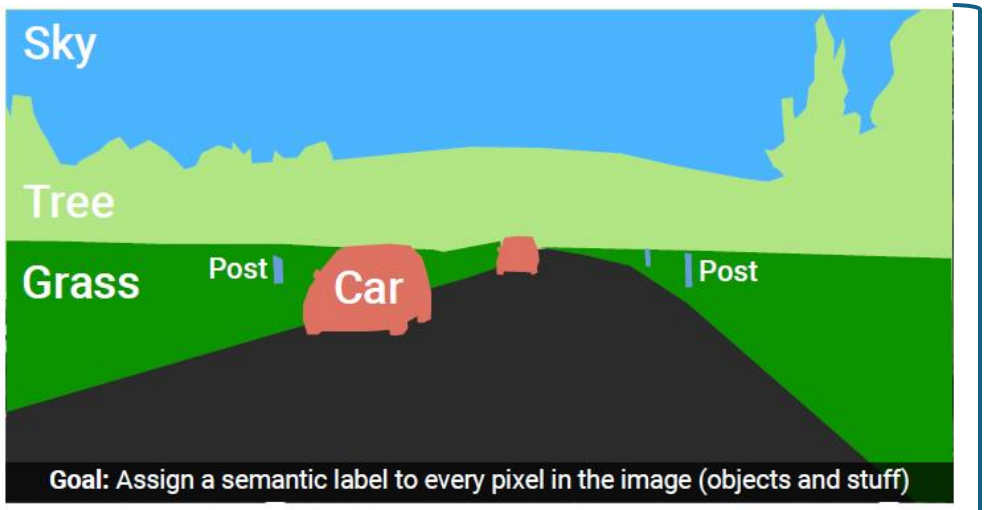
# Segmentation and Detection

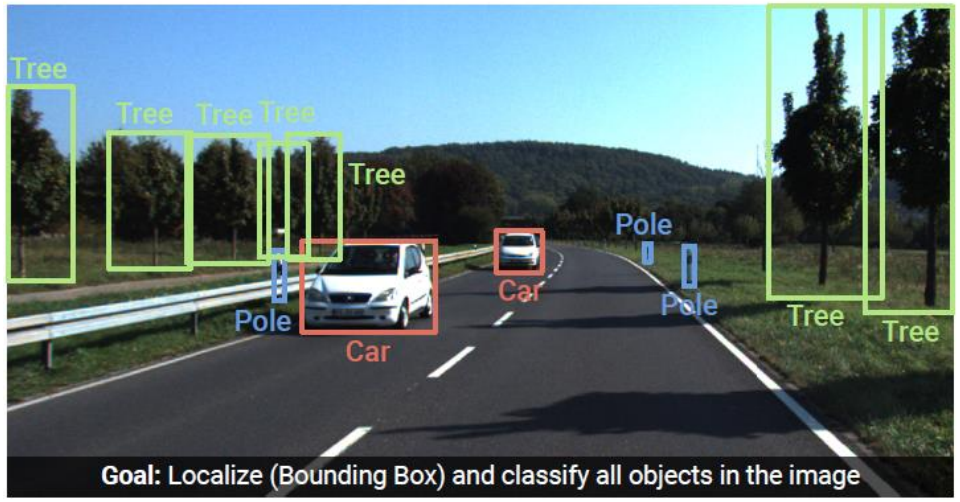
Computer Vision

# Image Understanding (Recognition)

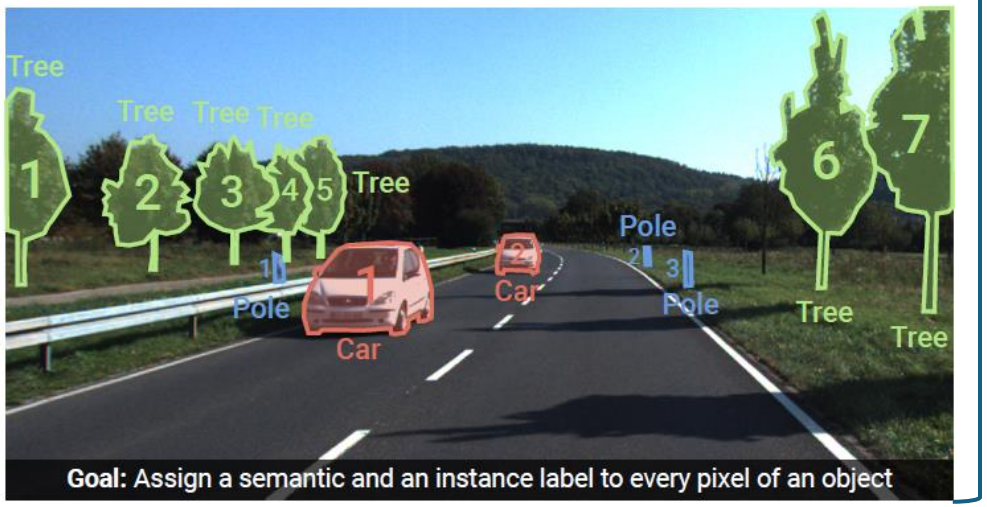

Image Classification




Semantic Segmentation




Object Detection



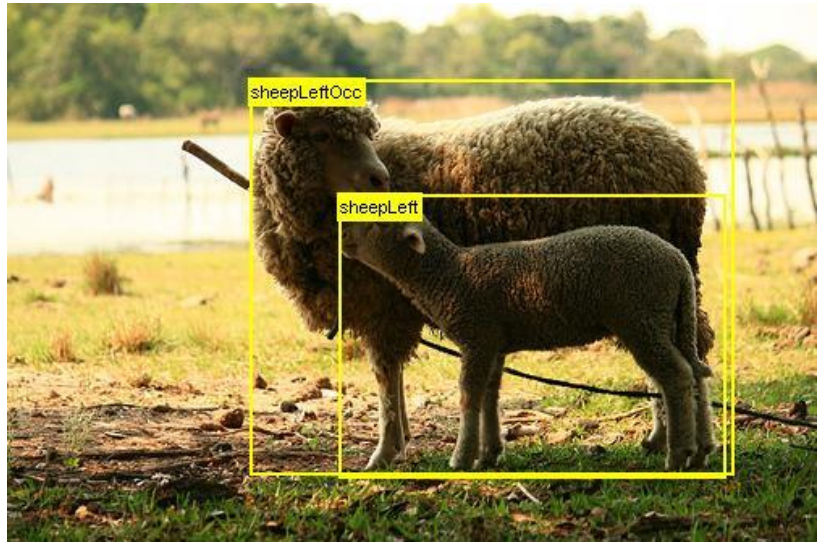

Instance Segmentation

combination of both: panoptic segmentation

# A Few More Image Data Sets

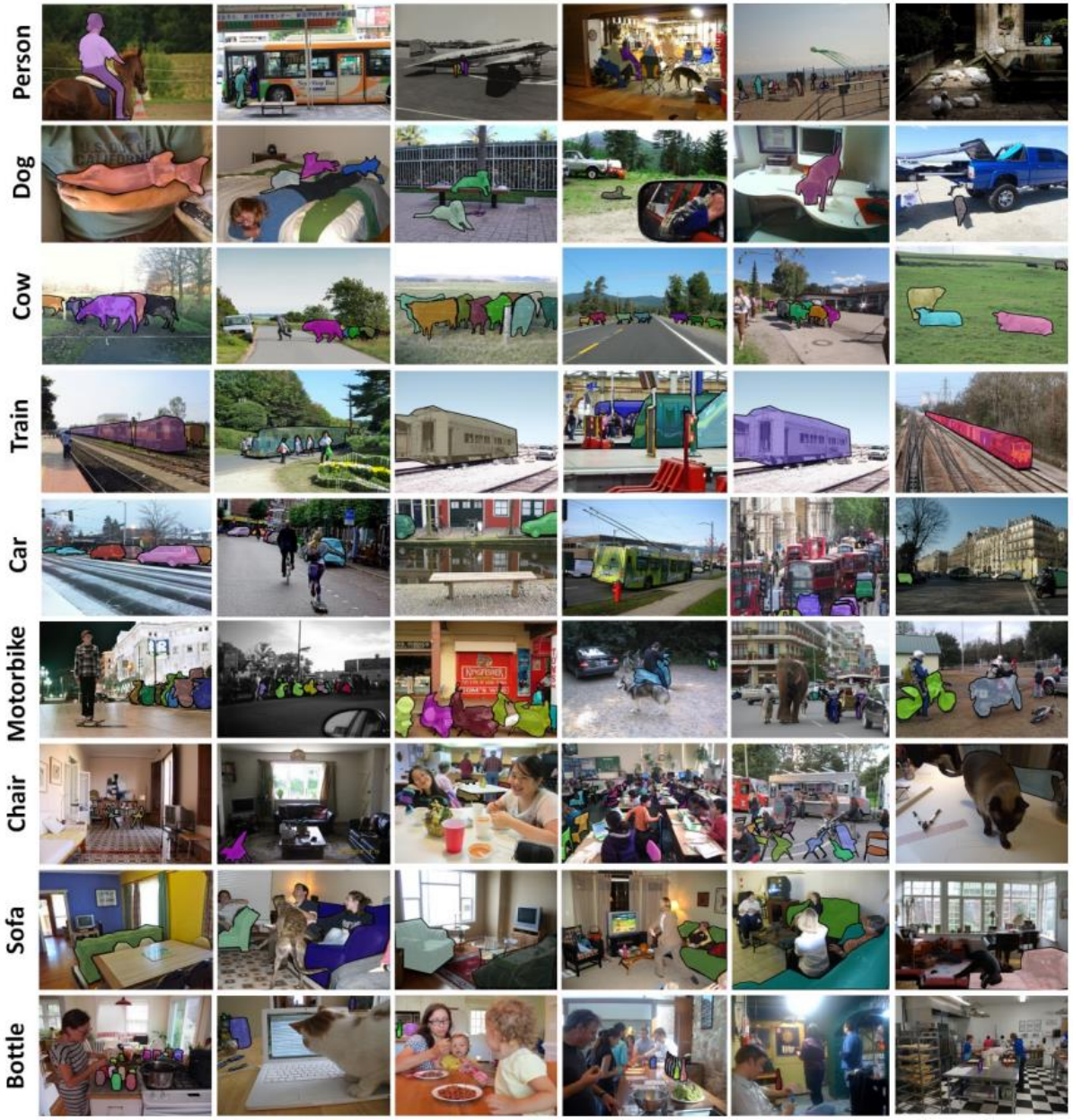
# Pascal VOC Data Set

- Pascal Visual Object Class challenge
- widely used as benchmark for object detection and semantic segmentation
- 20 object categories such as person, sofa, sheep, car, ...
- 11530 annotated images
- available annotations: pixel-level segmentation, bounding boxes, object classes



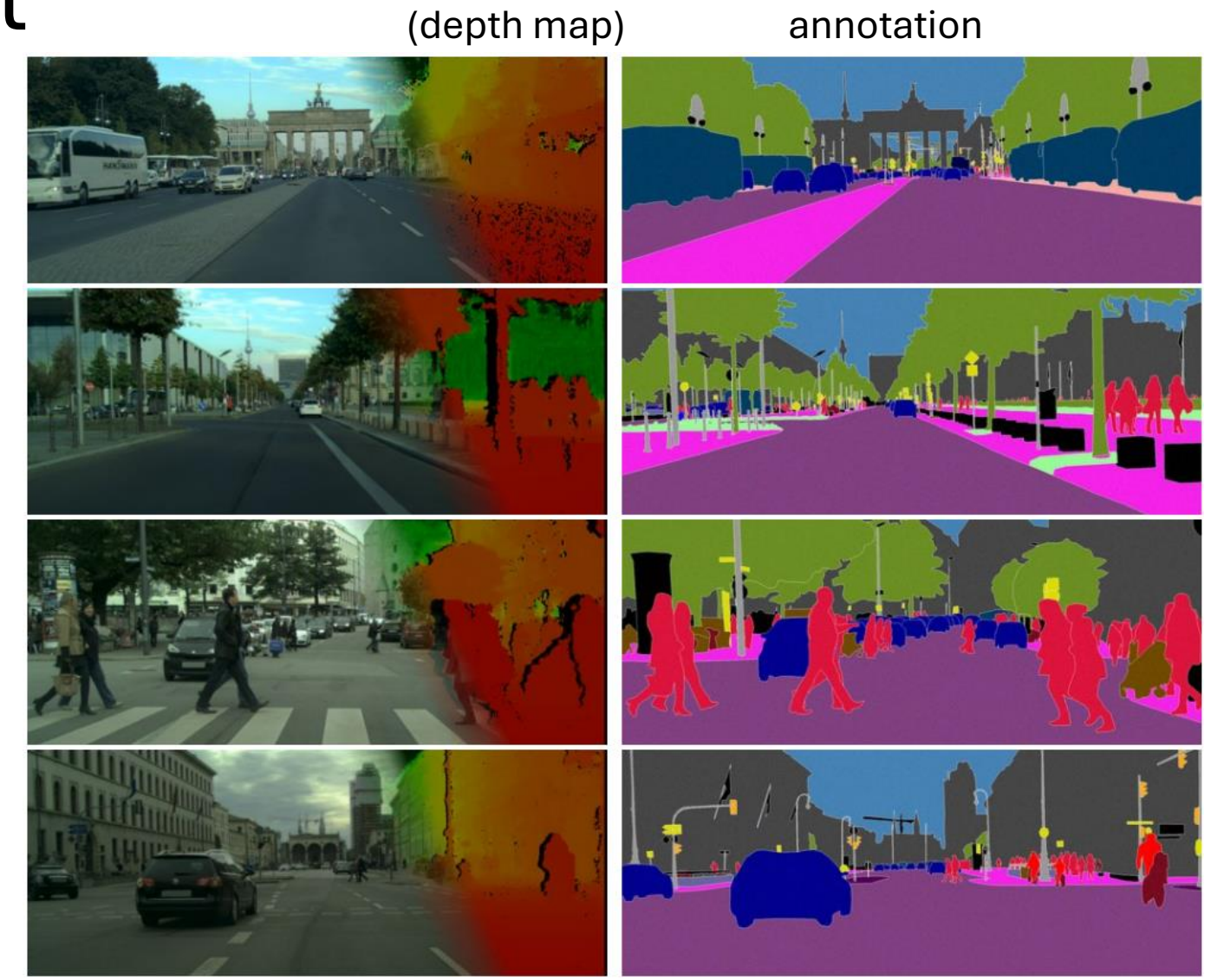

# MS COCO Data Set

- Microsoft Common Objects in Context
- images of complex everyday scenes containing common objects in their natural context
- 91 objects types
- 2.5 million annotated instances in 328k images → instance segmentation



# Cityscapes Data Set

- goal: semantic understanding of urban street scenes (captured in 50 cities)
- pixel annotations for 30 classes (person, car, building, …)
- 5000 fine-annotated and 20000 coarse-annotated images




source

# Semantic Segmentation

# Object Segmentation from DINO

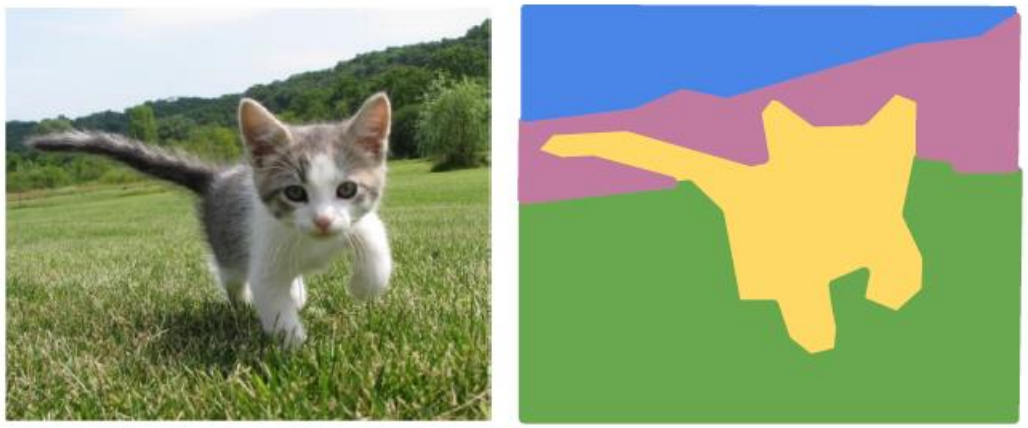
thresholding self-attention map of last layer:

source

not a full segmentation mask though …

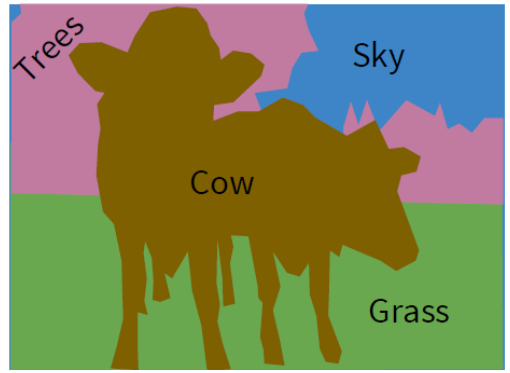
# Classification of Each Pixel

segmentation: no objects, just pixels



GRASS, CAT, TREE,
SKY, …

Paired training data: for each training image, each pixel is labeled with a semantic category.




At test time, classify each pixel of a new image.

minimize sum over classification losses
(cross-entropy loss at every output pixel)

# Idea: Fully Convolutional, No Downsampling

Input:
3 x H x W

Conv → Conv → Conv → Conv → argmax

stride 1,
padding
→ small kernels

feature maps → Convolutions:
D x H x W

Scores:
C x H x W

categories

Predictions:
H x W

replace flattened, fully-connected classification layers with $1 \times 1$ convolutions
→ maintain spatial relationships and enable pixel-wise classification:
conversion of feature maps into classification heat maps (one for each class)

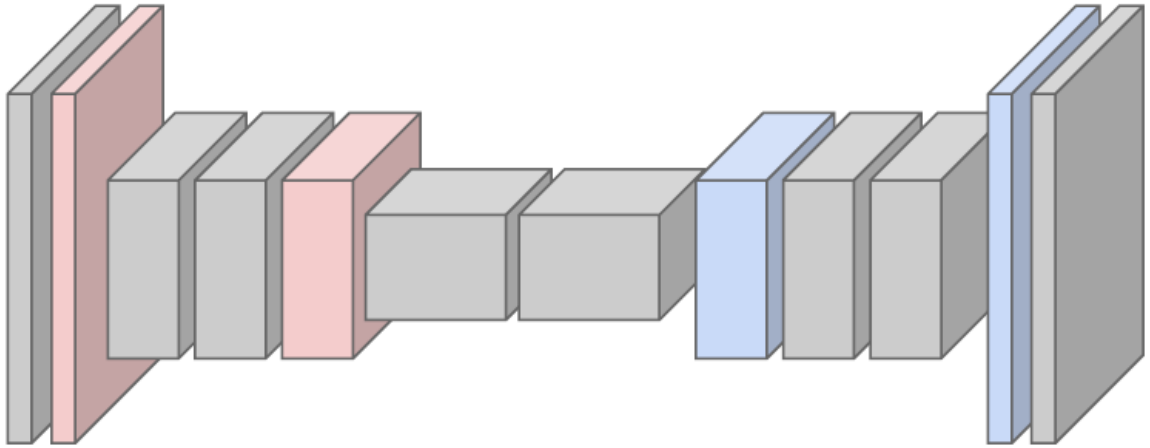
but no downsampling means small receptive field and no hierarchical learning

# Upsampling to the Rescue

typical spatial feature extraction: convolution and pooling layers → downsampling

upsampling layers to restore original image size

Input:
3 x H x W



C x H x W

pixel-wise classification

Predictions:
H x W

different options for down- (pooling, strided convolution) and upsampling …

# Reverse Pooling

## resampling (no learned parameters)

Nearest Neighbor

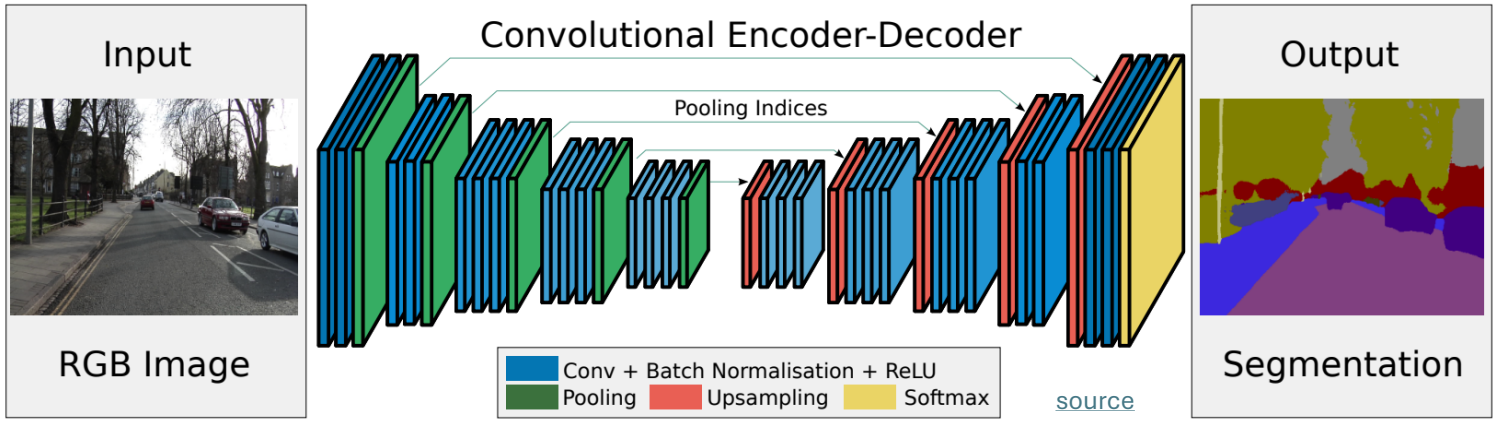
| | |
|---|---|
| 1 | 2 |
| 3 | 4 |

→

| | | | |
|---|---|---|---|
| 1 | 1 | 2 | 2 |
| 1 | 1 | 2 | 2 |
| 3 | 3 | 4 | 4 |
| 3 | 3 | 4 | 4 |

Input: 2 x 2 Output: 4 x 4

## unpooling (recording max positions from pooling)




Max Pooling
Remember which element was max!

| | | | |
|---|---|---|---|
| 1 | 2 | 6 | 3 |
| 3 | 5 | 2 | 1 |
| 1 | 2 | 2 | 1 |
| 7 | 3 | 4 | 8 |

→

| | |
|---|---|
| 5 | 6 |
| 7 | 8 |

→ • • • → Rest of the network

Input: 4 x 4 Output: 2 x 2

Max Unpooling
Use positions from pooling layer

| | |
|---|---|
| 1 | 2 |
| 3 | 4 |

→

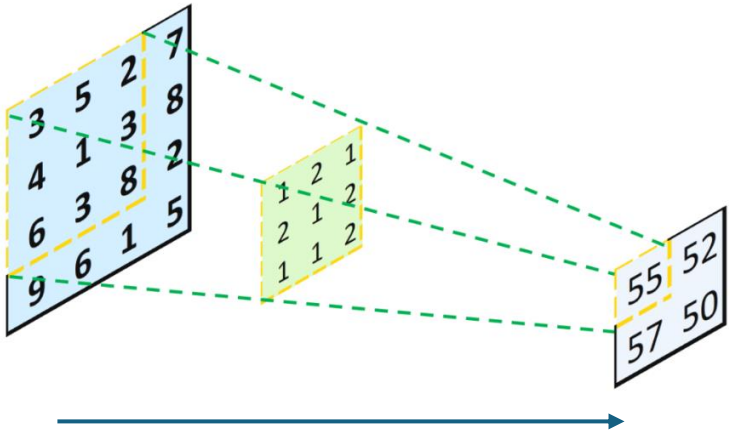
| | | | |
|---|---|---|---|
| 0 | 0 | 2 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 4 |

Input: 2 x 2 Output: 4 x 4

filled with learned parameters in subsequent convolution

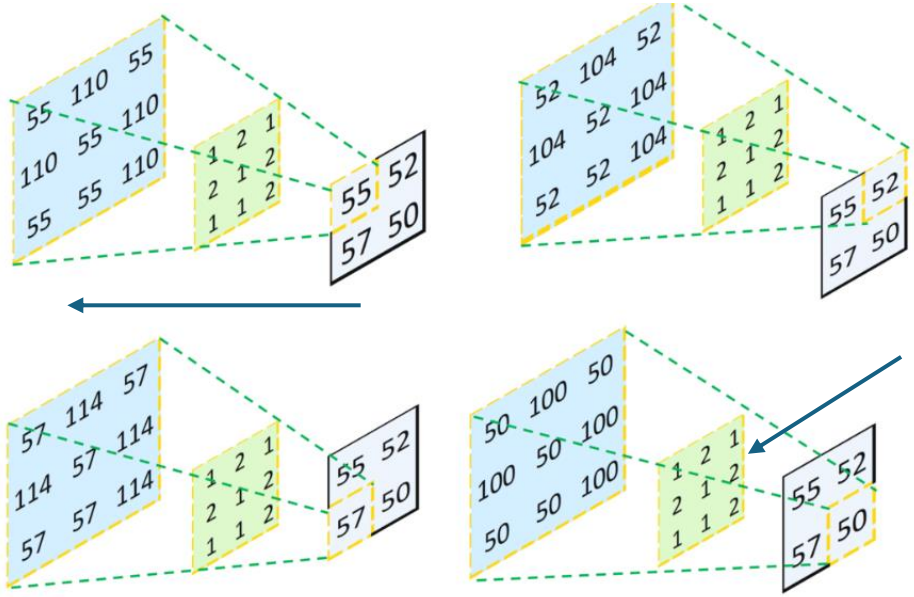
# Reverse Convolution

convolution

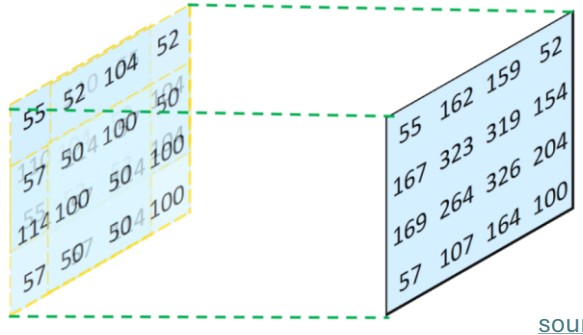



not inverse convolution
→ additional learned parameters

transposed convolution

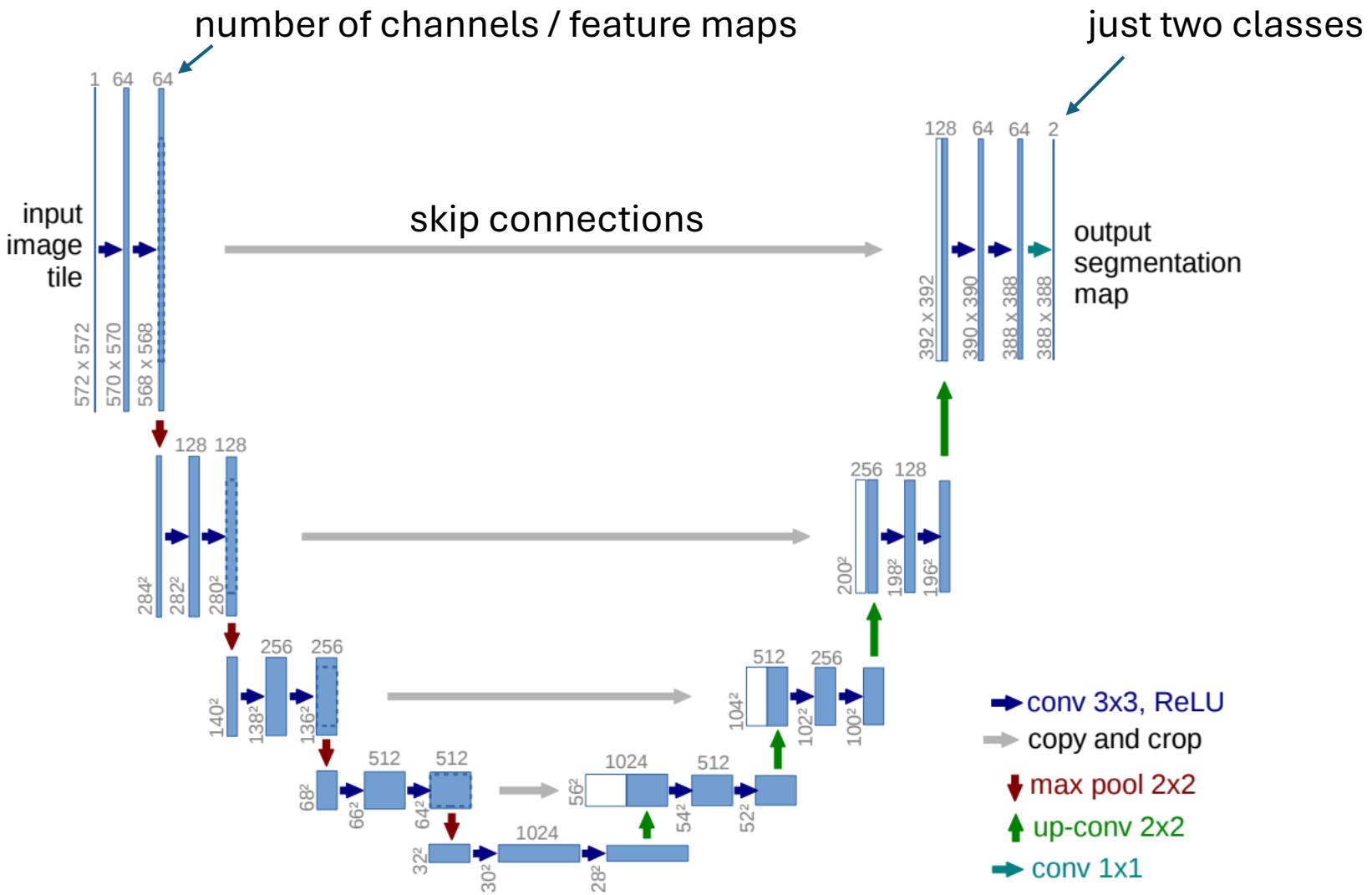



combine and sum overlaps:




source

# U-Net




also used for learning of depth maps, image synthesis (diffusion), ...

# Aside: Autoencoders

# Autoencoder

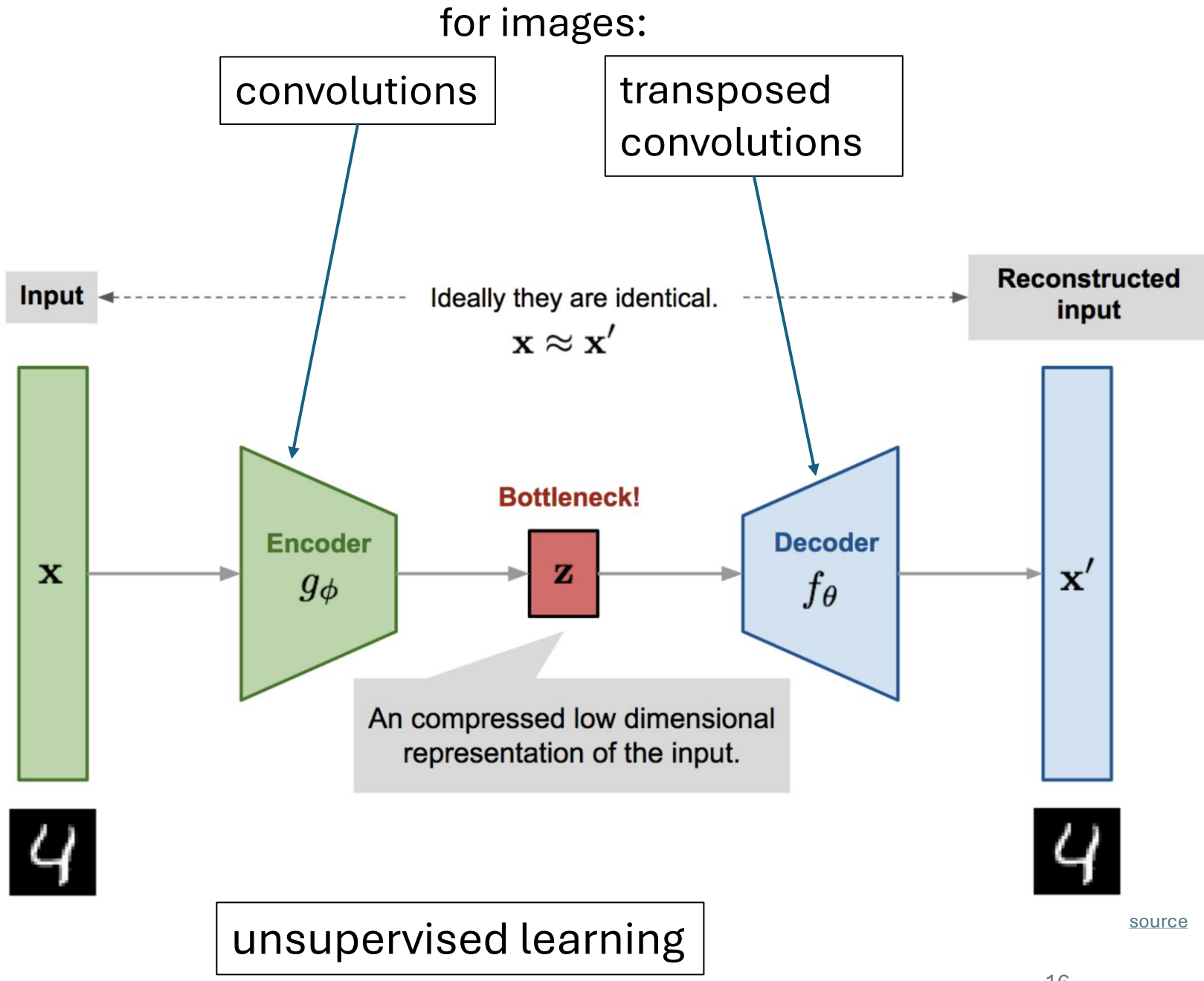
(deep) encoder network

(deep) decoder network

learned together by minimizing differences between original input and reconstructed input (expressed as losses)

compressed intermediate representation: dimensionality reduction

(alternative to PCA)



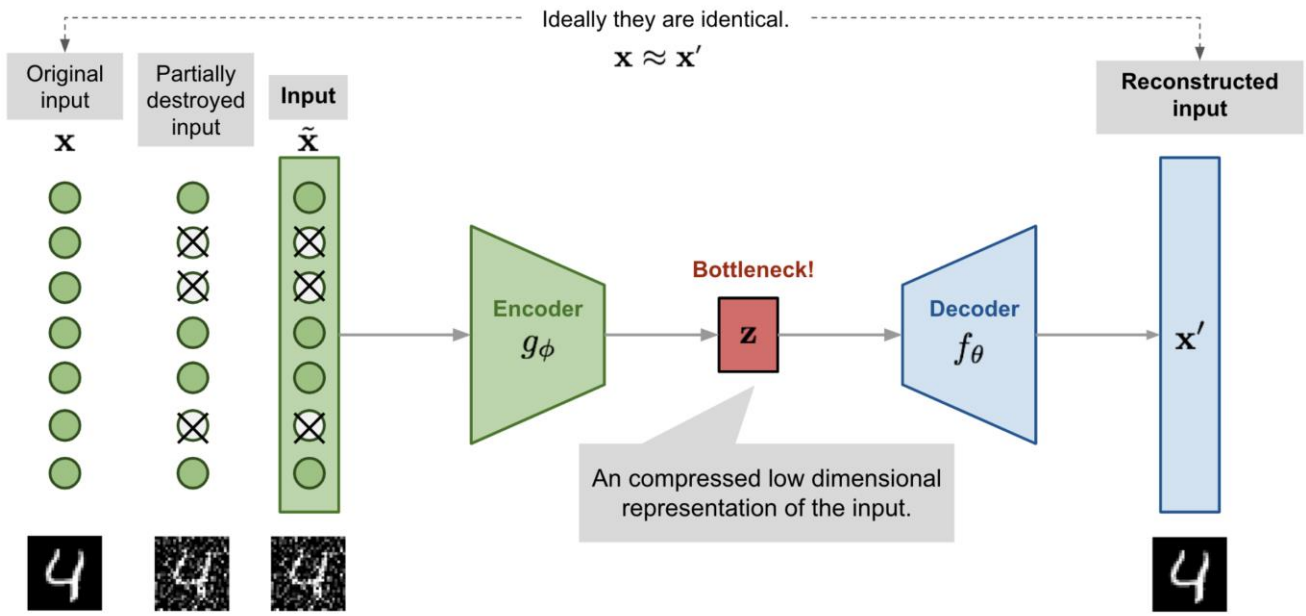

source

# Denoising Autoencoder

goal: avoid overfitting and improve robustness of plain autoencoder

learn to remove noise of distorted input $\tilde{x}$ → restore original input $x$

similar to dropout

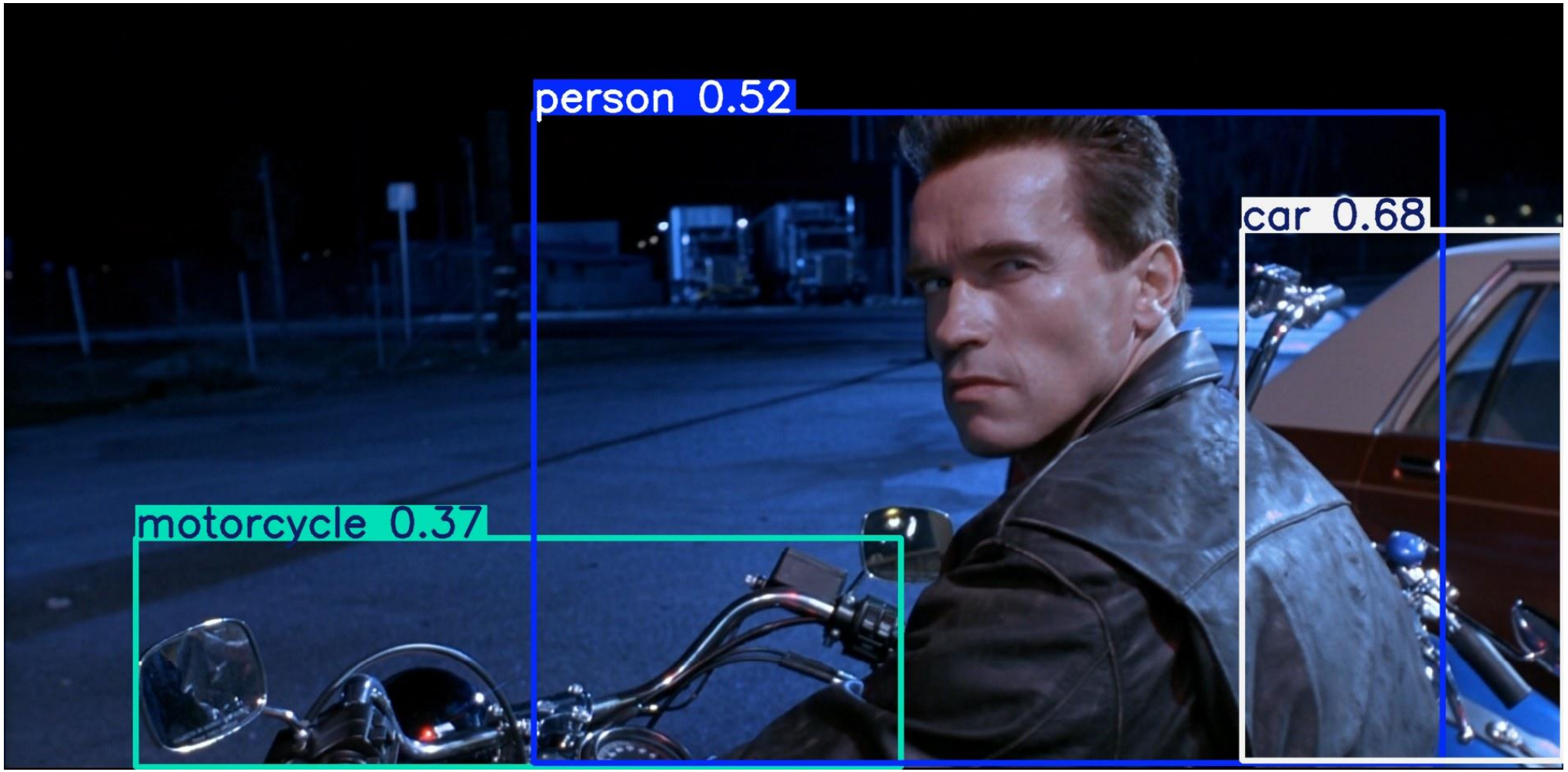



source

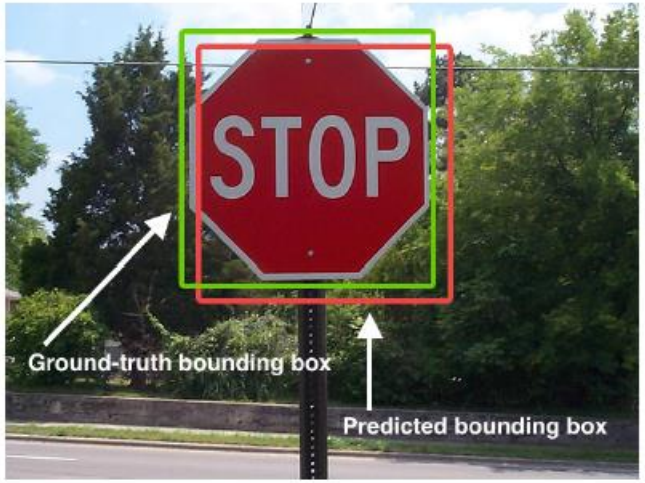
alternative to deconvolution (image restoration)

# Object Detection

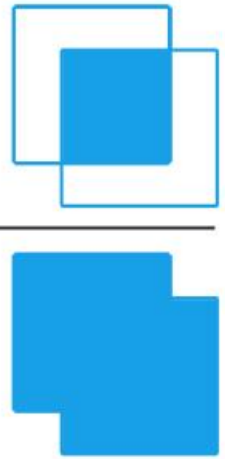
output: bounding boxes with category label and confidence



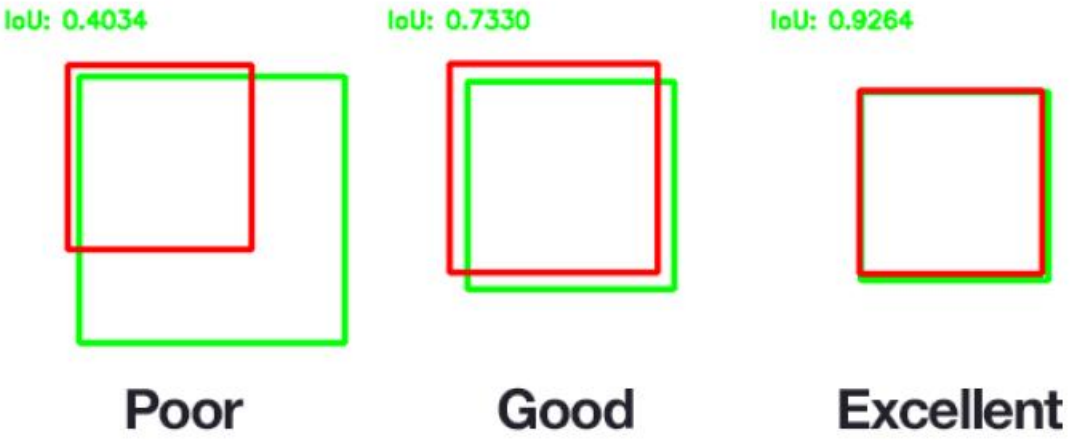
# Performance Evaluation of Localization




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



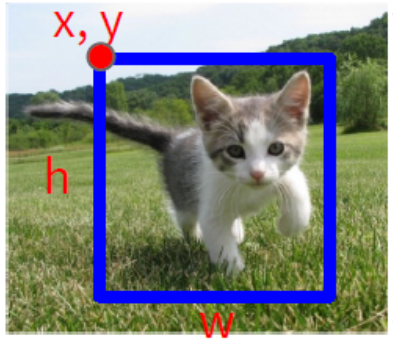
Intersection-over-Union

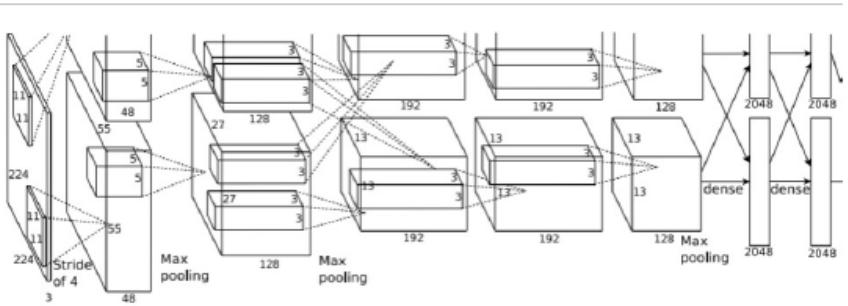



images with multiple objects: number of detections dependent on used confidence threshold

# Object Detection: Single Object

(Classification + Localization)




Vector: 4096

Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Softmax Loss

Correct label: Cat

Fully Connected: 4096 to 4

Box Coordinates (x, y, w, h)

L2 Loss

Correct box: (x', y', w', h')

Multitask Loss

+ → Loss

Treat localization as a regression problem!

too complicated for multiple objects

# Localization for Multiple-Object Images

idea: classify many different crops of the image as object or background

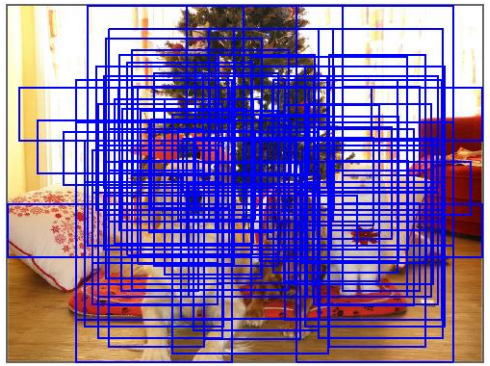
(crops: sliding window over image, iterated at multiple window sizes)

Dog? No
Cat? No
Background? Yes

Dog? Yes
Cat? No
Background? No

Dog? Yes
Cat? No
Background? No

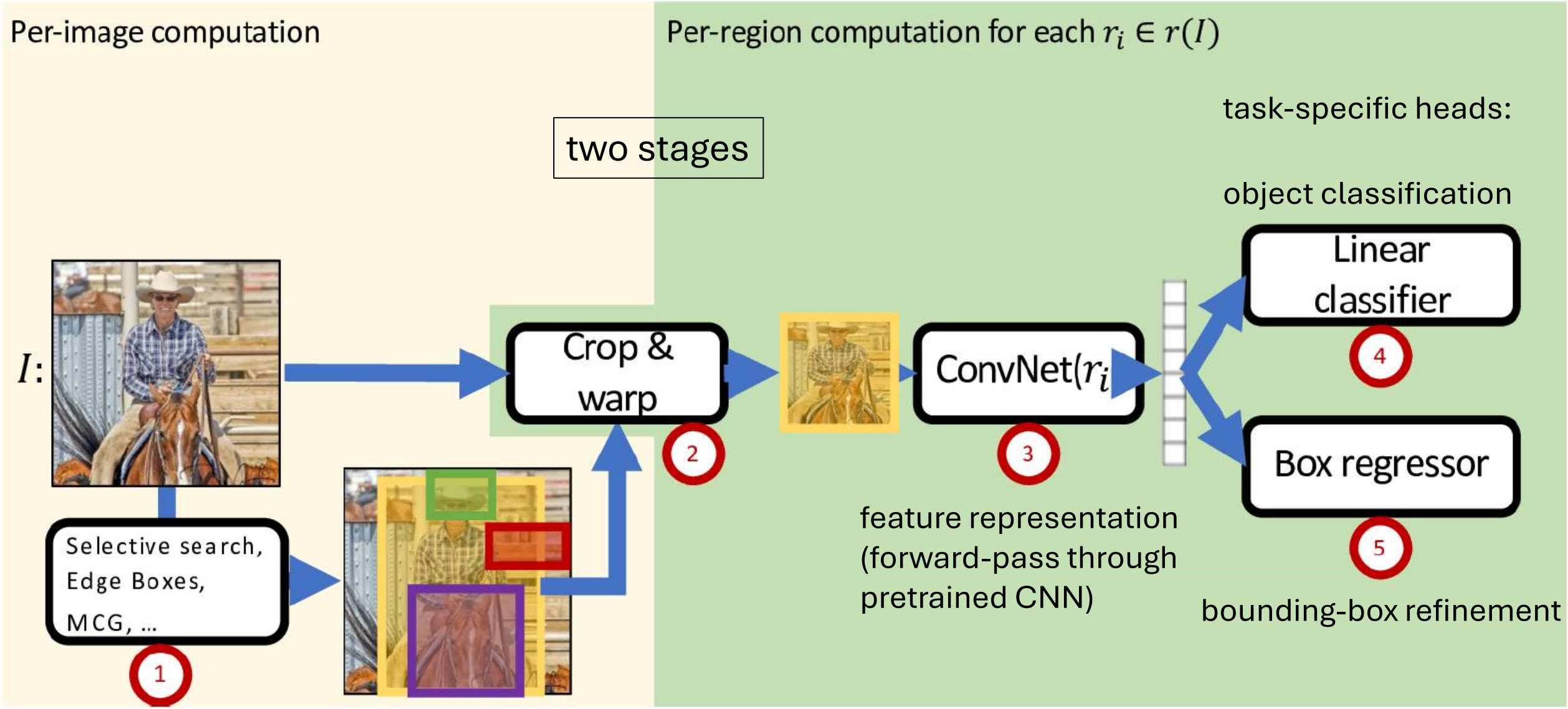
Dog? No
Cat? Yes
Background? No



issue: too many possible crops

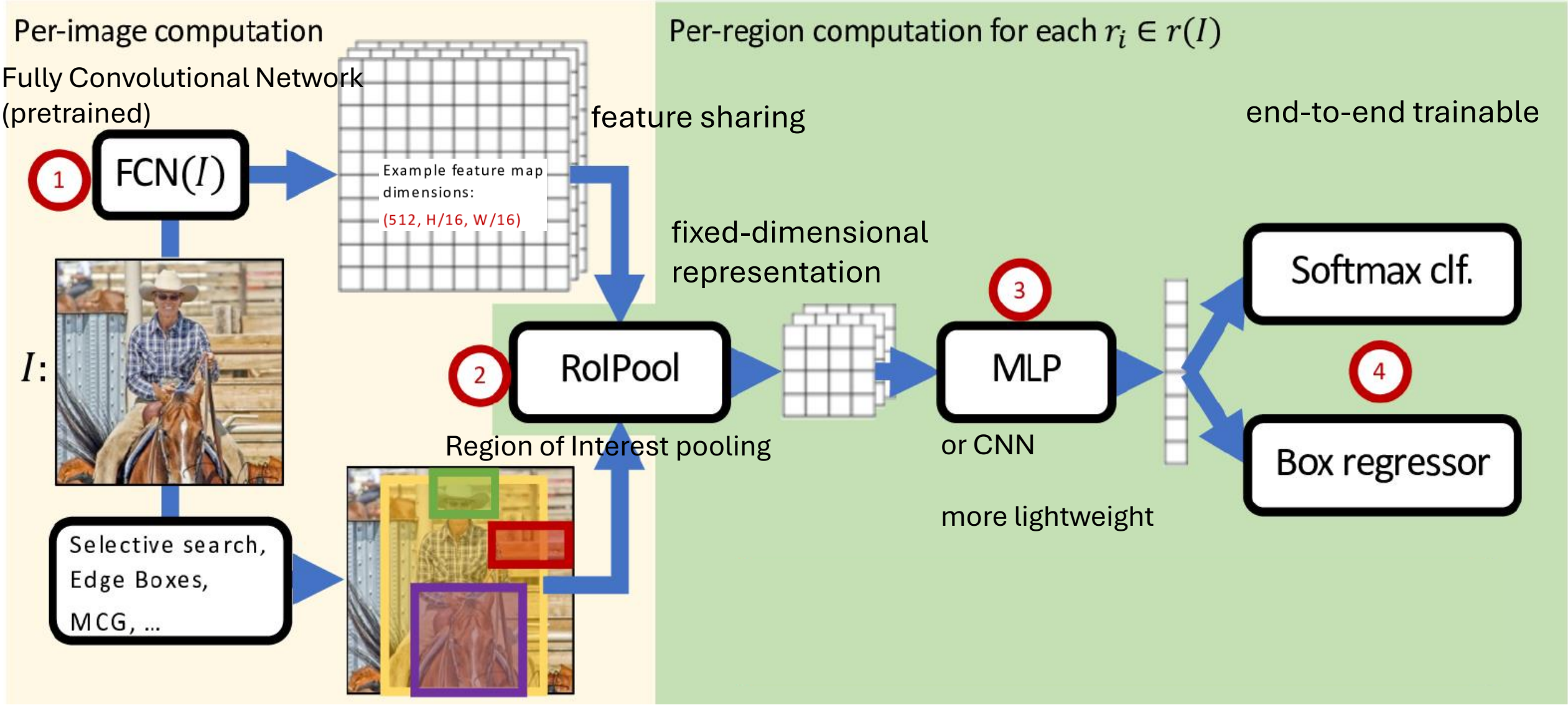
→ need for region proposals

# Region-Based Convolutional Neural Network (R-CNN)

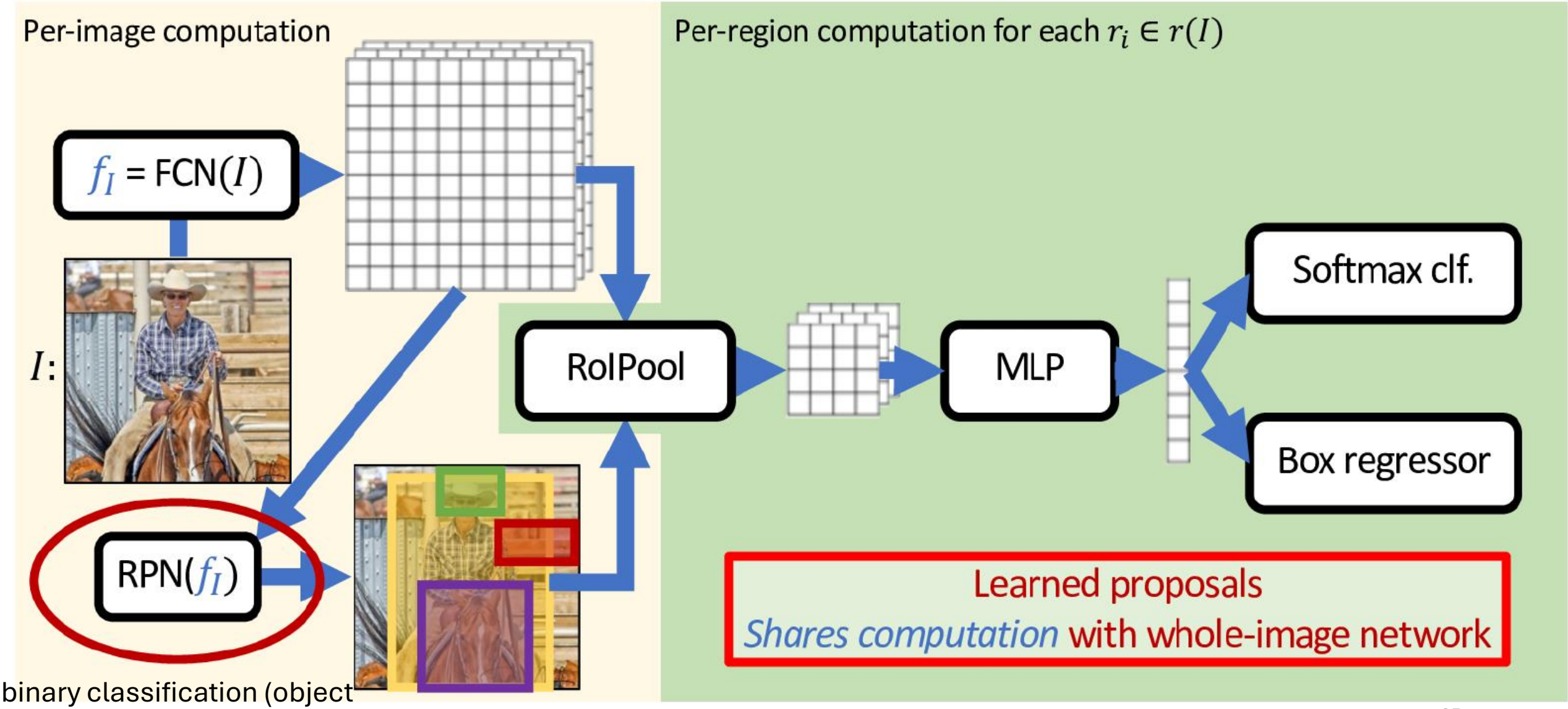



region proposals by "classic" method (~2k)

# Fast R-CNN: Crop ConvNet Features

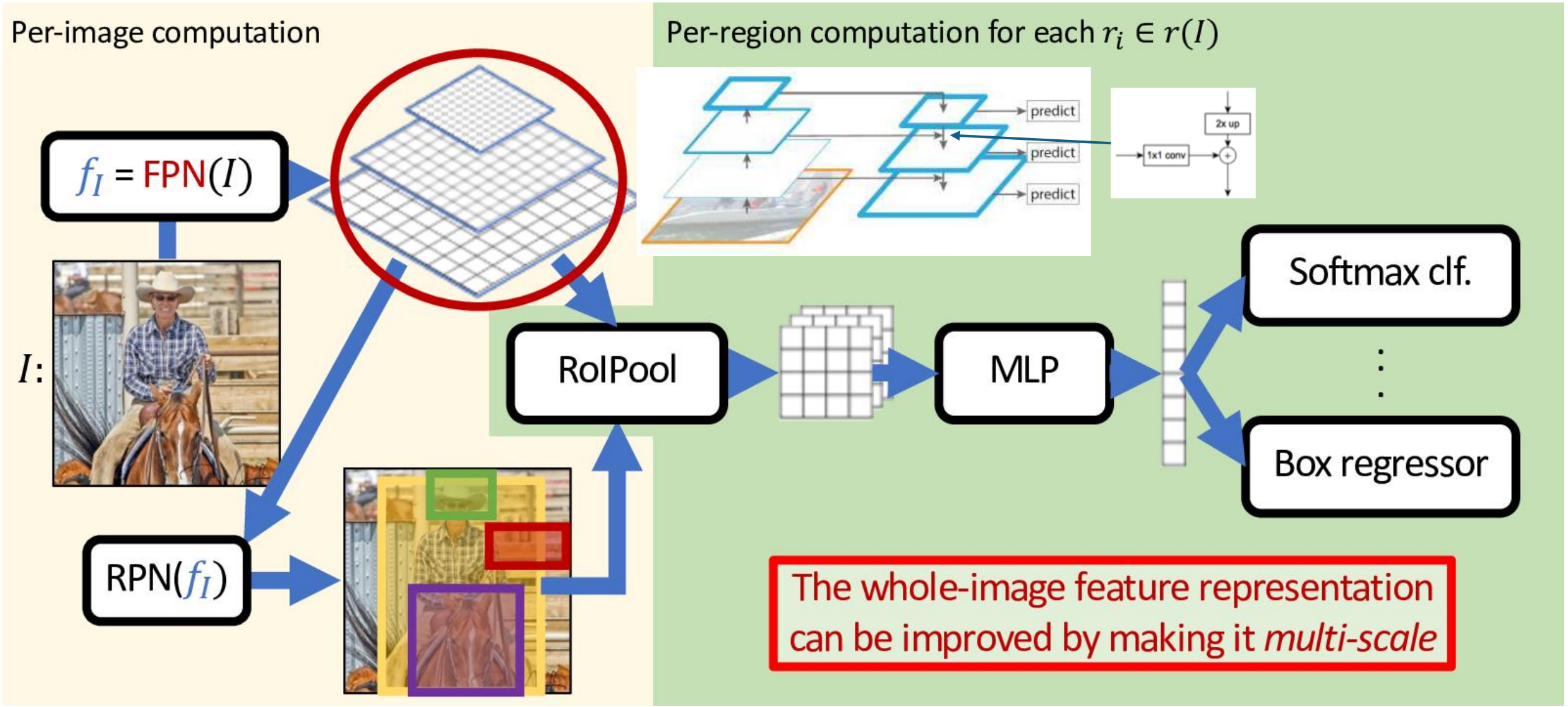


# Faster R-CNN: Use Region Proposal Network



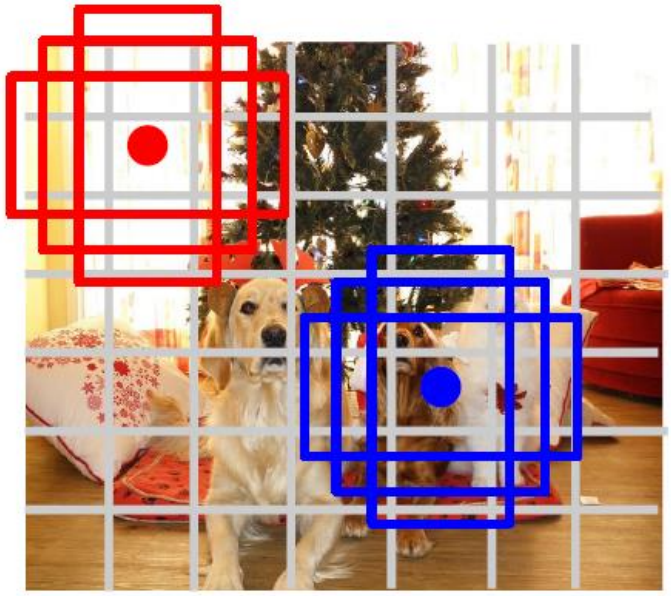

binary classification (object or not) with sliding window

# Feature Pyramid Network (FPN)



# Single-Stage Detectors: Drop Per-Region Computation

Input image
3 x H x W



Divide image into grid
7 x 7

Image a set of base boxes centered at each grid cell
Here B = 3

Within each grid cell:

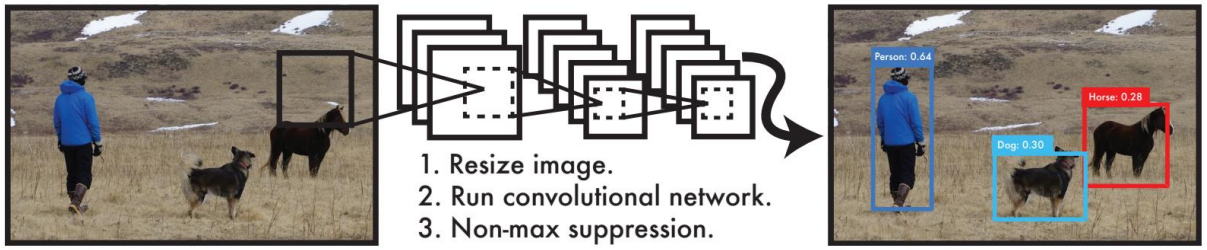
- Regress from each of the B base boxes to a final box with 5 numbers: (dx, dy, dh, dw, confidence)
- Predict scores for each of C classes (including background as a class)
- Looks a lot like RPN, but category-specific!

Output:
7 x 7 x (5 * B + C)

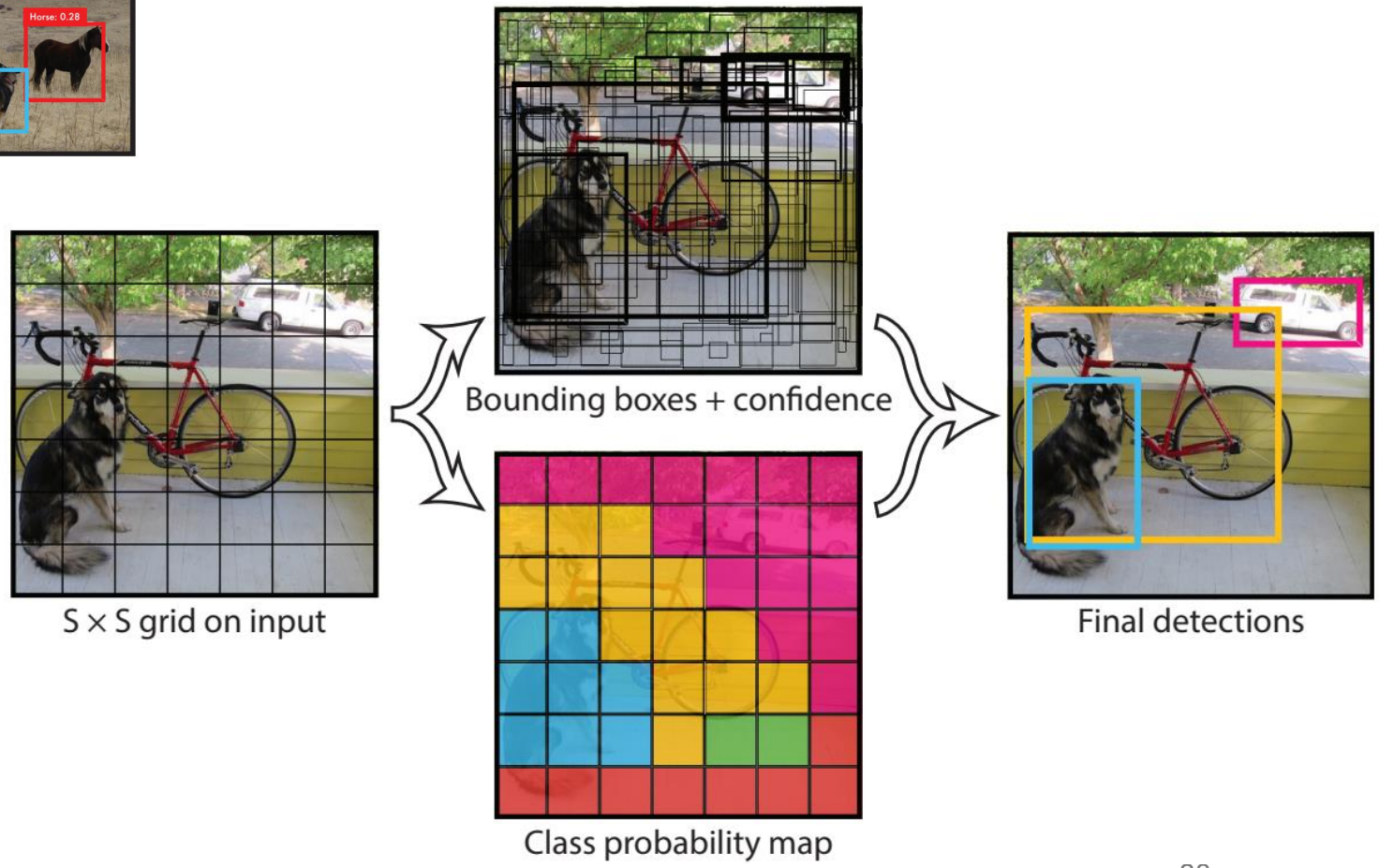
Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", CVPR 2016
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016
Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017

faster, but usually worse performance

# YOLO: Real-Time Object Detection




You Only Look Once:
prediction of bounding boxes
and class probabilities in one go



# Object Tracking

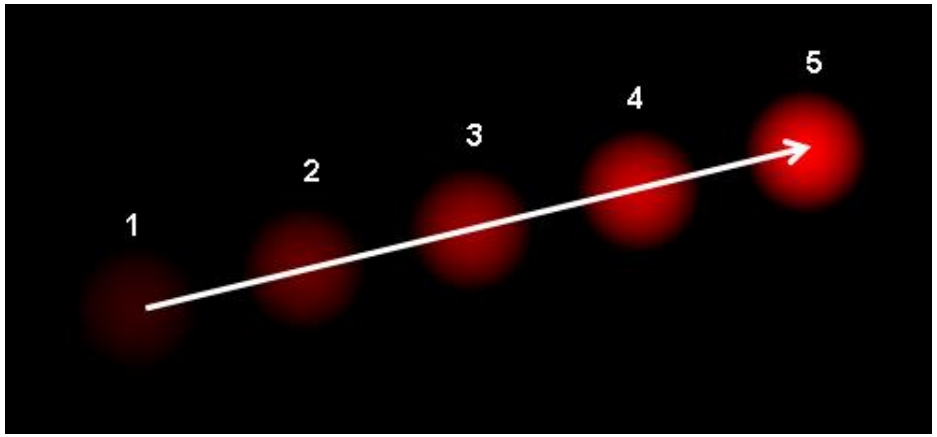
task: locating an object in successive video frames

basic approach:

1. detection: localize target object(s) (e.g., YOLO)
2. motion estimation: predict next location (e.g., Kalman filter or optical flow)
3. appearance matching: comparison of previous and predicted next location (e.g., feature descriptor, CNN features)

detection repeated frequently to correct for accumulated deviations

# Optical Flow




dense optical flow:

estimate motion vector of every pixel

e.g., Horn-Schunck method or Lucas-Kanade (LK) method

sparse optical flow:

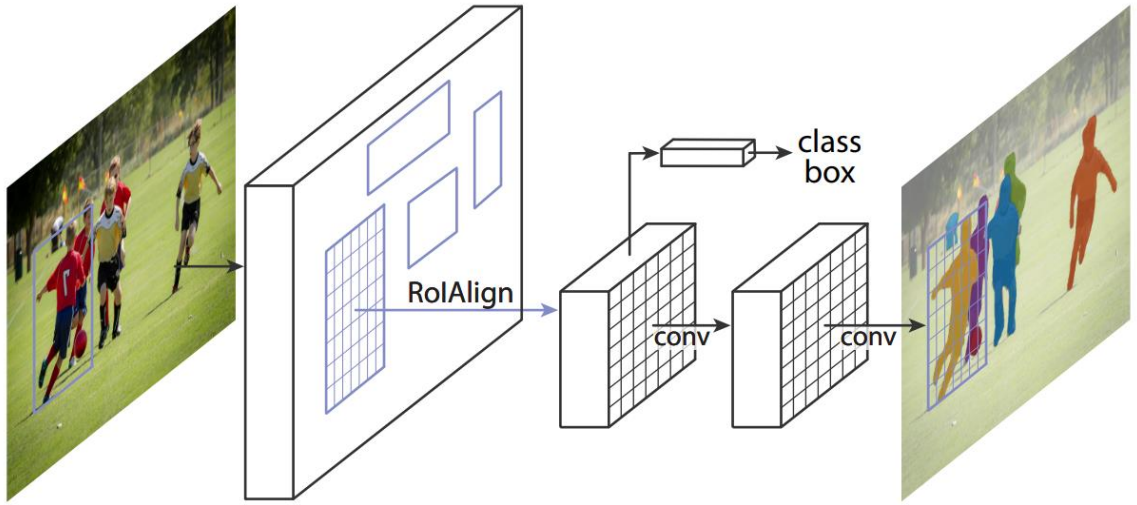
estimate motion vector of few image features

e.g., Kanade-Lucas-Tomasi (KLT) feature tracker

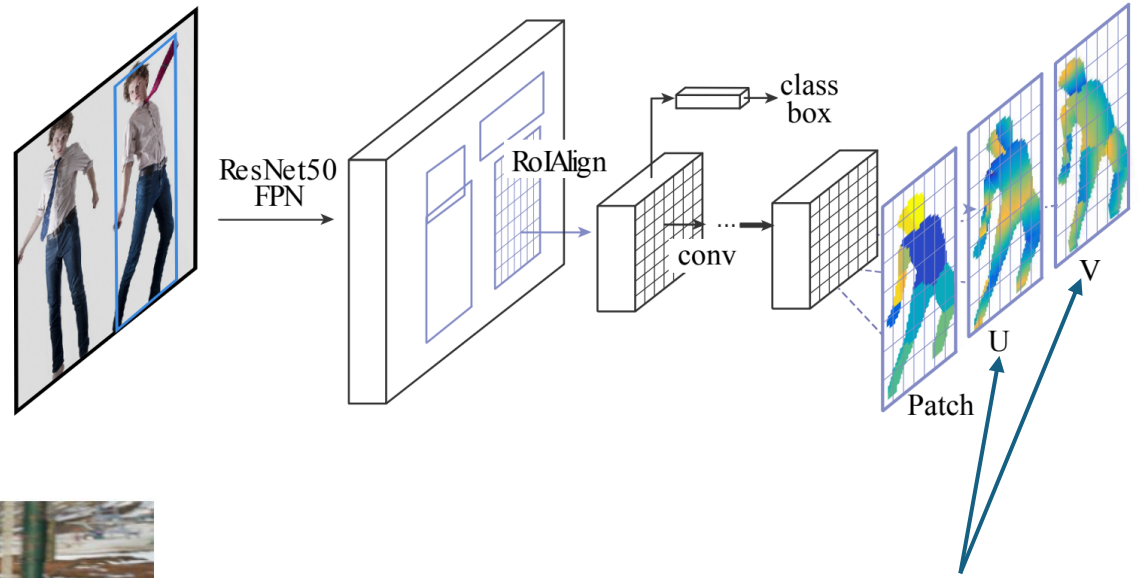
# Instance Segmentation

# Add Additional Network Heads to R-CNN

instance segmentation (Mask R-CNN):

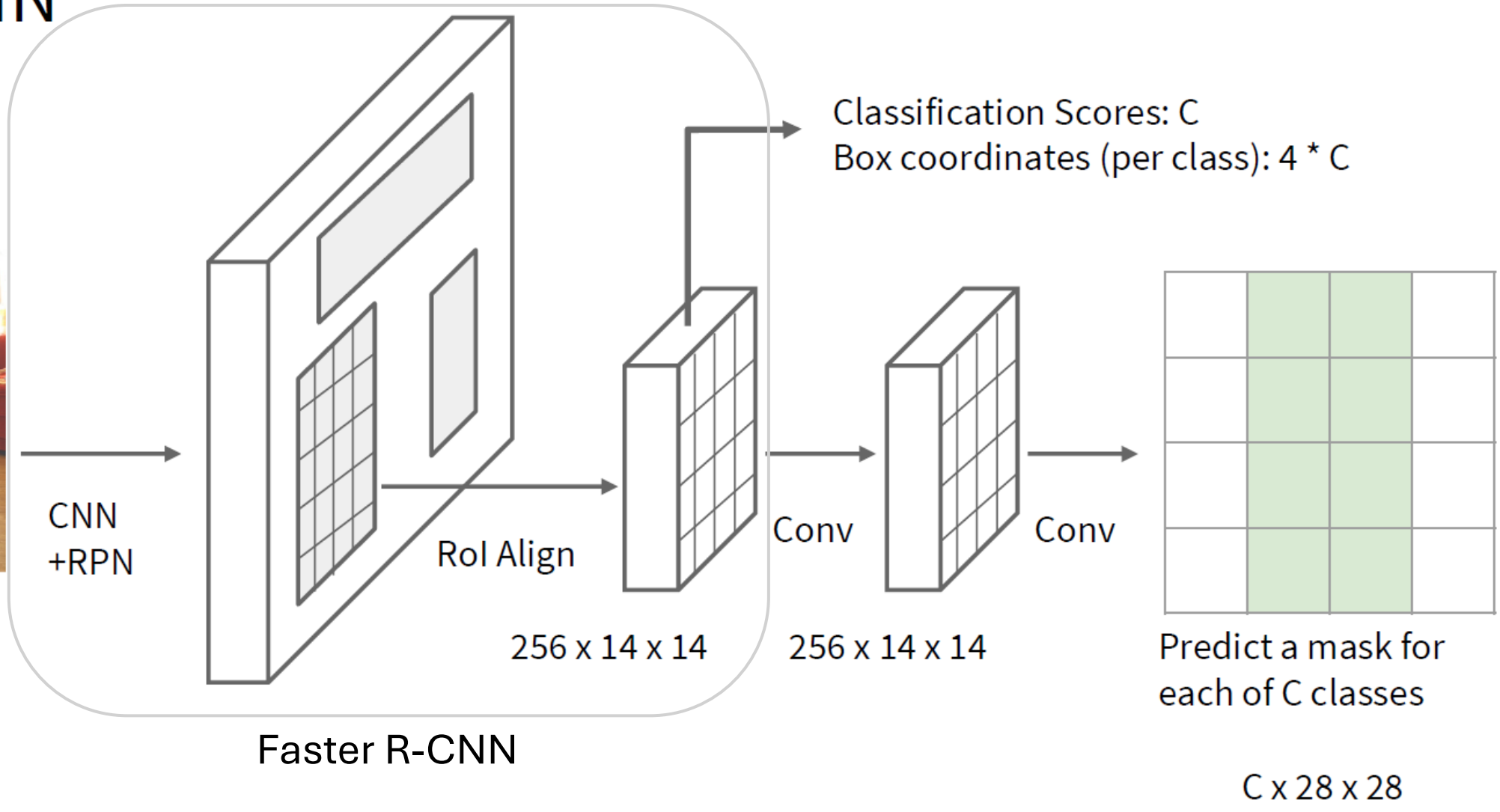



pose estimation (DensePose):



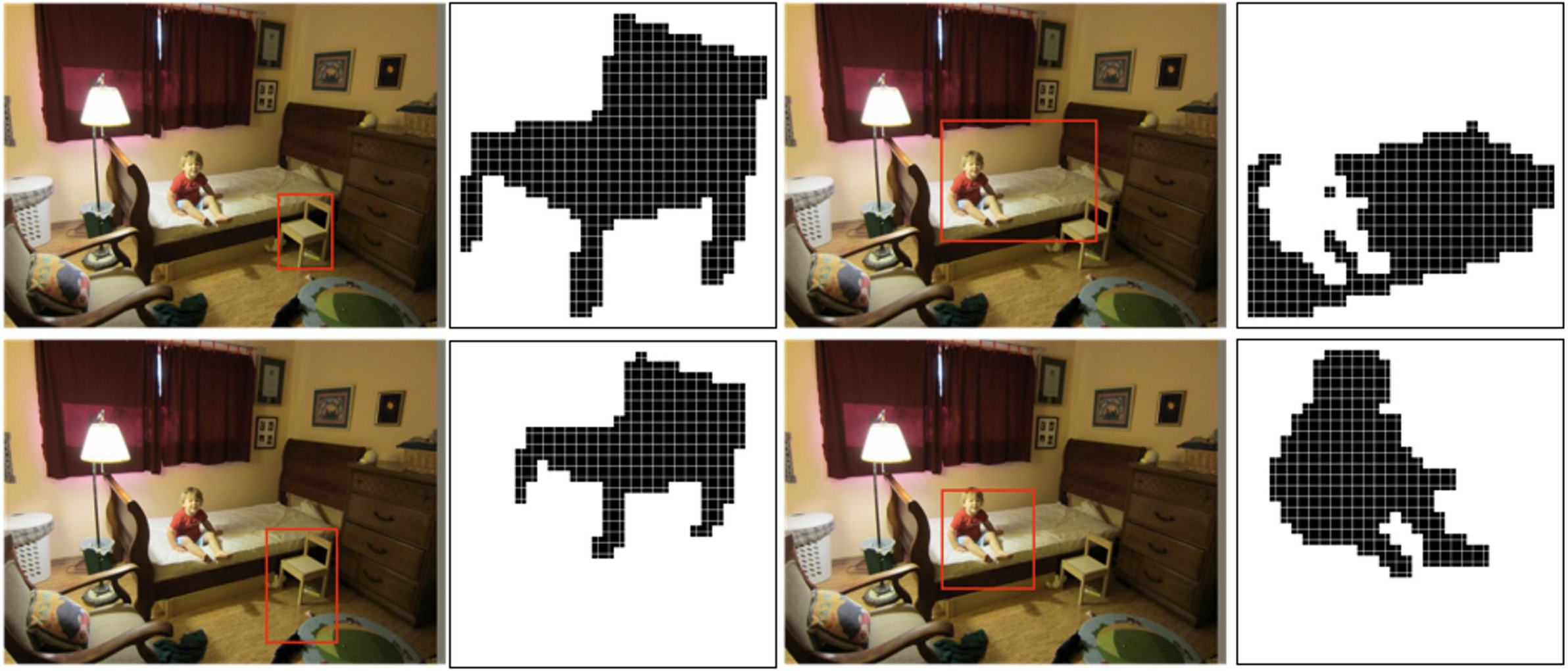

3D model's surface to 2D

# Mask R-CNN

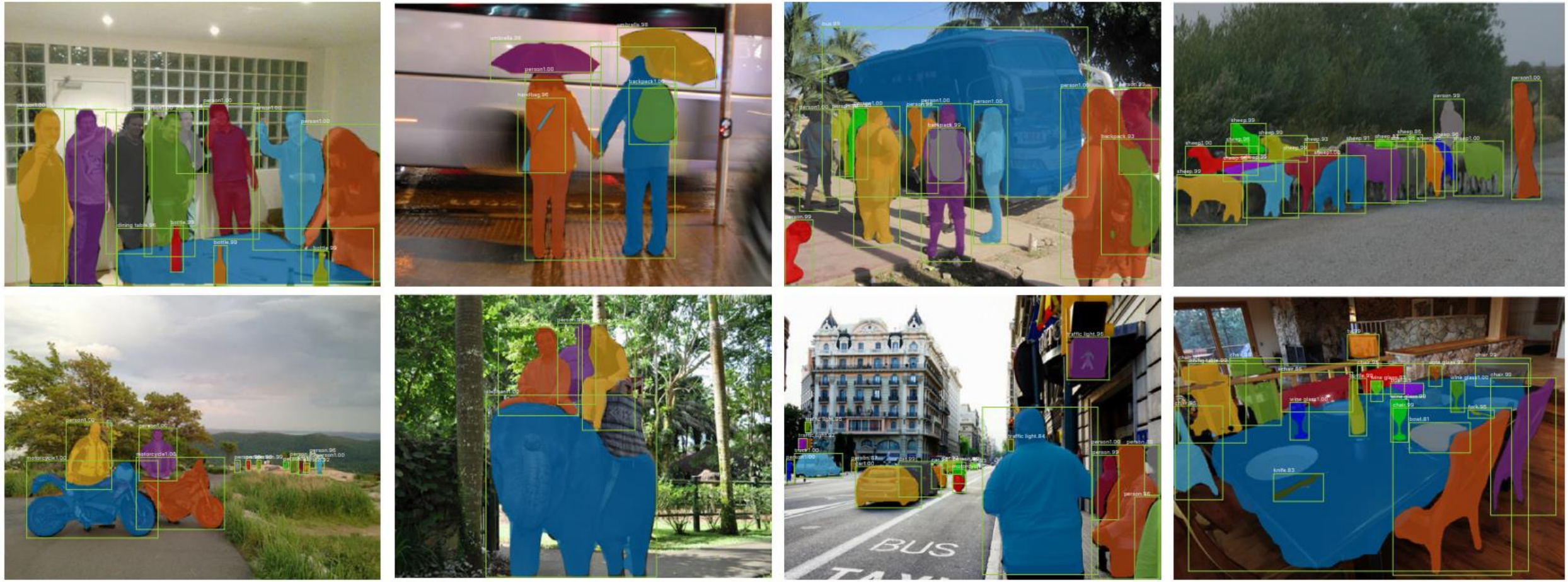



Faster R-CNN

image with training proposal | 28 × 28 mask target | image with training proposal | 28 × 28 mask target

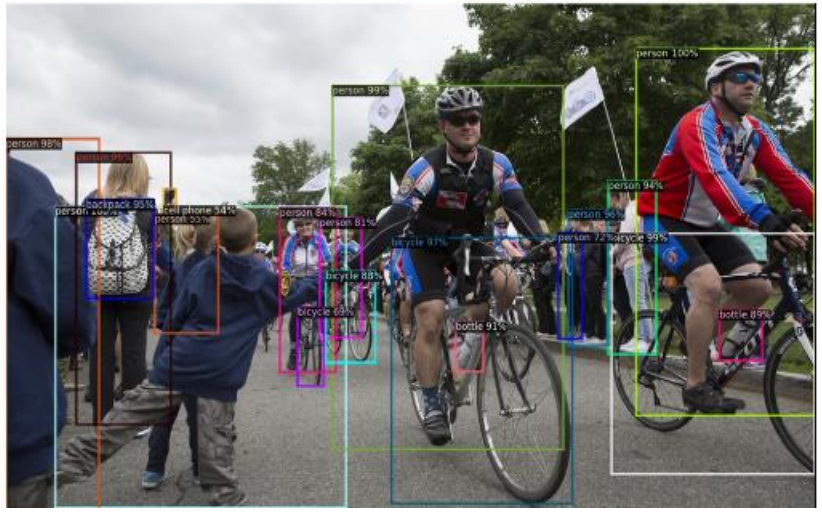


# results on MS COCO data set



# Detectron2: PyTorch Implementations






# Promptable Segmentation with Transformers

Segment Anything Model (SAM)

SAM2 includes also video segmentation