for the considered class.

The mean over the classes is the **mean average precision** (mAP).



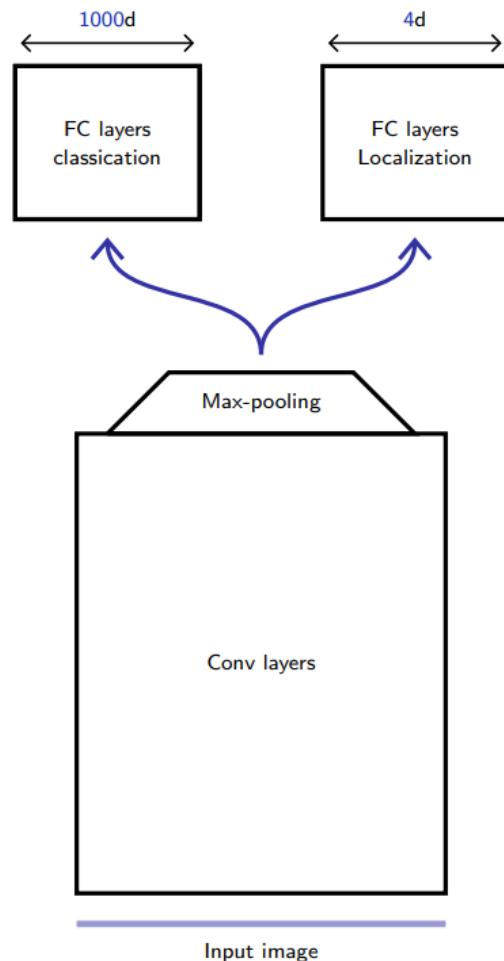
The sliding window approach evaluates a classifier at a large number of locations and scales.

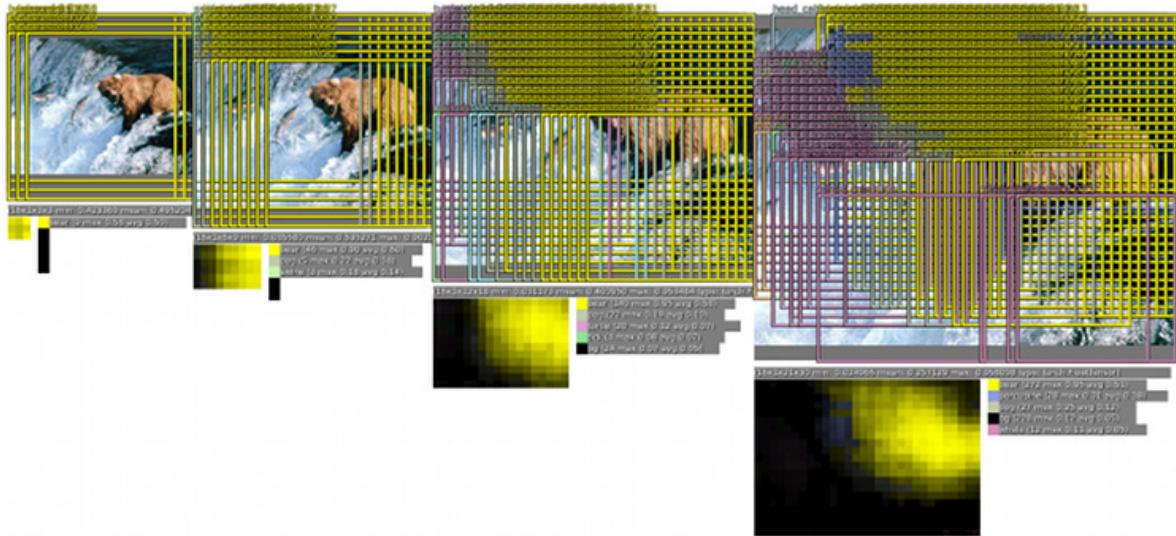
This approach is usually **very computationally expensive** as performance directly depends on the resolution and number of the windows fed to the classifier (the more the better, but also the more costly).

OverFeat

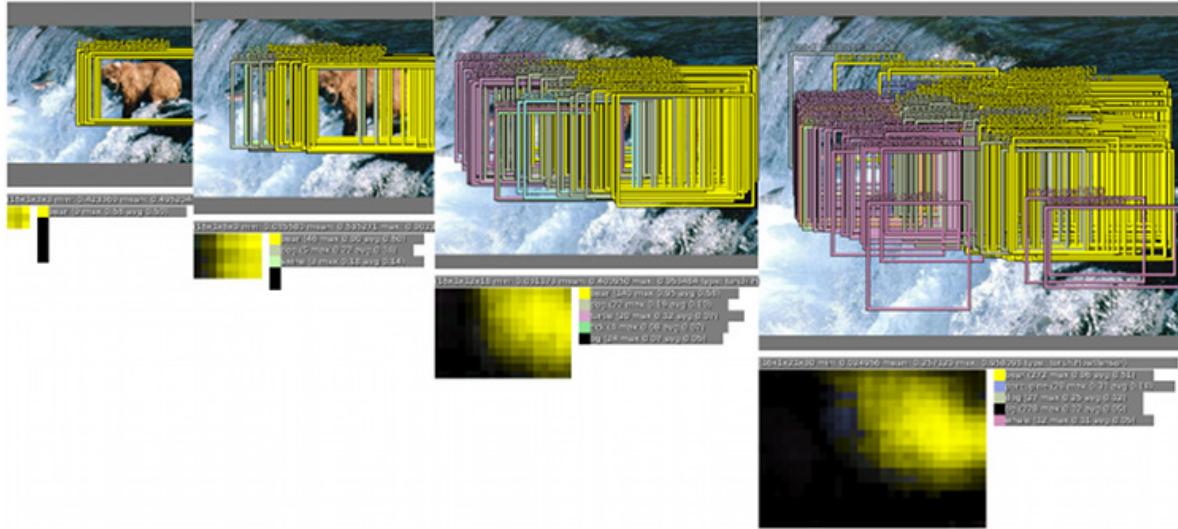
The complexity of the sliding window approach was mitigated in the pioneer OverFeat network (Sermanet et al, 2013) by adding a **regression head** to predict the object **bounding box** (x, y, w, h) .

For training, the convolutional layers are fixed and the regression network is trained using an ℓ_2 loss between the predicted and the true bounding box for each example.





The classifier head outputs a class and a confidence for each location and scale pre-defined from a coarse grid. Each window is resized to fit with the input dimensions of the classifier.



The regression head then predicts the location of the object with respect to each window.

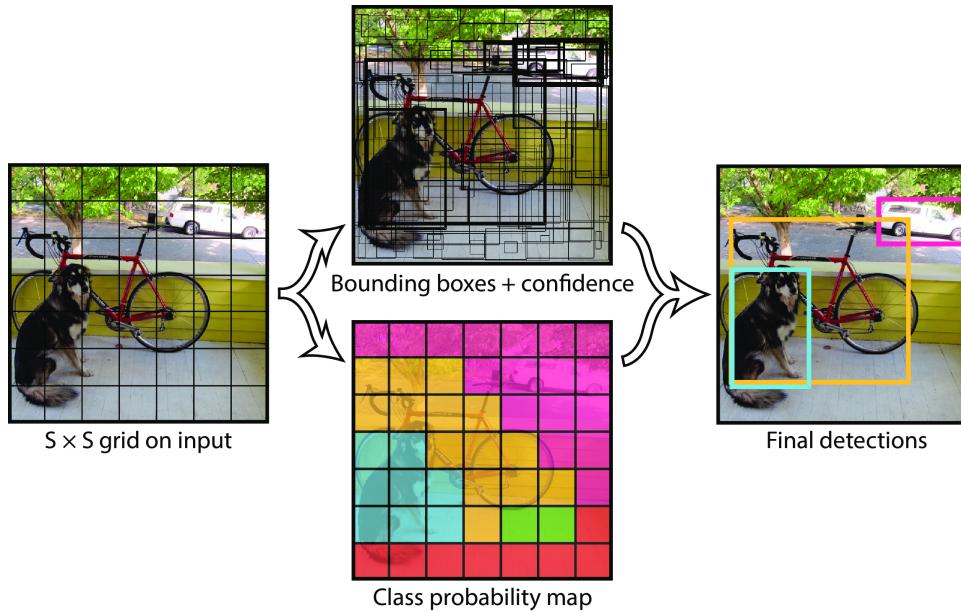


These bounding boxes are finally merged with an [ad-hoc greedy procedure](#) (e.g., Non-maximum suppression) to produce the final predictions over a small number of objects.

Although OverFeat is one of the earliest successful networks for object detection, its 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.

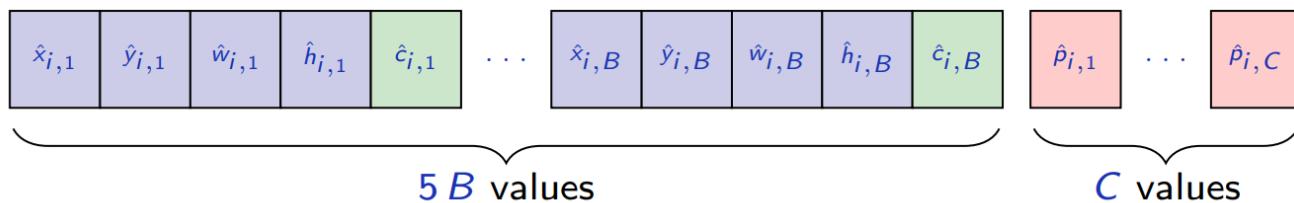
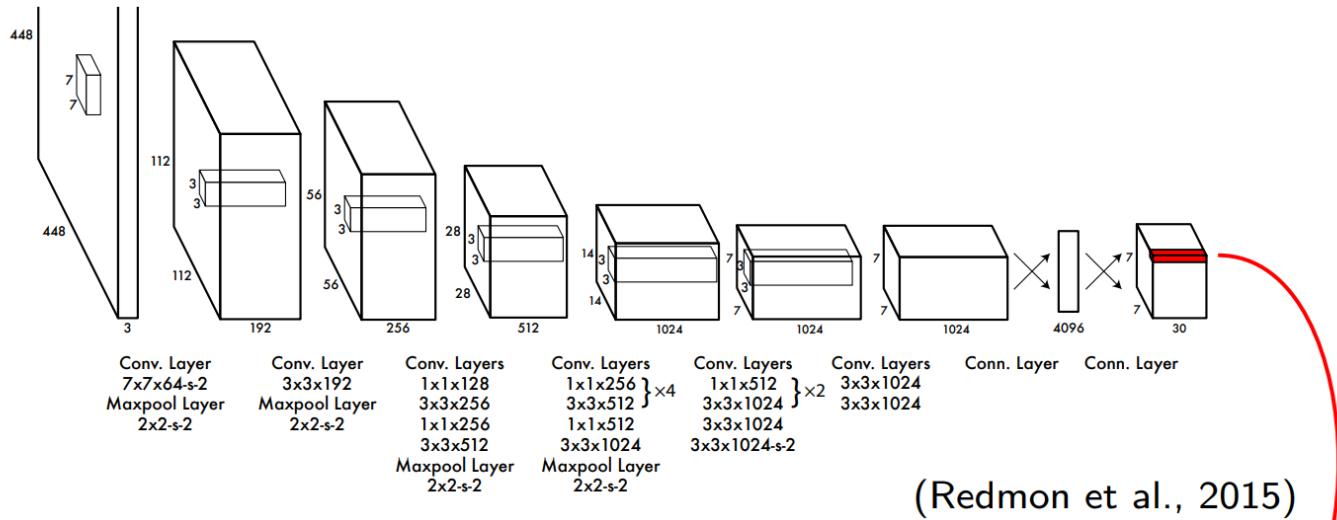
YOLO



YOLO (You Only Look Once; 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.

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 $i = 1, \dots, S \times S$, predicted box $j = 1, \dots, B$ and class index $c = 1, \dots, C$,

- 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 annotated bounding 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

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

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.

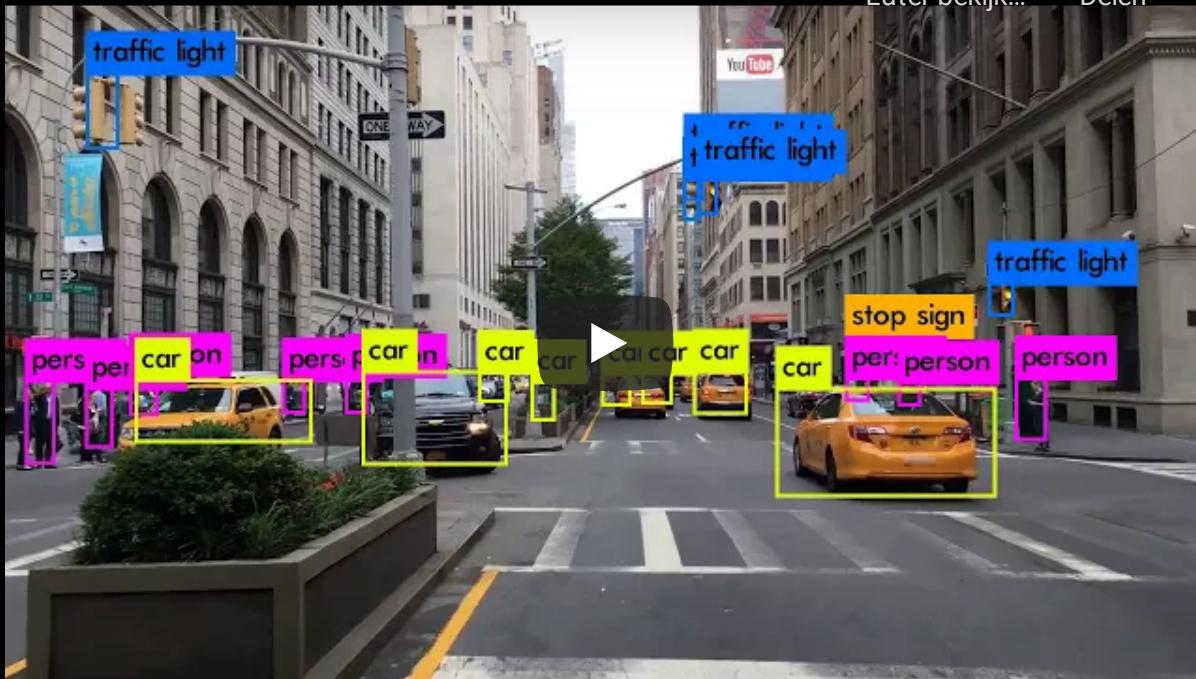


YOLO in New York



Later bekijk...

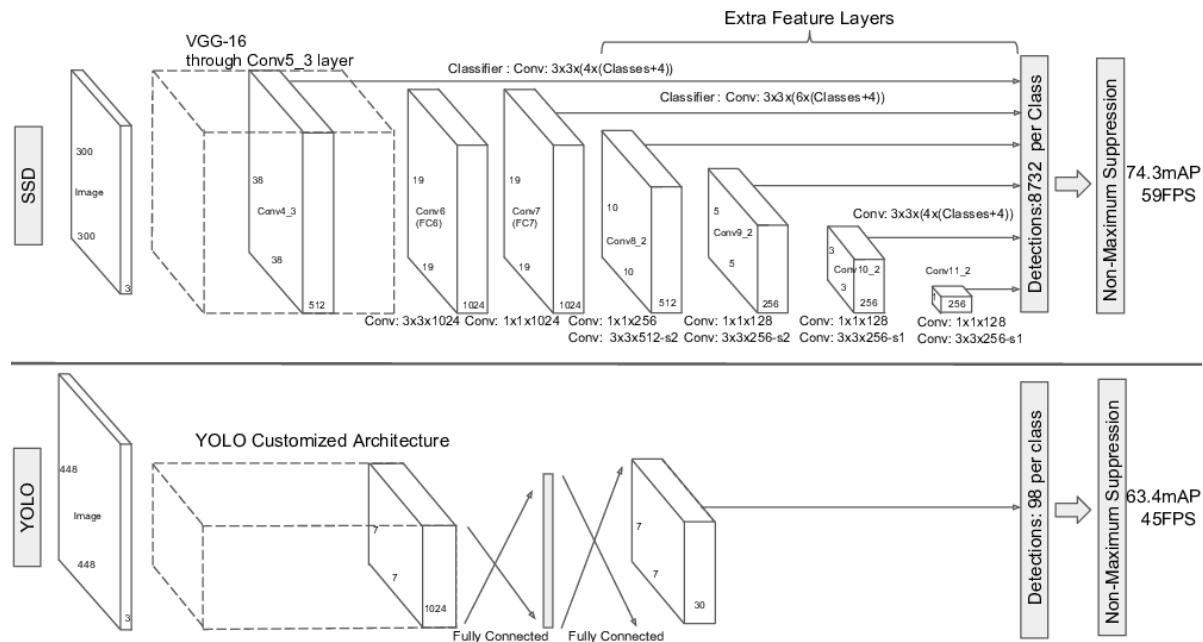
Delen



Redmon, 2017.

SSD

The Single Shot Multi-box Detector (SSD; Liu et al, 2015) improves upon YOLO by using a fully-convolutional architecture and multi-scale maps.



Region-based CNNs

An alternative strategy to having a huge predefined set of box proposals, as in OverFeat or YOLO, is to rely on [region proposals](#) first extracted from the image.

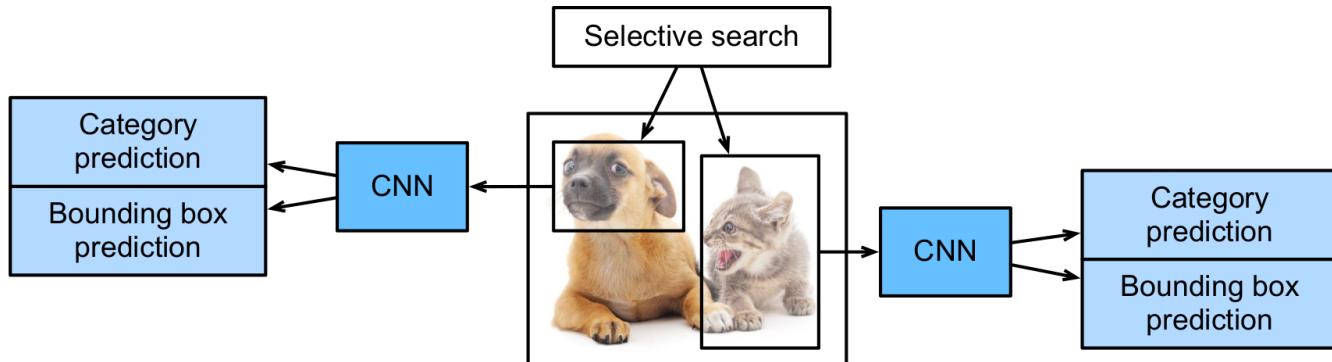
The main family of architectures following this principle are [region-based](#) convolutional neural networks:

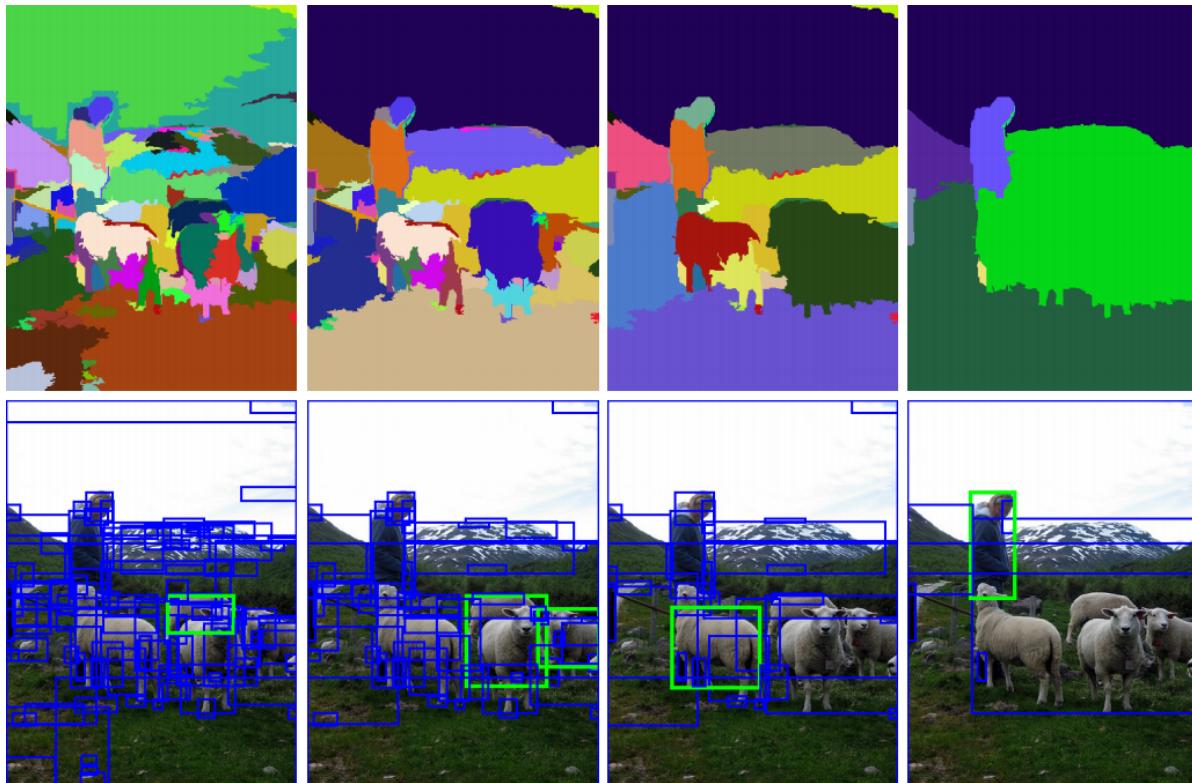
- (Slow) R-CNN (Girshick et al, 2014)
- Fast R-CNN (Girshick et al, 2015)
- Faster R-CNN (Ren et al, 2015)
- Mask R-CNN (He et al, 2017)

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.





Selective search (Uijlings et al, 2013) looks at the image through windows of different sizes, and for each size tries to group together adjacent pixels that are similar by texture, color or intensity.

Fast R-CNN

- The main performance bottleneck of an R-CNN model 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.

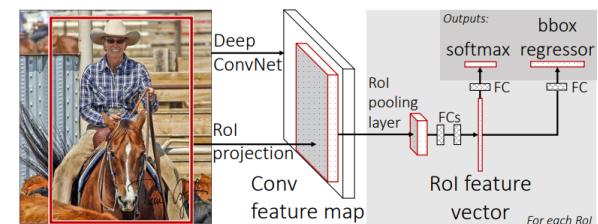
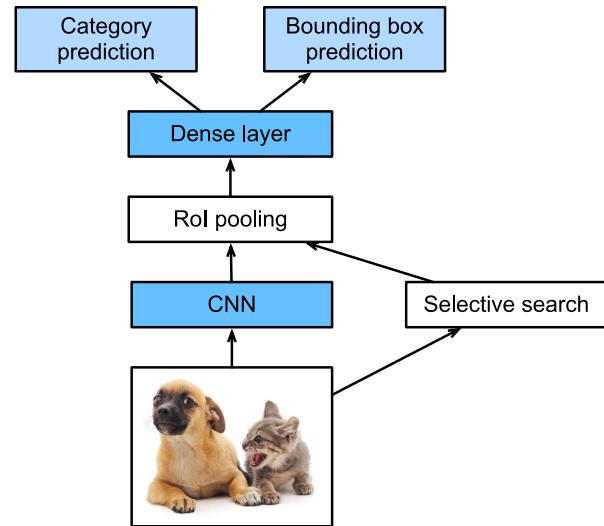
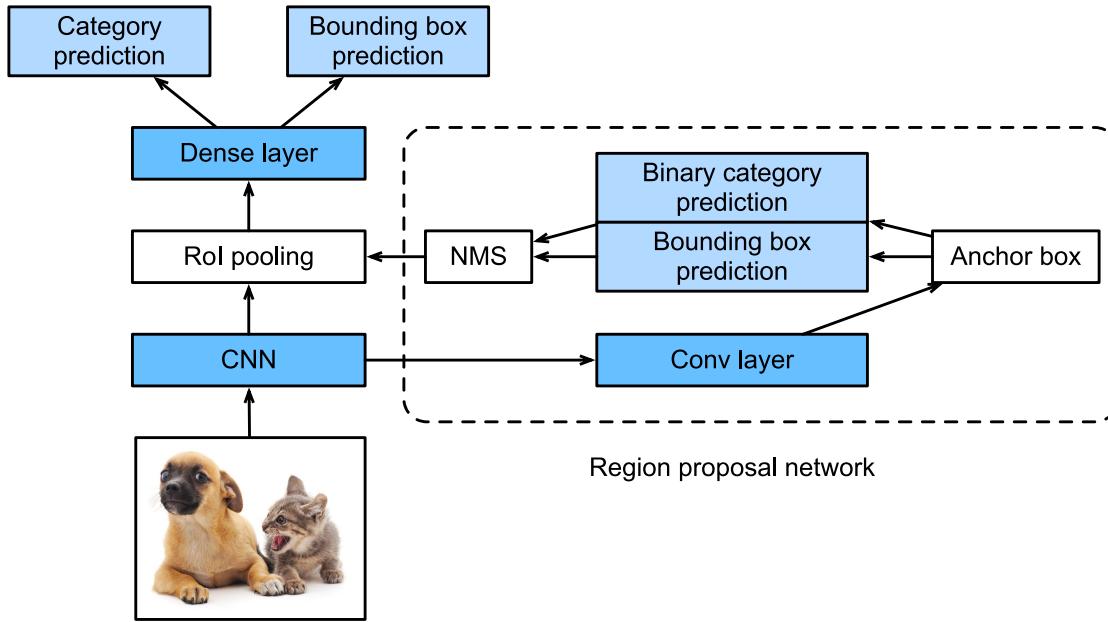


Figure 1. Fast R-CNN architecture. An input image and multiple regions of interest (ROIs) are input into a fully convolutional network. Each ROI is pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers (FCs). The network has two output vectors per ROI: softmax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



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.



YoloV2, Yolo 9000, SSD Mobilenet, Faster R...



Later bekijk...

Delen



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

Takeaways

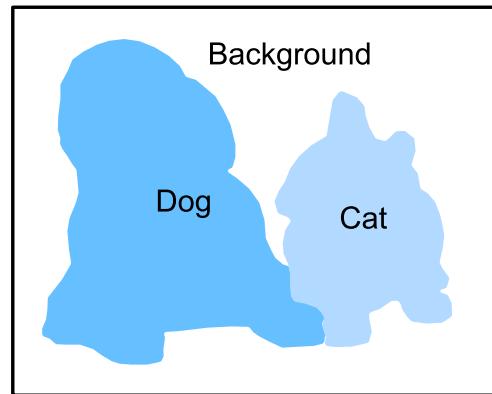
- One-stage detectors (YOLO, SSD, RetinaNet, etc) are fast for inference but are usually not the most accurate object detectors.
- Two-stage detectors (Fast R-CNN, Faster R-CNN, R-FCN, Light head R-CNN, etc) are usually slower but are often more accurate.
- All networks depend on lots of engineering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

(demo)

Segmentation

Semantic **segmentation** is the task of partitioning an image into regions of different semantic categories.

These semantic regions label and predict objects at the pixel level.

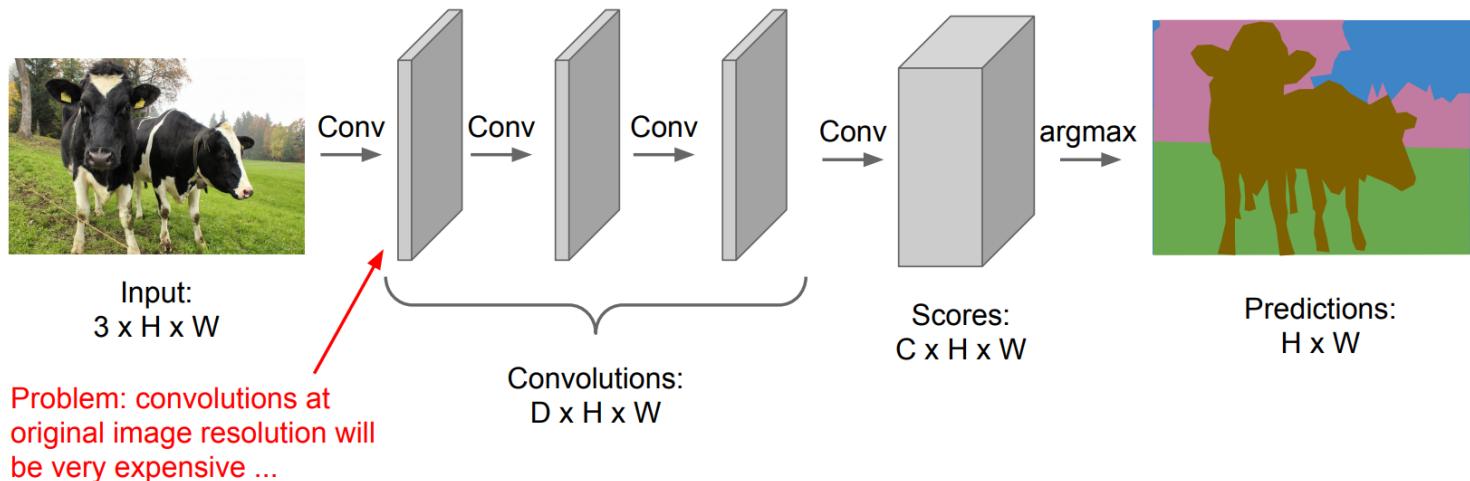


Fully convolutional networks

The historical approach to image segmentation was to define a measure of similarity between pixels, and to cluster groups of similar pixels. Such approaches account poorly for semantic content.

The deep-learning approach re-casts semantic segmentation as pixel classification, and re-uses networks trained for image classification by making them **fully convolutional** (FCNs).

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

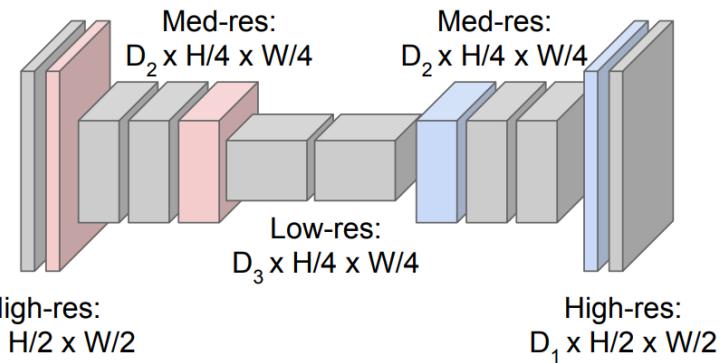


Downsampling:
Pooling, strided convolution



Input:
 $3 \times H \times W$

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



Upsampling:
???



Predictions:
 $H \times W$

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

The convolution and pooling layers introduced so far often reduce the input width and height, or keep them unchanged.

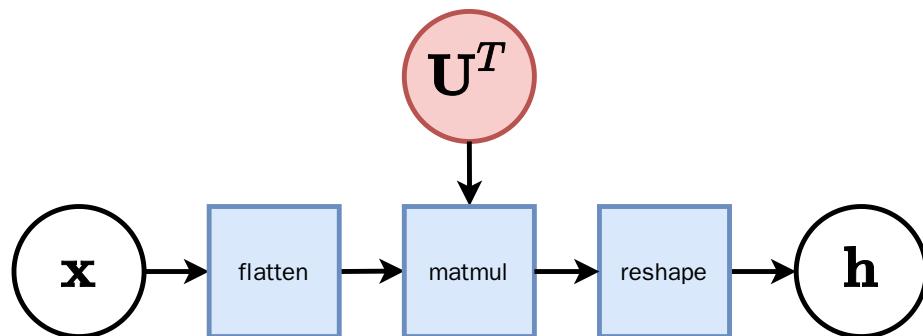
- Semantic segmentation requires to predict values for each pixel, and therefore needs to increase input width and height.
- Fully connected layers could be used for that purpose but would face the same limitations as before (spatial specialization, too many parameters).
- Ideally, we would like layers that implement the inverse of convolutional and pooling layers.

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

Given a convolutional kernel \mathbf{u} ,

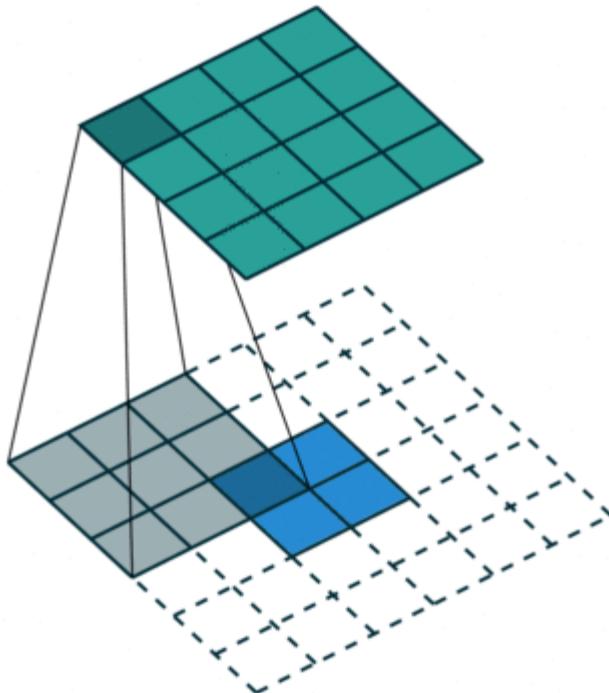
- the forward pass is implemented as $v(\mathbf{h}) = \mathbf{U}^T v(\mathbf{x})$ with appropriate reshaping, thereby effectively up-sampling an input $v(\mathbf{x})$ into a larger one;
- the backward pass is computed by multiplying the loss by \mathbf{U} instead of \mathbf{U}^T .

Transposed convolutions are also referred to as fractionally-strided convolutions or deconvolutions (mistakenly).



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

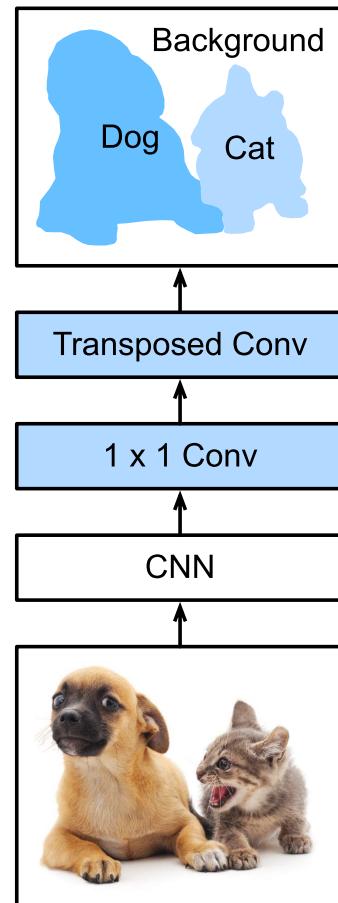
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 4 & 1 & 0 & 0 \\ 1 & 4 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 4 & 1 & 4 & 1 \\ 3 & 4 & 1 & 4 \\ 0 & 3 & 0 & 1 \\ 3 & 0 & 1 & 0 \\ 3 & 3 & 4 & 1 \\ 1 & 3 & 3 & 4 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 3 & 3 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix}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



FCNs for segmentation

The simplest design of a fully convolutional network for semantic segmentation 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.

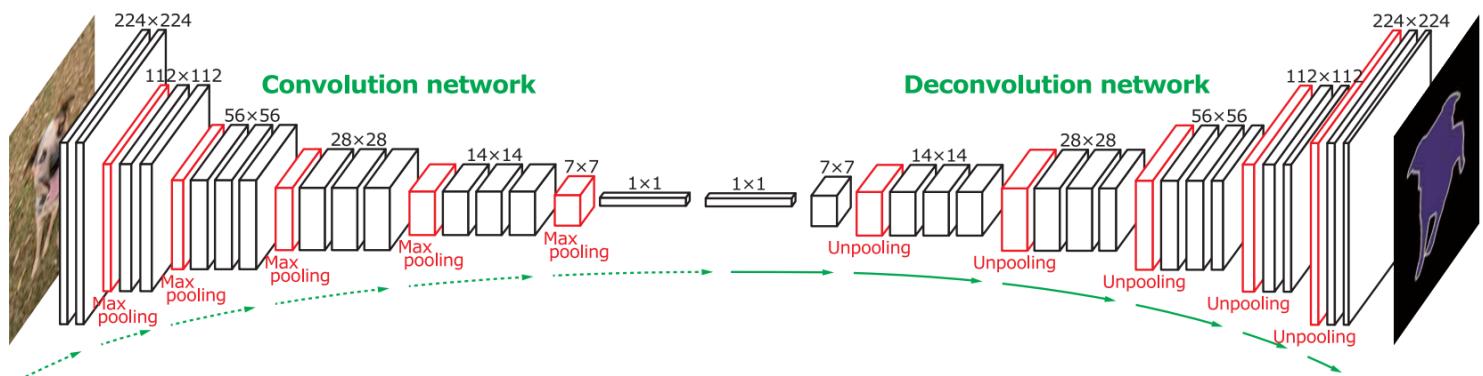
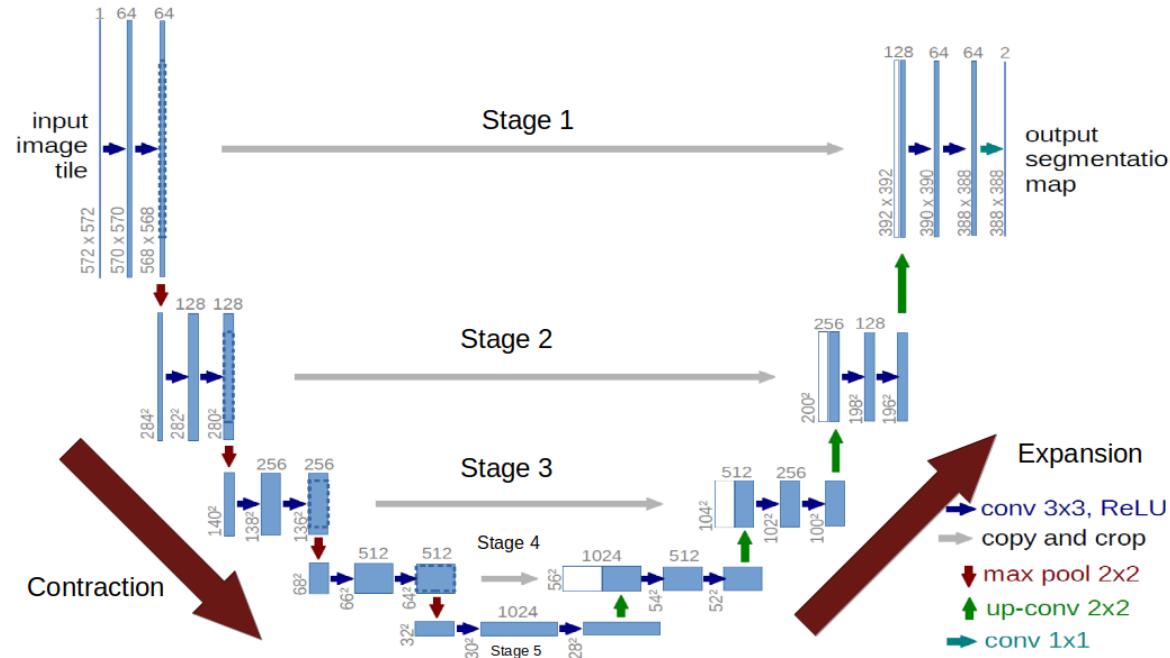


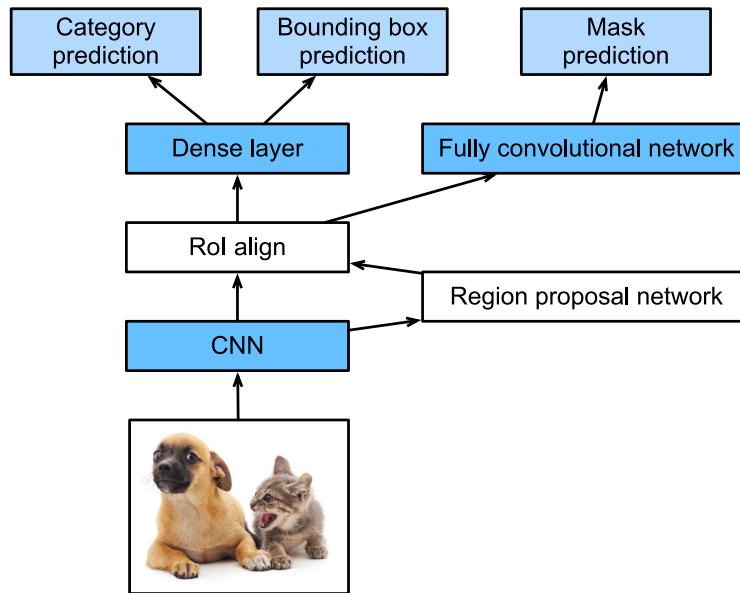
Figure 2. Overall architecture of the proposed network. On top of the convolution network based on VGG 16-layer net, we put a multi-layer deconvolution network to generate the accurate segmentation map of an input proposal. Given a feature representation obtained from the convolution network, dense pixel-wise class prediction map is constructed through multiple series of unpooling, deconvolution and rectification operations.

UNet

The UNet architecture builds upon the previous FCN architecture.

It consists in symmetric contraction and expansion paths, along with a concatenation of high resolution features from the contracting path to the unsampled features from the expanding path. These connections allow for localization.





Mask R-CNN

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 segmentation mask.
- Object detection combined with mask prediction enables **instance segmentation**.





Mask RCNN - COCO - instance segmentation



Later bekijk...

Delen



It is noteworthy that for detection and semantic 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.