Deep Learning

Object Detection

Lionel Fillatre

2025-2026

Outline

- Introduction
- FCN for Semantic Segmentation
- Object Detection
- R-CNN
- Fast R-CNN
- Faster R-CNN
- Mask R-CNN
- Conclusion

Introduction

So far: Image Classification

This image is CC0 public domain

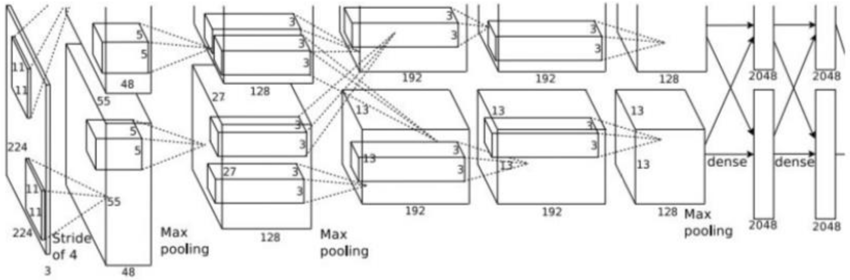


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Vector:
4096

Fully-Connected:
4096 to 1000

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

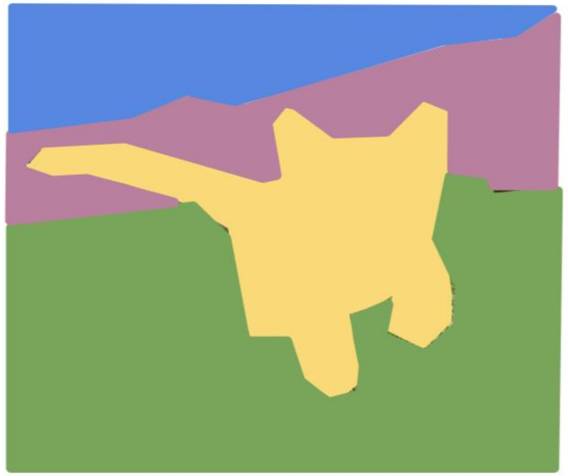
Computer Vision Tasks

Classification

CAT

No spatial extent

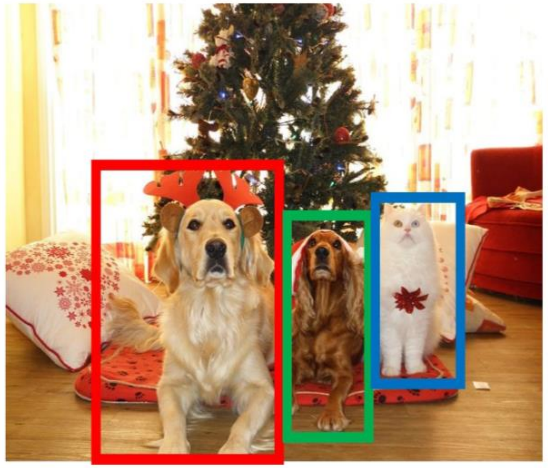
Semantic Segmentation



GRASS, CAT, TREE, SKY

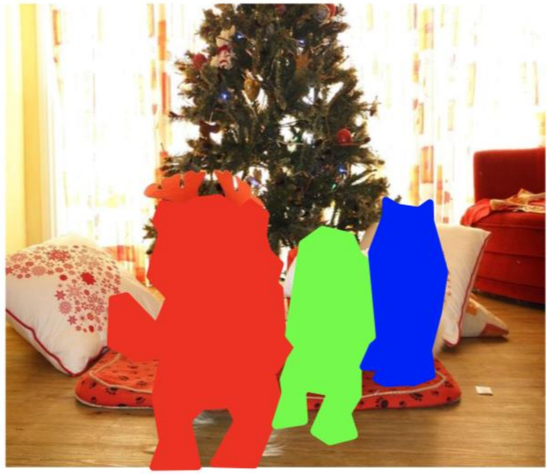
No objects, just pixels

Object Detection



DOG, DOG, CAT

Instance Segmentation



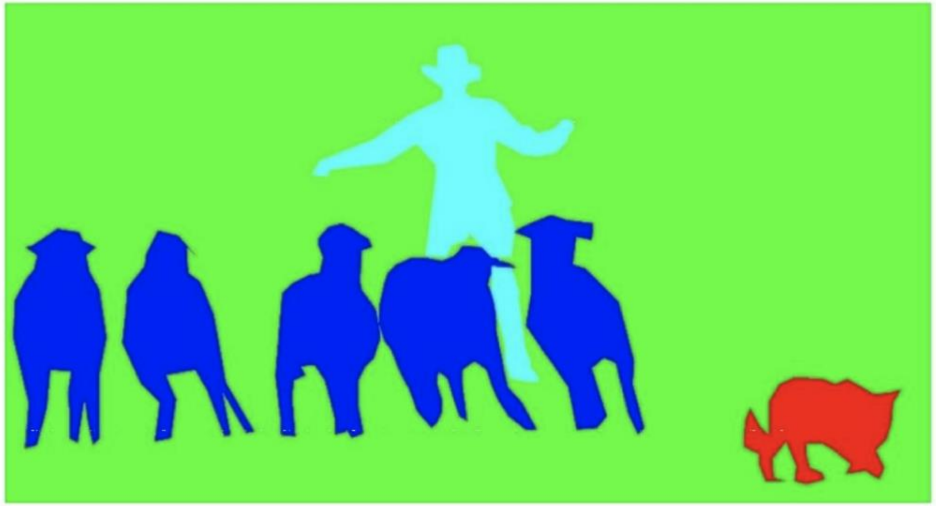
DOG, DOG, CAT

Multiple Object

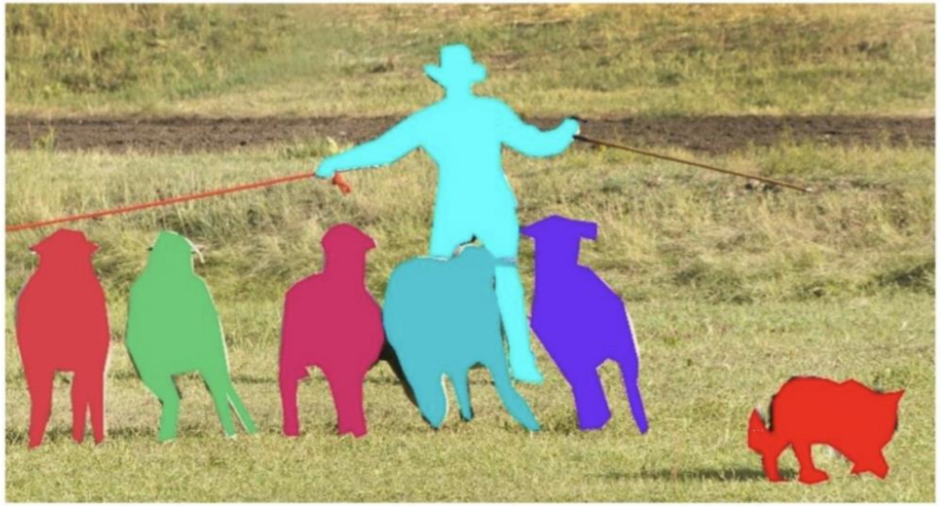
Fully Convolutional Networks (FCN) for Semantic Segmentation

Semantic Segmentation

- Semantic segmentation refers to the process of linking each pixel in an image to a class label. These labels could include a person, car, flower, piece of furniture, etc.
  - We can think of semantic segmentation as image classification at a pixel level.
- A separate class of models known as instance segmentation is able to label the separate instances where an object appears in an image.
  - This kind of segmentation can be very useful in applications that are used to count the number of objects, such as counting the amount of foot traffic in a mall.

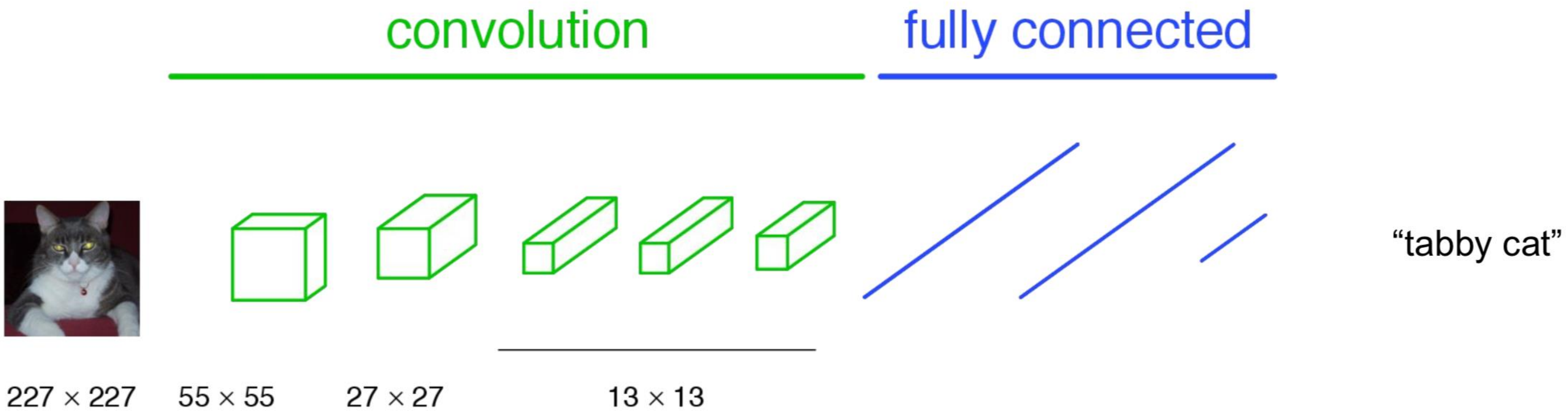


Semantic segmentation

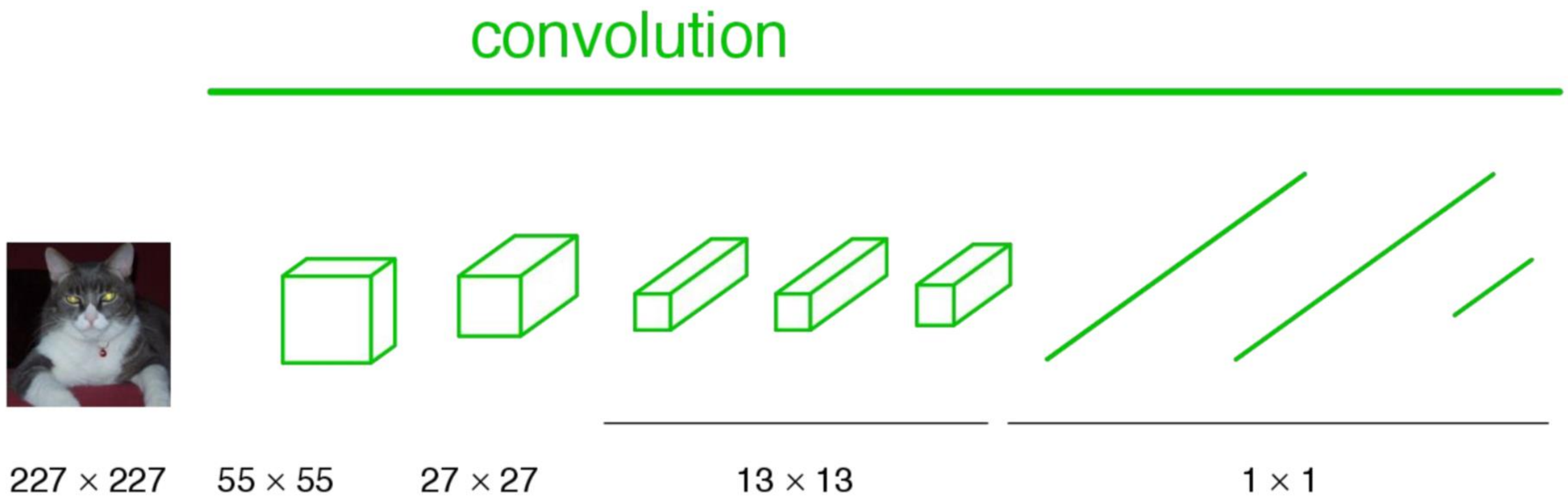


instance segmentation

Usual classification network (a.k.a. CNN)



Equivalent Fully Convolutional Network (FCN)



Equivalence between FC and convolution 1×1

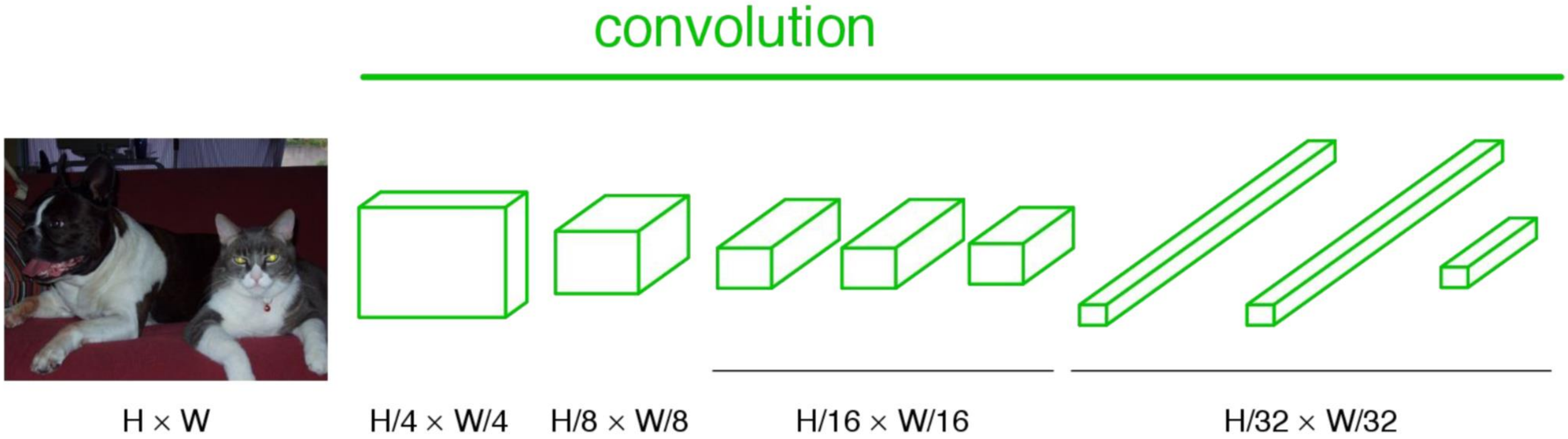
Fully Connected Version

```
>>> inputs = torch.tensor([[[[1., 2.],  [3., 4.]]]])
>>> fc = torch.nn.Linear(4, 2)
>>> weights = torch.tensor([[1.1, 1.2, 1.3, 1.4],
                            [1.5, 1.6, 1.7, 1.8]])
>>> bias = torch.tensor([1.9, 2.0])
>>> fc.weight.data = weights
>>> fc.bias.data = bias
>>> flat_inputs = inputs.view(-1, 4)
>>> fc_outputs = torch.relu(fc(flat_inputs))
>>> print(fc_outputs.view(-1, 2))
                tensor([[14.9000, 19.0000]])
```

1×1 Convolution Version

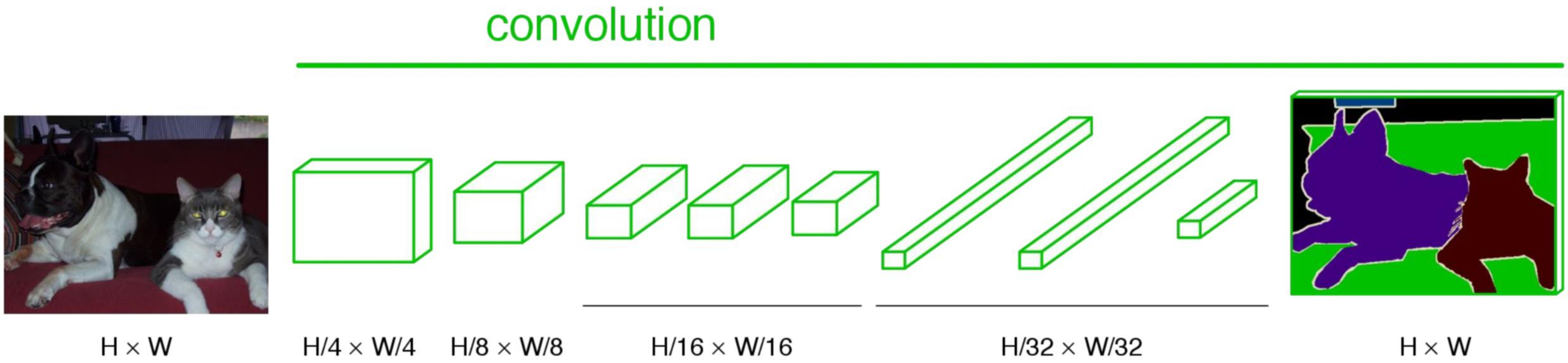
```
>>> conv = torch.nn.Conv2d(in_channels=4,
                           out_channels=2,
                           kernel_size=(1, 1))

>>> conv.weight.data = weights.view(2, 4, 1, 1)
>>> conv.bias.data = bias
>>> reshaped_inputs = inputs.view(1, 4, 1, 1)
>>> conv_outputs =  torch.relu(conv(reshaped_inputs))
>>> print(conv_outputs.view(-1, 2))
                tensor([[14.9000, 19.0000]])
```

A more general FCN



Upsampling output

- The size of the network output is too small for annoting all the pixels of the input image
  - We need to upsample the output



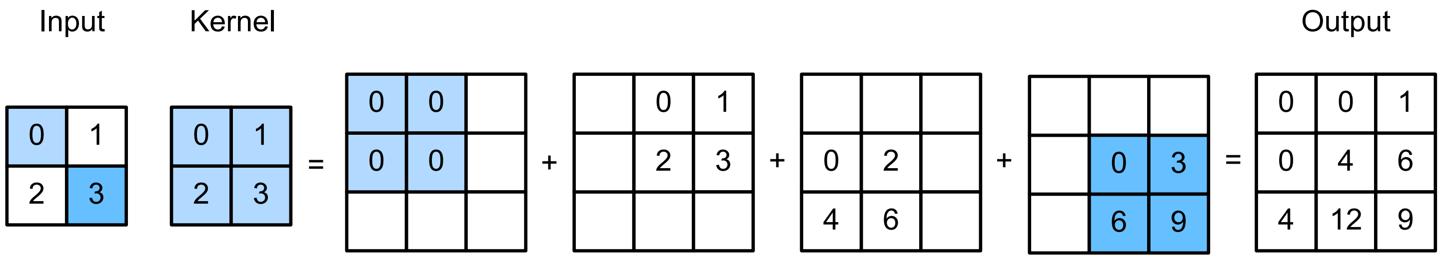
https://towardsdatascience.com/review-fcn-semantic-segmentation-eb8c9b50d2d1

Upsampling a layer

- Some generative processes optimize the input, and as such rely on backpropagation to expand the signal from a low-dimension representation to the high-dimension signal space.
- The same can be done in the forward pass with transposed convolution layers whose forward operation corresponds to a convolution layer’s backward pass.
- Fractionally strided convolutions, also referred to as deconvolutions or transposed convolutions, transpose images, typically from a minimized format to a larger one.

Transposed convolution with a 2×2 kernel

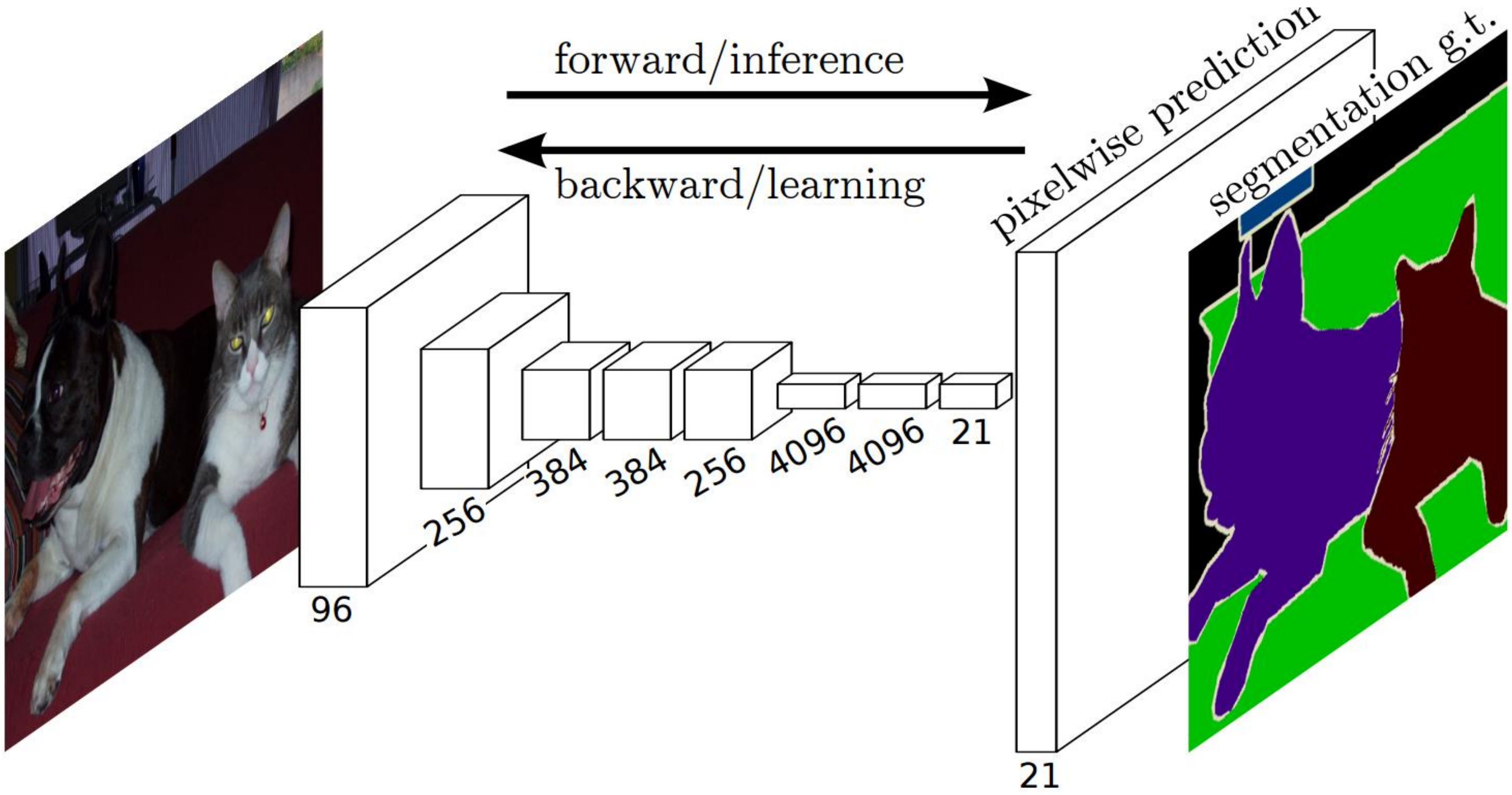
- Let us consider a basic case that both input and output channels are 1, with 0 padding and 1 stride.
- It illustrates how transposed convolution with a 2×2 kernel is computed on the 2×2 input matrix.



https://distill.pub/2016/deconv-checkerboard/#citation

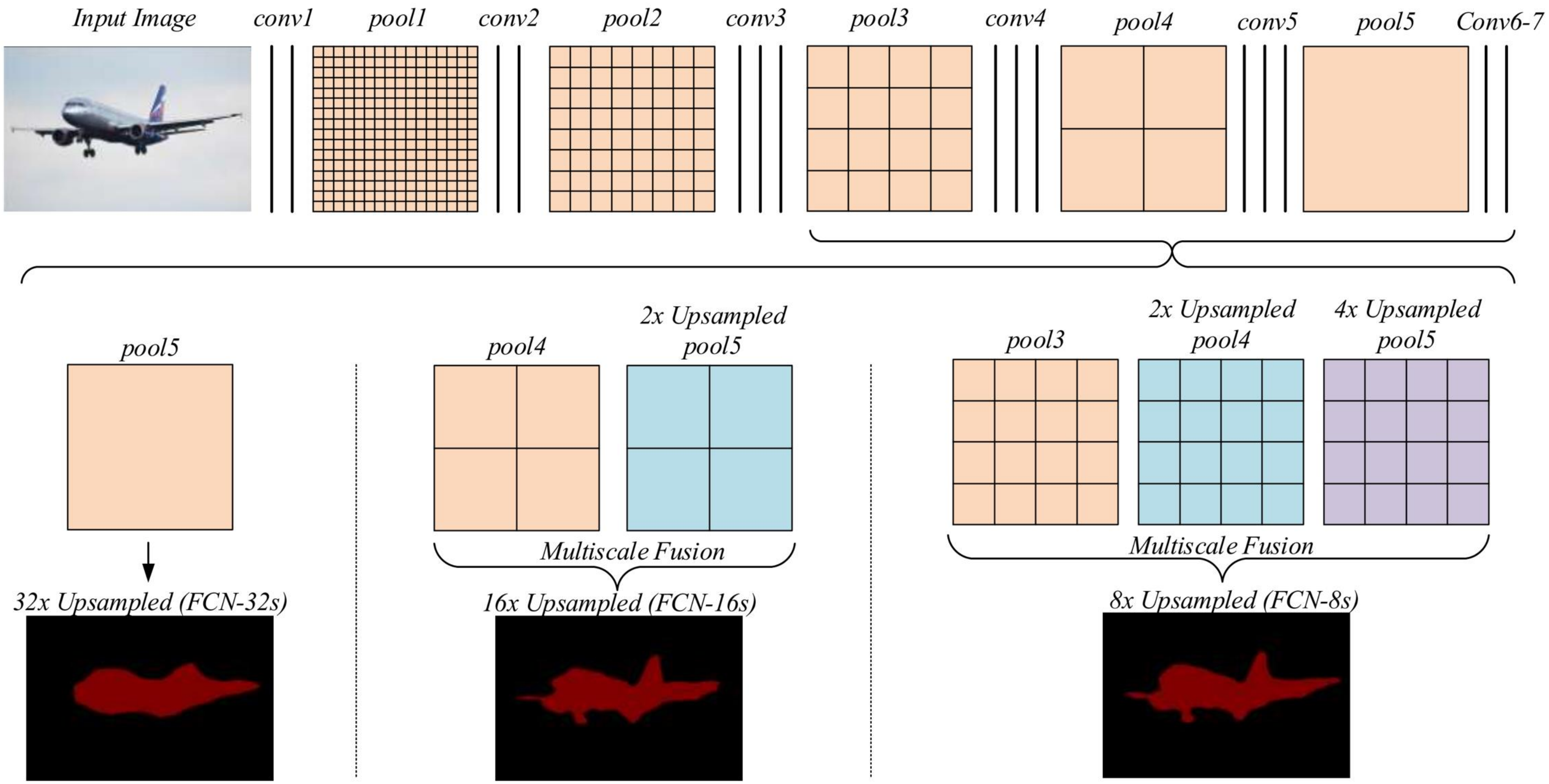
Fully Convolutional Networks (FCN) for 2D segmentation

g.t.: ground truth



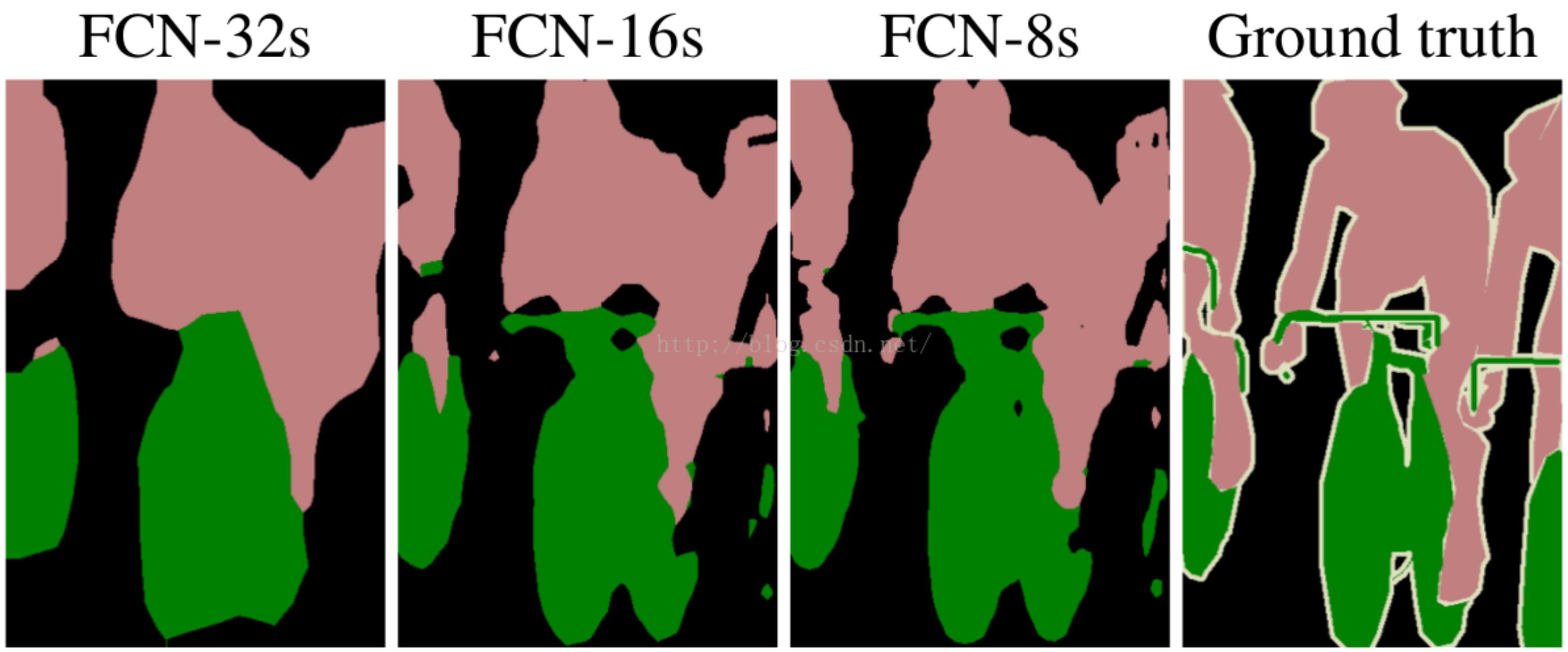
http://www.deeplearning.net/tutorial/fcn_2D_segm.html

Combine several upsampled feature maps



Upsampling is explained in next slides.

FCN results (from FCN paper)



Evan Shelhamer, Jonathan Long, Trevor Darrell, Fully Convolutional Networks for Semantic Segmentation, CVPR 2015

Multinomial Cross-Entropy Loss

- We train a FCN from a set of images and their corresponding ground truth segmentations, $\{(x_i, y_i)\}_{i=1,\ldots,N}$
- The FCN's prediction of y is denoted y^*.
- A segmentation of a color image $x \in \mathbb{R}^{H\times W\times 3}$ assigns the p-th pixel x_p in x a vector $y_p = \left(y_p^1, y_p^2, \ldots, y_p^R\right) \in \{0,1\}^R$, where y_p^r indicates whether pixel x_p belongs to region r, and R is the number of region labels.
- The total loss is

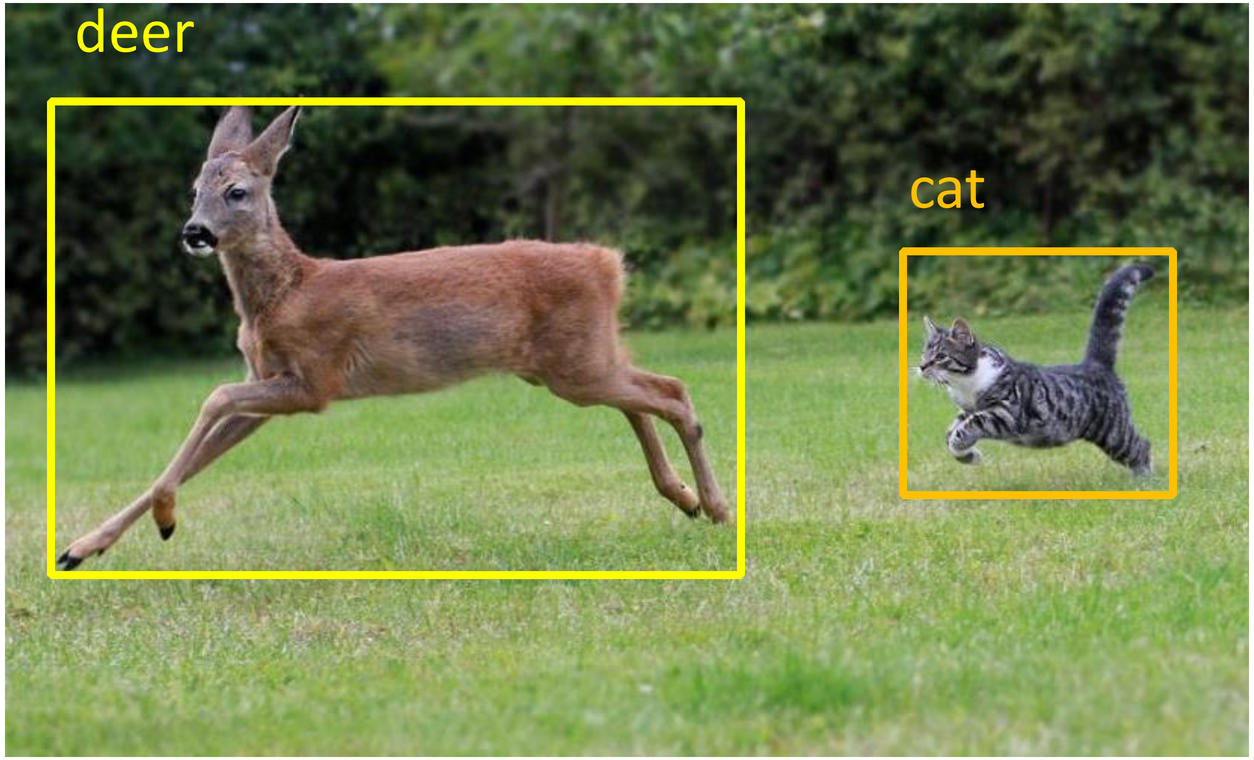
$$L_{FCN}(x) = \sum_{p=1}^{H\times W\times 3} \sum_{r=1}^{R} -y_p^r \log P\left(y_p^r = 1 \middle| x_p\right)$$

$$P\left(y_p^r = 1 \middle| x_p\right) = \frac{\exp\left(a_r(x_p)\right)}{\sum_{k=1}^{R} \exp\left(a_k(x_p)\right)}$$

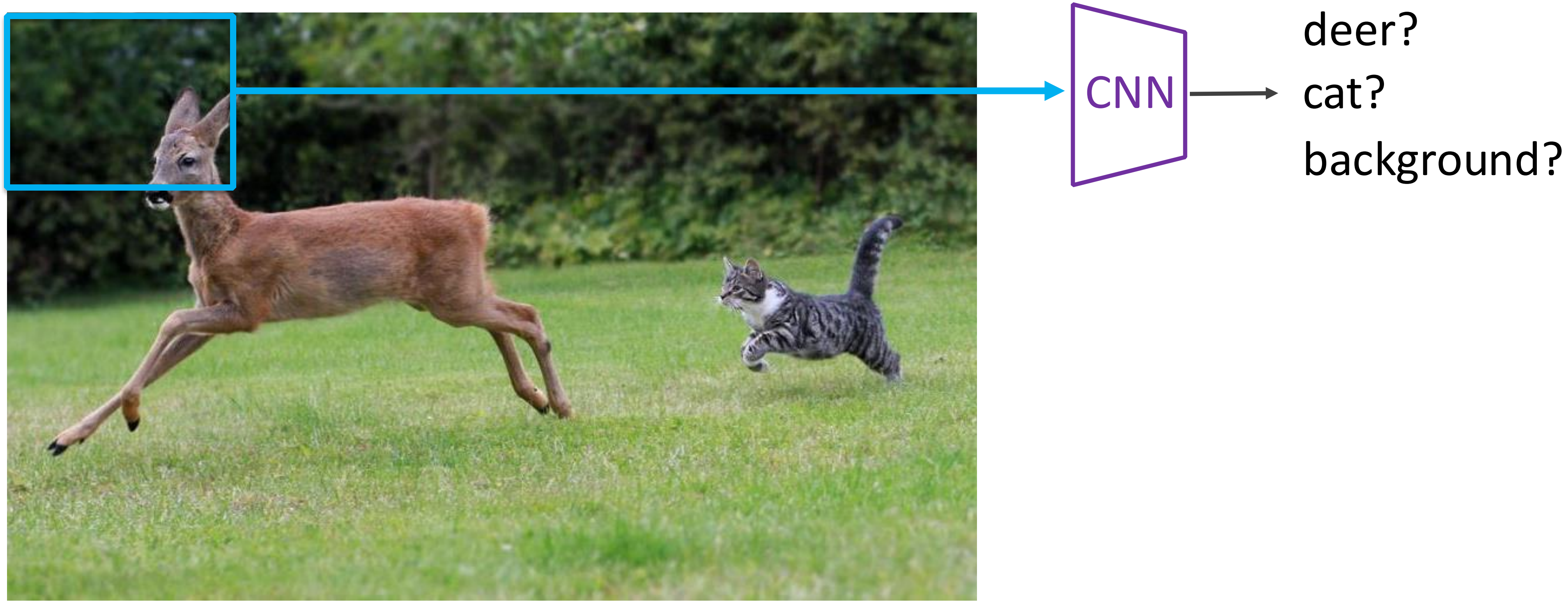
where L_{FCN} is the multinomial cross-entropy loss, and P are the class probabilities output of the softmax function of the FCN, which is based on $a_r(x_p)$, the output activation for region r and pixel p

Object Detection

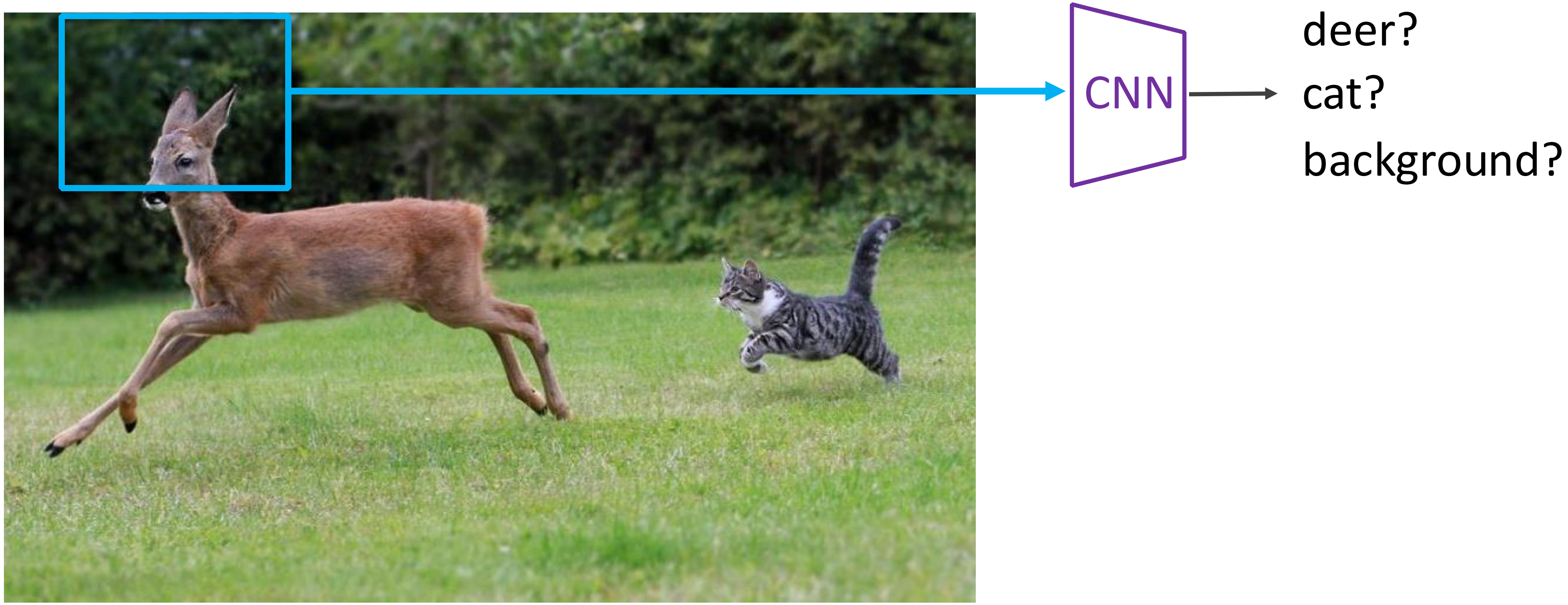
Object Detection



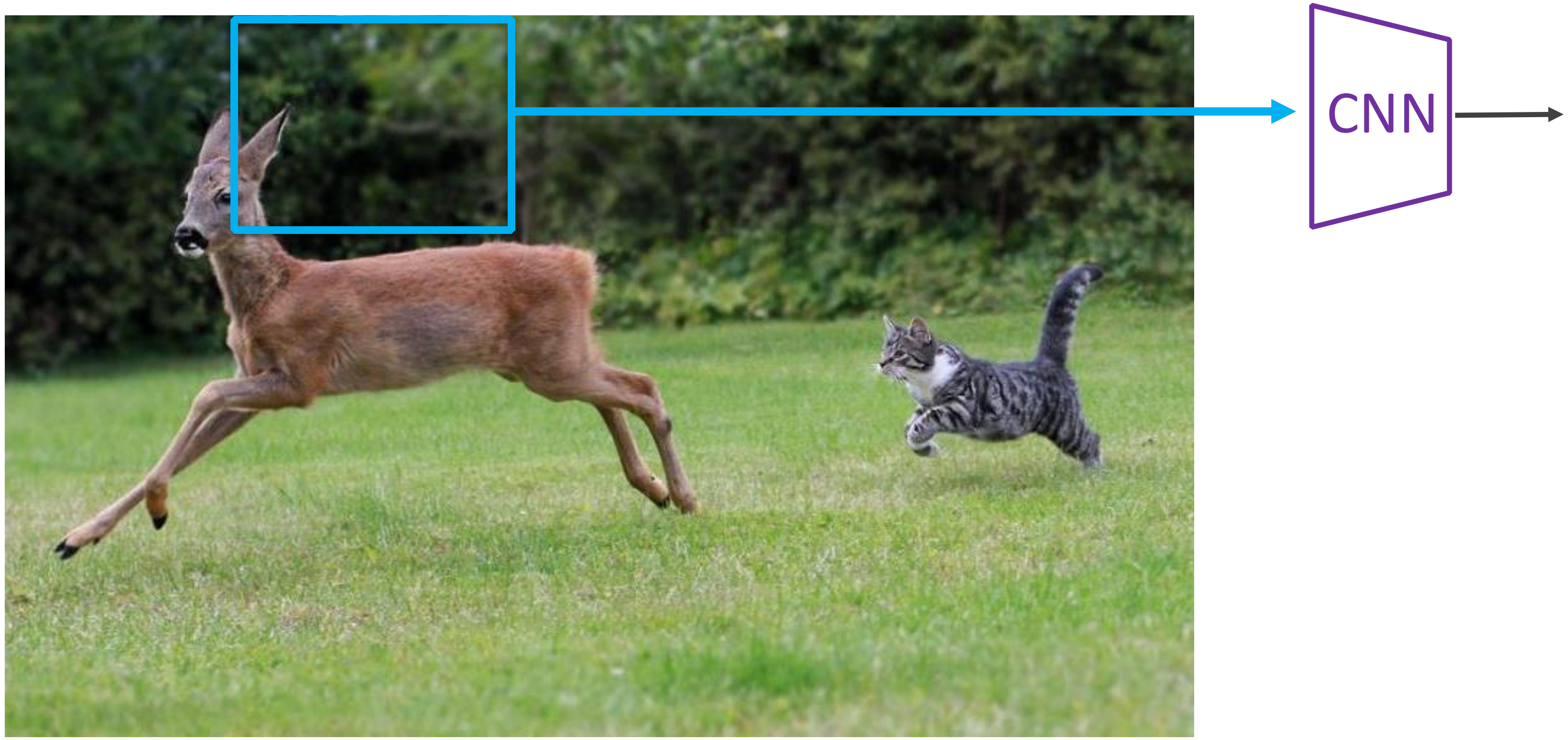
Object Detection as Classification



Object Detection as Classification

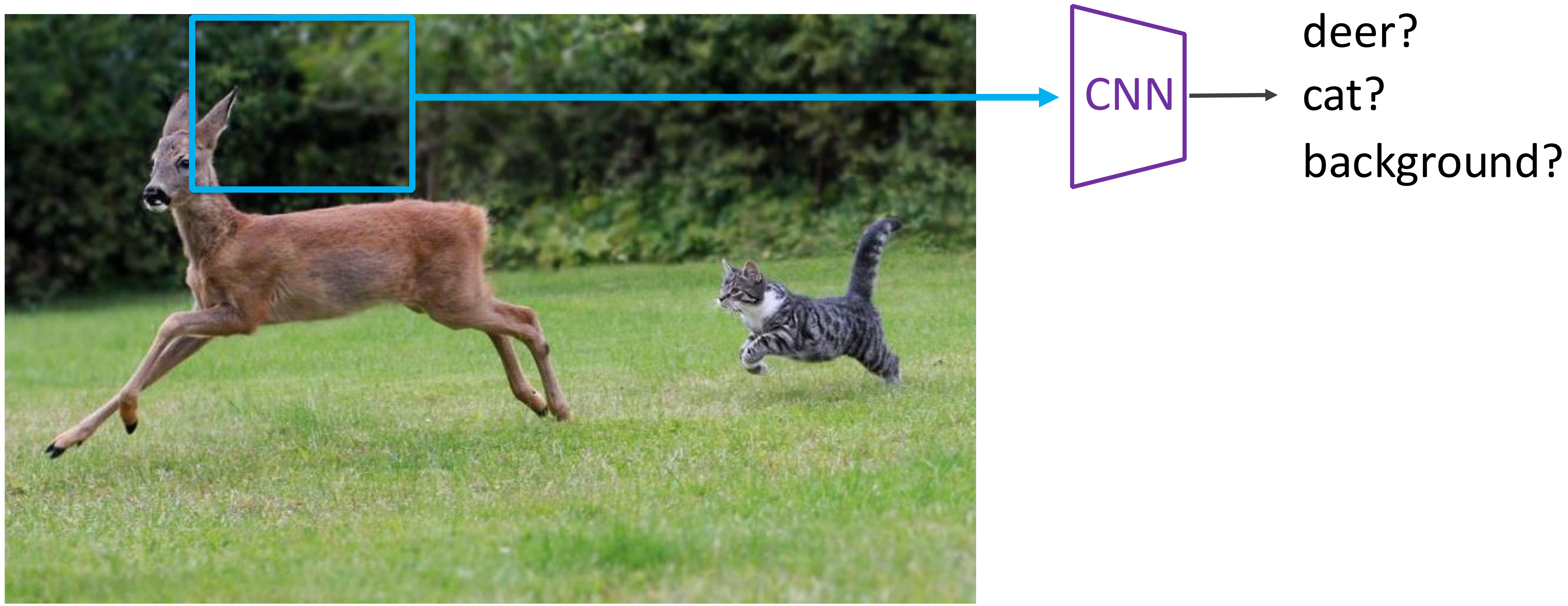


Object Detection as Classification

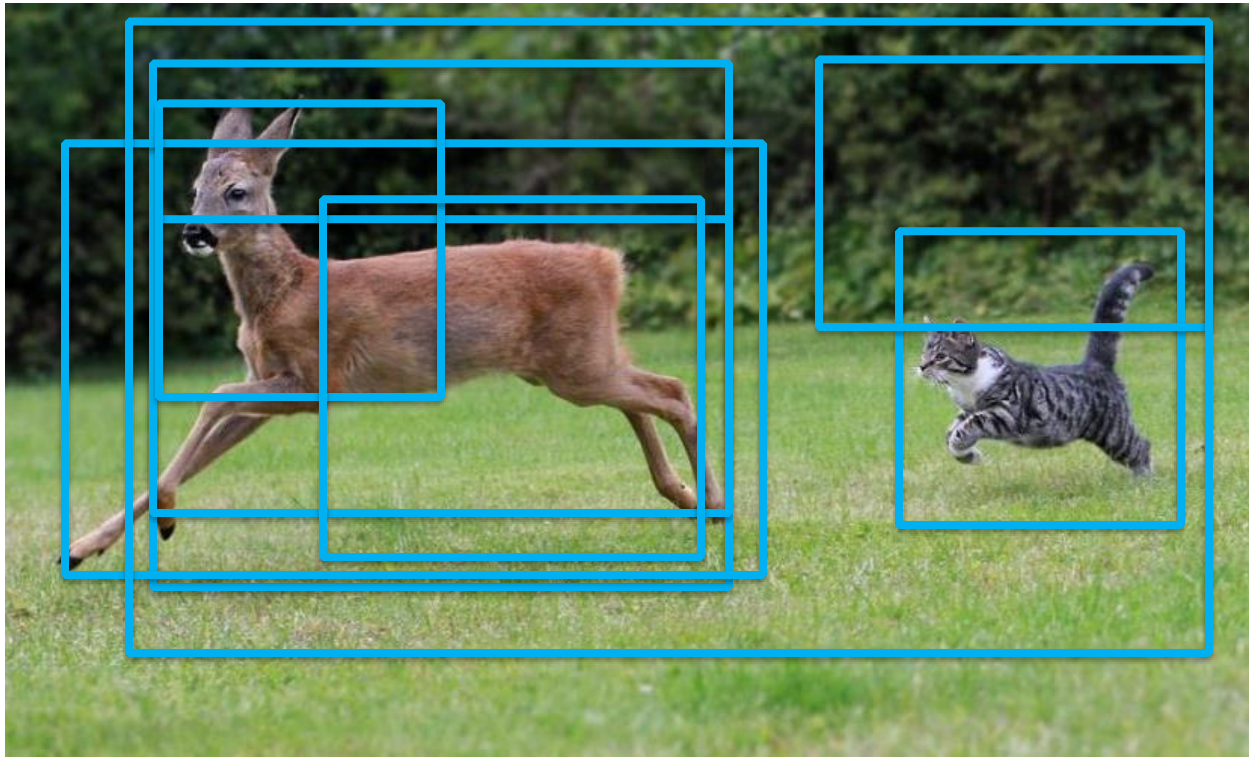


deer?
cat?
background?

Object Detection as Classification with Sliding Window



Object Detection as Classification with Box Proposals



Object Detection: Single Object (Classification + Localization)

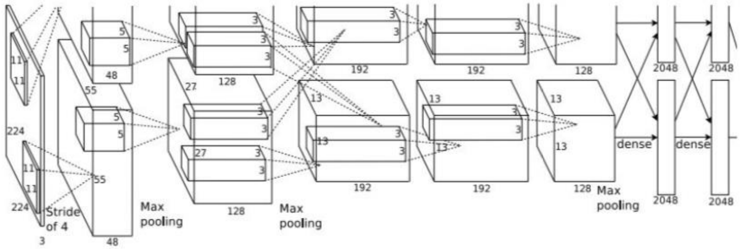
Correct label:
Cat

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

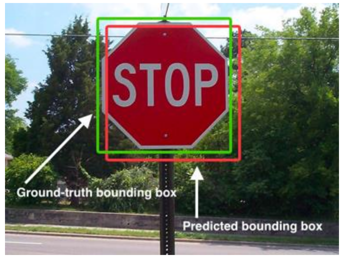
Fully Connected:
4096 to 1000

Softmax Loss

Loss



This image is CC0 public domain



Vector:
4096

Fully Connected:
4096 to 4

Box Coordinates
(x, y, w, h)

L2 Loss

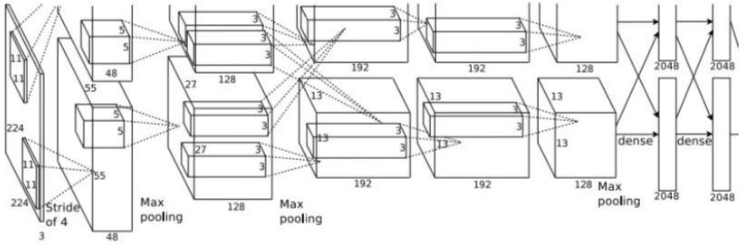
(x, y)

h

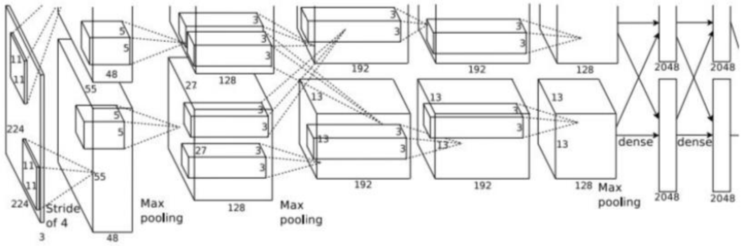
w

Correct box:
(x^*, y^*, w^*, h^*)

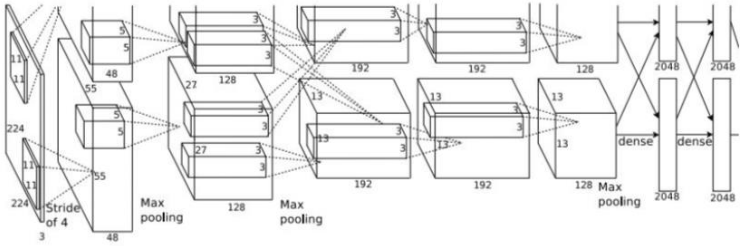
Object Detection: Multiple Objects



CAT: (x, y, w, h)



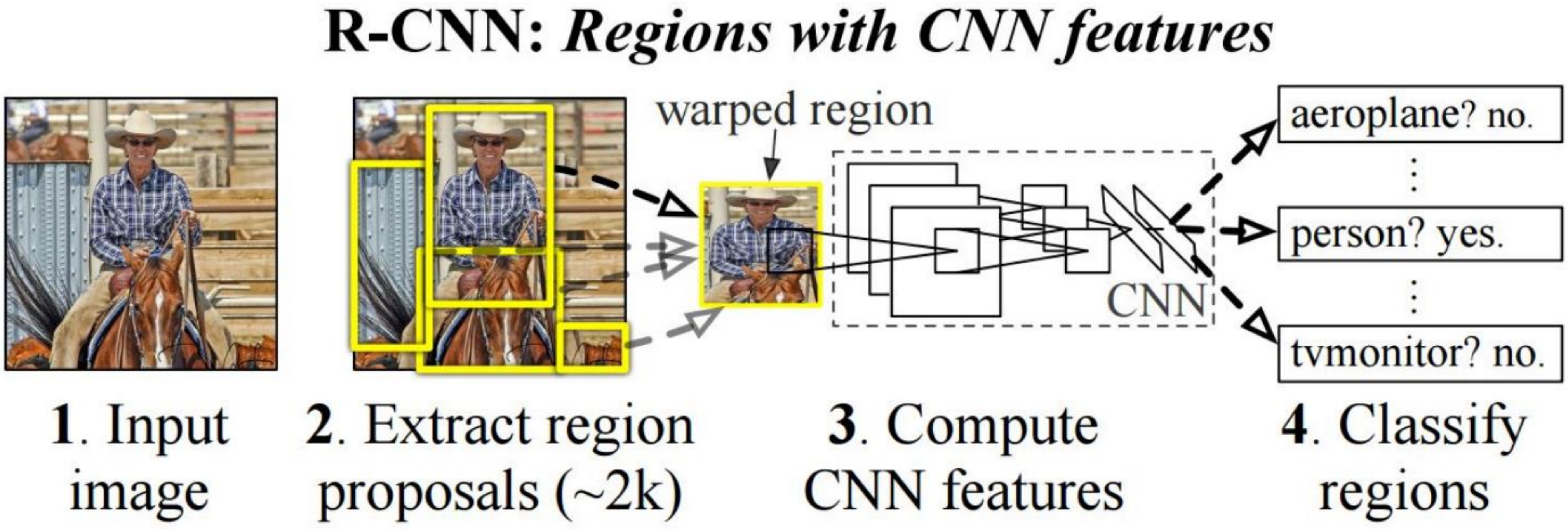
DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)



DUCK: (x, y, w, h)
DUCK : (x, y, w, h)
…

R-CNN

RCNN

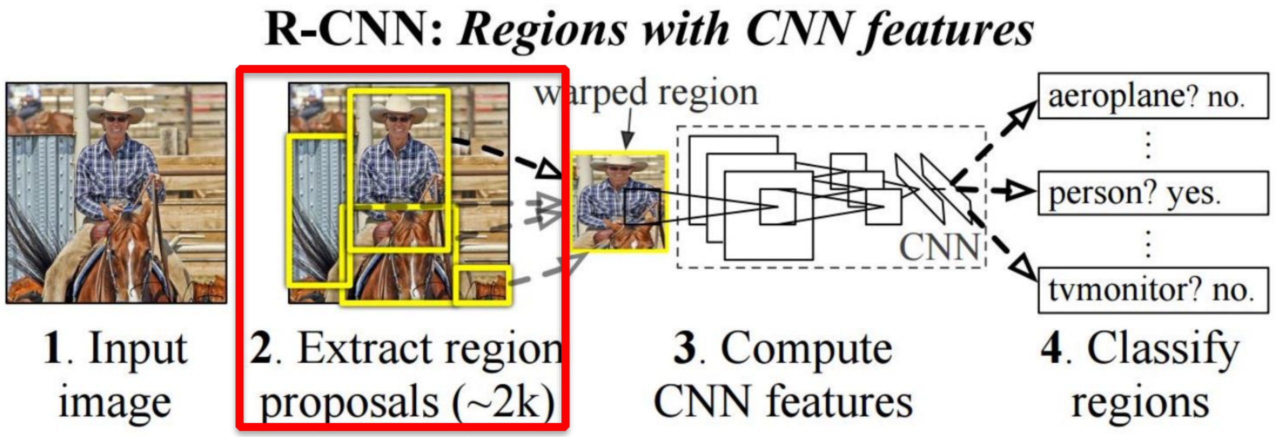


http://www.rossgirshick.info
Rich feature hierarchies for accurate object detection and semantic segmentation.
Girshick et al. CVPR 2014.

RCNN

First stage: generate category-independent region proposals.

- 2000 Region proposals for every image



Selective Search: combine the strength of both an exhaustive search and segmentation. Uijlings et al. IJCV 2013.

Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. *IJCV*, 2013.

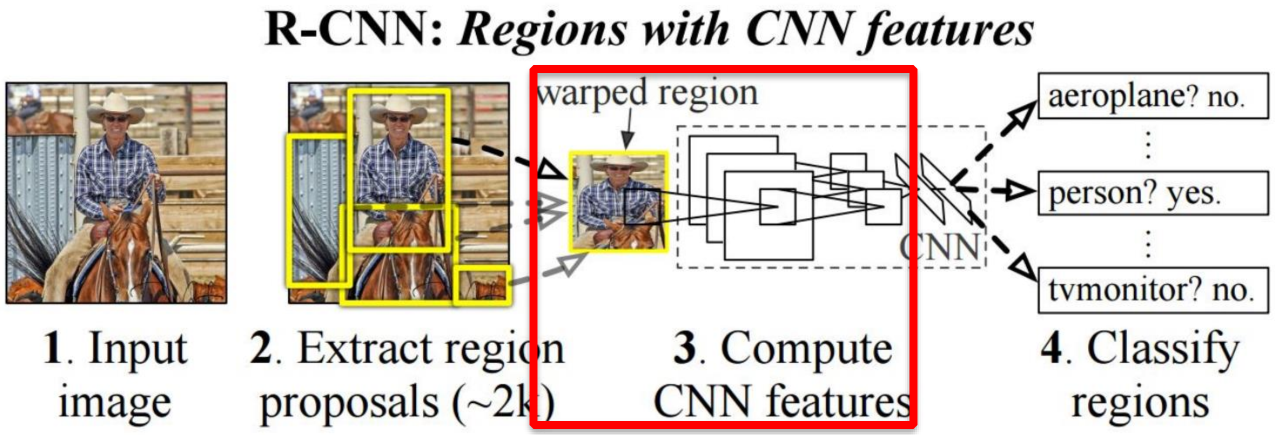
RCNN

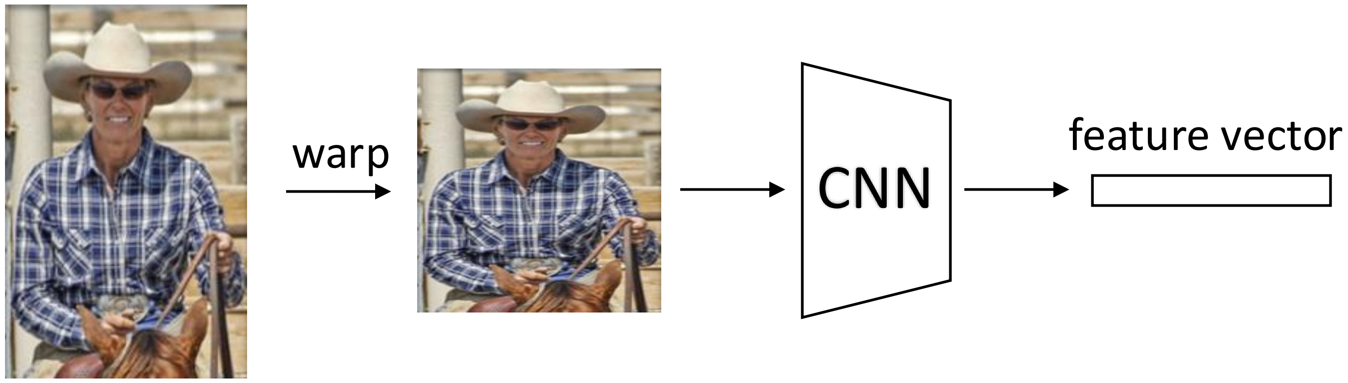
First stage: generate category-independent region proposals.

- 2000 Region proposals for every image

Second stage: extracts a fixed-length feature vector from each region.

- a 4096-dimensional feature vector from each region proposal

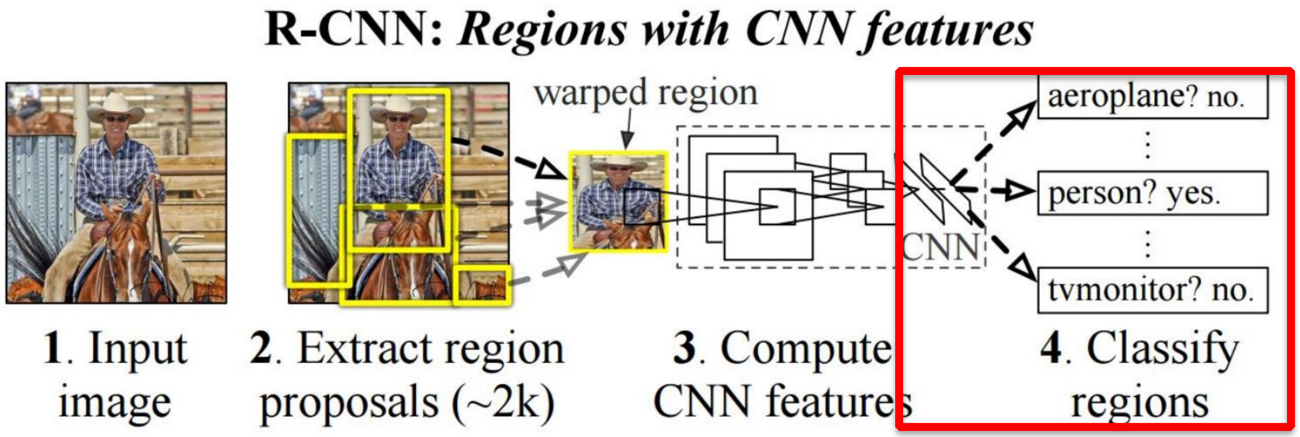




Arbitrary rectangles?
A fixed size input? 227 x 227

5 conv layers + 2 fully connected layers

RCNN



<u>First stage</u>: generate category-independent region proposals.

- 2000 Region proposals for every image

<u>Second stage</u>: extracts a fixed-length feature vector from each region.

- a 4096-dimensional feature vector from each region proposal

<u>Third stage</u>: a set of class- specific linear SVMs and a ridge regression

- SVM: object category
- Ridge regression: location

feature vector → linear svm → people? horse? background?

proposal location

feature vector, proposal location → Bounding box regression → x y w h

How to train?

- Training datasets: $\left\{\left(I^j, \left(g^{*,j}, c^{*,j}\right)\right)\right\}_{j=1,\ldots,N}$
  - I^j: jth image
  - $g^{*,j}$: coordinates $\left(x^{*,j}, y^{*,j}, w^{*,j}, h^{*,j}\right)$ of the ground-truth bounding box for image I^j
  - $c^{*,j}$: label of the object in the bounding box $g^{*,j}$

- Note: if an image contains several boxes, we repeat the image in the training set for each box.

- K is the number of object classes, plus 1 for background,
  - $K = 20$ for PASCAL VOC 2011 detection dataset
  - $K = 200$ for ILSVRC2013 detection dataset

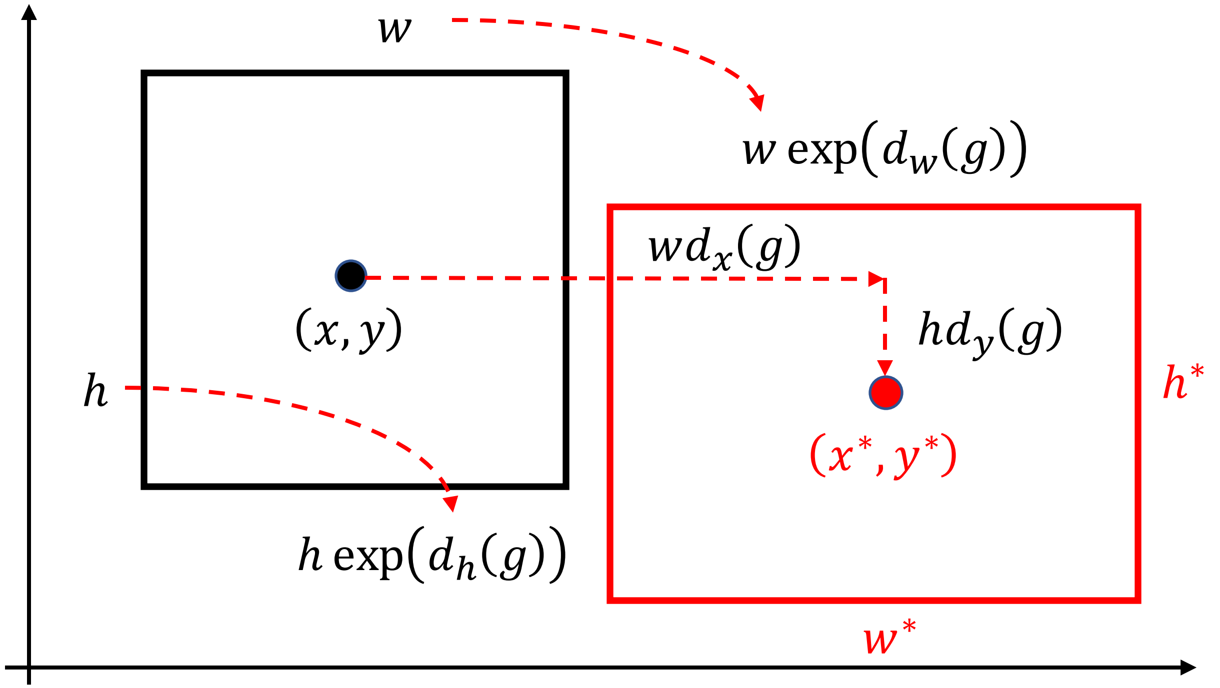
Bounding Box Regression

- Given a proposal bounding box coordinate (given by the region proposal stage)
$$g = (x, y, w, h)$$
as (center coordinates, width, height)

- Its corresponding ground truth box coordinates
$$g^* = (x^*, y^*, w^*, h^*)$$

- The regressor is configured to learn scale-invariant transformation between two centers and log-scale transformation between widths and heights:
$$x^* = w\, d_x(g) + x$$
$$y^* = h\, d_y(g) + y$$
$$w^* = w \exp\big(d_w(g)\big)$$
$$h^* = h \exp\big(d_h(g)\big)$$

where $d_i(g)$ are some corrections to learn.



Reminder:

$$x^* = w\, d_x(g) + x$$
$$y^* = h\, d_y(g) + y$$
$$w^* = w \exp\big(d_w(g)\big)$$
$$h^* = h \exp\big(d_h(g)\big)$$

Regression Targets

- The regression functions $d_i(g)$ must be learned.

- Ideally, for all training pairs $\left(g^j, g^{*,j}\right)$ of the training set, we should have

$$d_x\left(g^j\right) \approx \frac{x^{*,j} - x^j}{w^j} = t_x^{*,j}$$

$$d_y\left(g^j\right) \approx \frac{y^{*,j} - y^j}{h^j} = t_y^{*,j}$$

$$d_w\left(g^j\right) \approx \log\left(\frac{w^{*,j}}{w^j}\right) = t_w^{*,j}$$

$$d_h\left(g^j\right) \approx \log\left(\frac{h^{*,j}}{h^j}\right) = t_h^{*,j}$$

- Hence, the values $t_x^{*,j}$, $t_y^{*,j}$, $t_w^{*,j}$ and $t_h^{*,j}$ are the target values to approximate (see next slide).

- An obvious benefit of applying such transformation is that all the bounding box correction functions, $d_i(g)$ where $i \in \{x, y, w, h\}$ for any input image, can take any value between $(-\infty, +\infty)$

- The goal is to learn the correction functions $d_i(g)$ as a function of g

Linear Regression

- Let us denote $\phi(I^j)$ the feature vector used by the bounding box regression when analyzing the image I^j of the training set (it is the output of the last layer of the CNN).

- It is assumed that $\phi(I^j) = \phi(I^j, g^j)$ depends on g^j

- We assume a linear model for $d_i(g^j)$

$$d_i(g^j) = w_i^T \phi(I^j, g^j)$$

where w_i is a vector of learnable model parameters for each $i \in \{x, y, w, h\}$ of the box parameters.

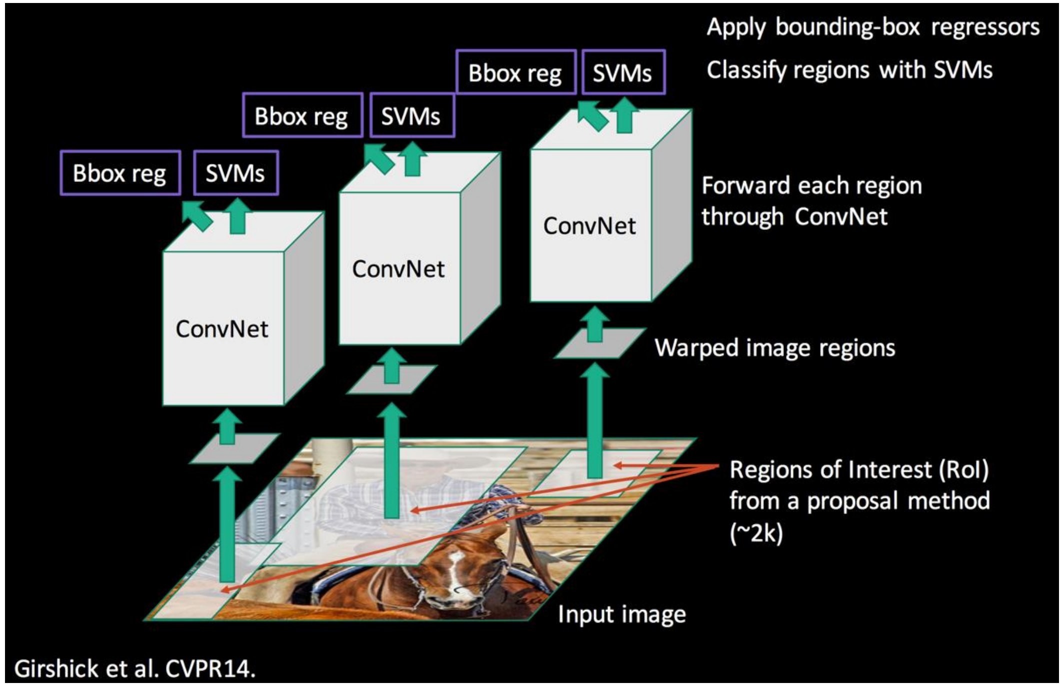
- We learn w_i by optimizing the regularized least squares objective (ridge regression):

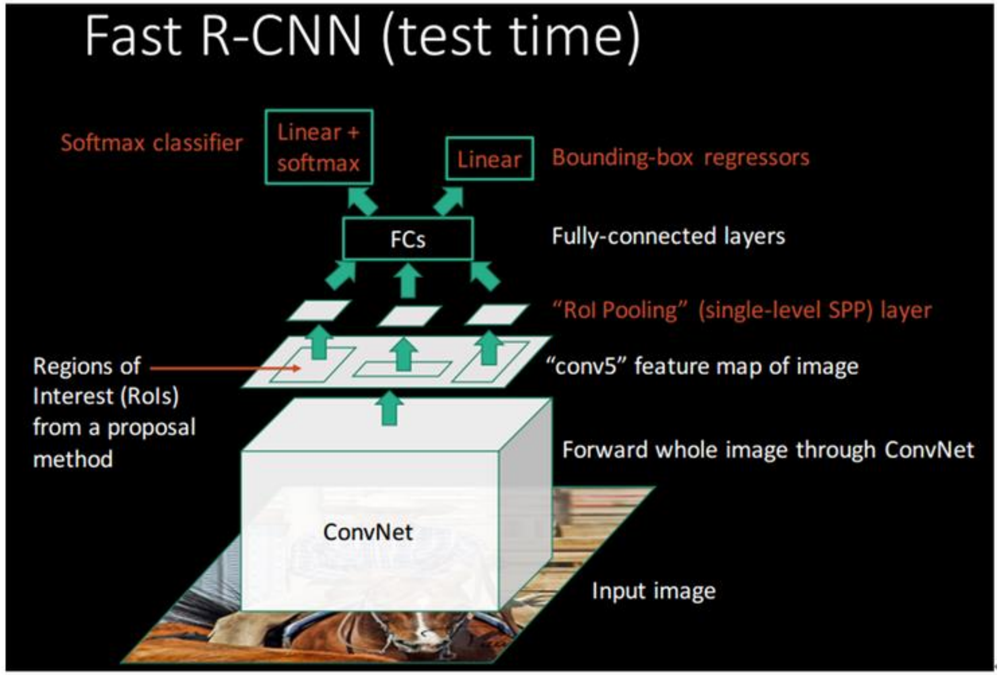
$$\mathcal{L}_{box}(w_i) = \sum_{j=1}^{N} \left(t_i^{*,j} - w_i^T \phi(I^j, g^j)\right)^2 + \lambda \|w_i\|_2^2$$

Fast RCNN

Main idea: reduce the number of computations

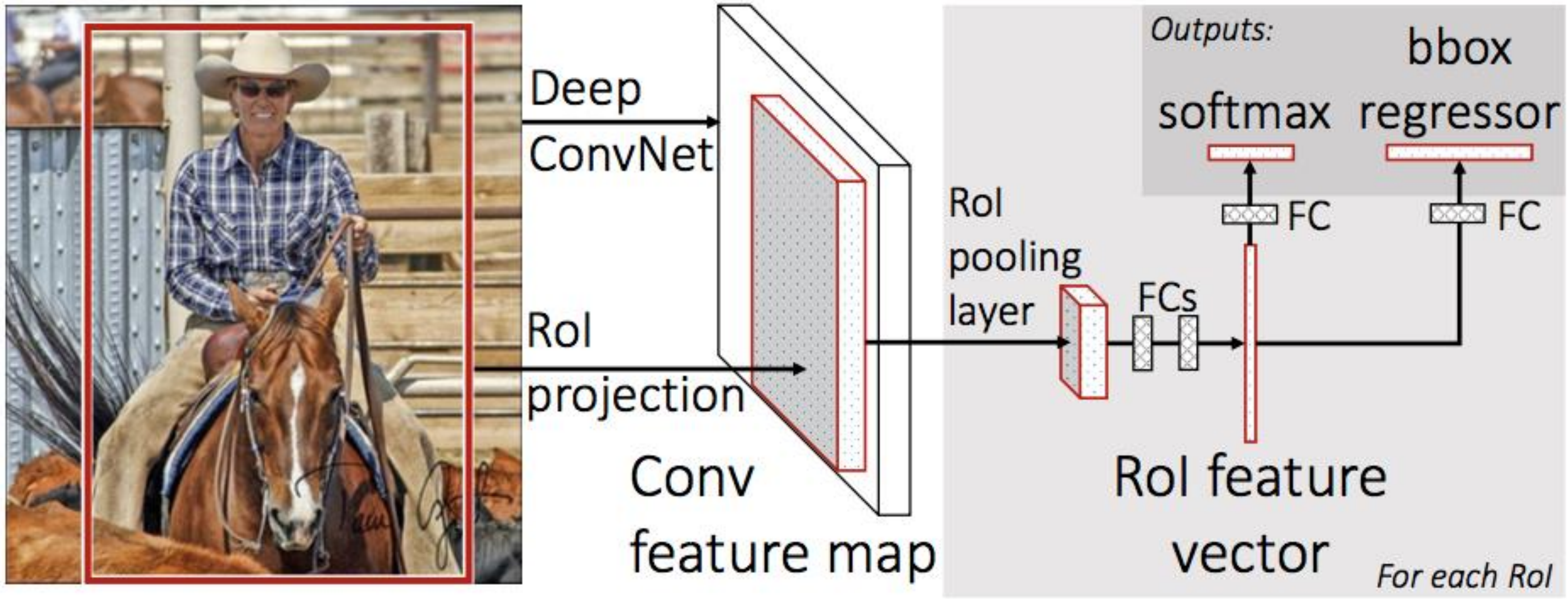
- Computation sharing speeds up R-CNN
  - Instead of extracting CNN feature vectors independently for each region proposal, this model aggregates them into one CNN forward pass over the entire image and the region proposals share this feature matrix.
  - Then the same feature matrix is branched out to be used for learning the object classifier and the bounding-box regressor for all the region proposals.





https://www.saagie.com/blog/object-detection-part2/

Fast-RCNN

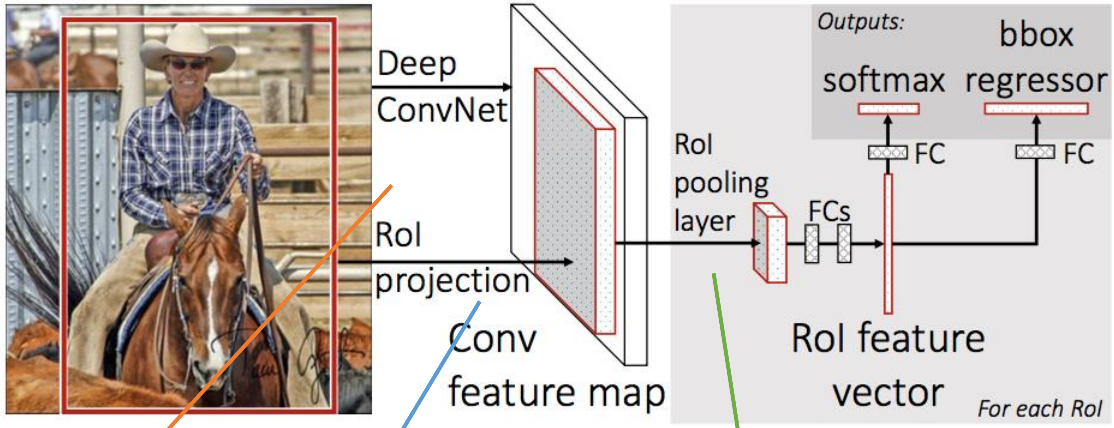


Idea: No need to recompute features for every box independently,
Regress refined bounding box coordinates.

Fast R-CNN. Girshick. ICCV 2015. https://arxiv.org/abs/1504.08083

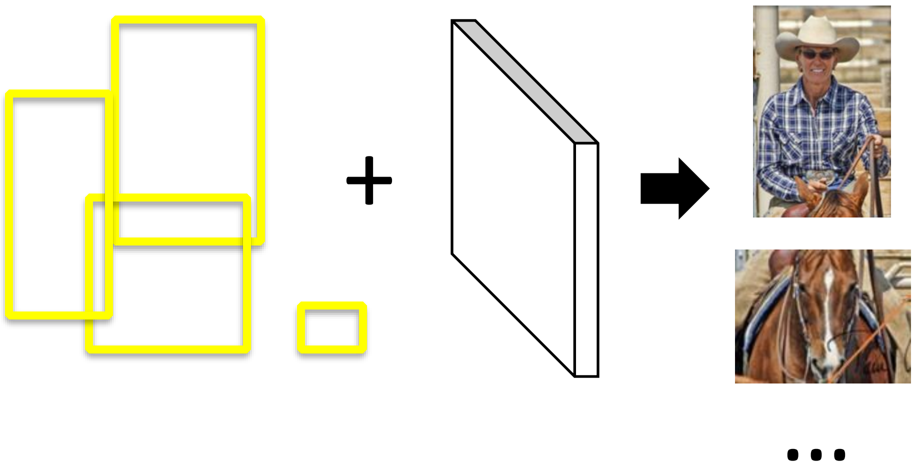
https://github.com/sunshineatnoon/Paper-Collection/blob/master/Fast-RCNN.md

Fast-RCNN

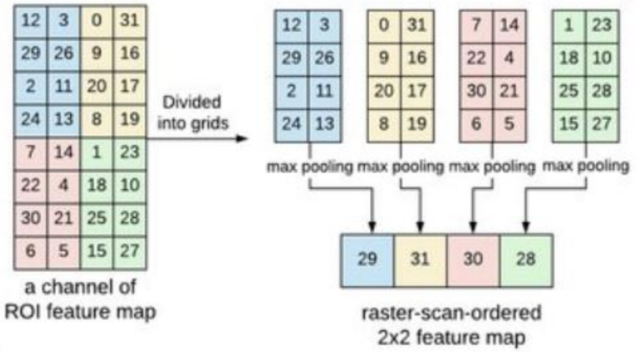


Process the whole image with several convolutional (*conv*) and max pooling layers to produce a conv feature map.

a region of interest (*RoI*) pooling layer extracts a fixed-length feature vector from the region feature map.



feature vector



FC+
softmax

$K + 1$ categories

FC+
regressor

four real-valued numbers for each of the K object classes.

Multi-task loss

- A Fast R-CNN network has two sibling output layers:
  - First: a discrete probability distribution (per RoI), $p = (p_0, \dots, p_K)$, over $K + 1$ categories.
  - Second: bounding-box regression offsets, $t^k = \left(t_x^k, t_y^k, t_w^k, t_h^k\right)$, for each of the K object classes, indexed by k.

- Fast-RCNN uses the RCNN parameterization for t^k: a scale-invariant translation and log-space height/width shift relative to an object proposal.

Multi-task loss to minimize during training

- Each training RoI is labeled with a ground-truth class c (background: $c = 0$) and a ground-truth bounding-box regression target $t^* = \left(t_x^*, t_y^*, t_w^*, t_h^*\right)$.
- We use a multi-task loss L on each labeled RoI to jointly train for classification and bounding-box regression:

$$L(p, c, t^c, t^*) = L_{cls}(p, c) + \lambda 1_{\{c \geq 1\}} L_{loc}(t^c, t^*)$$

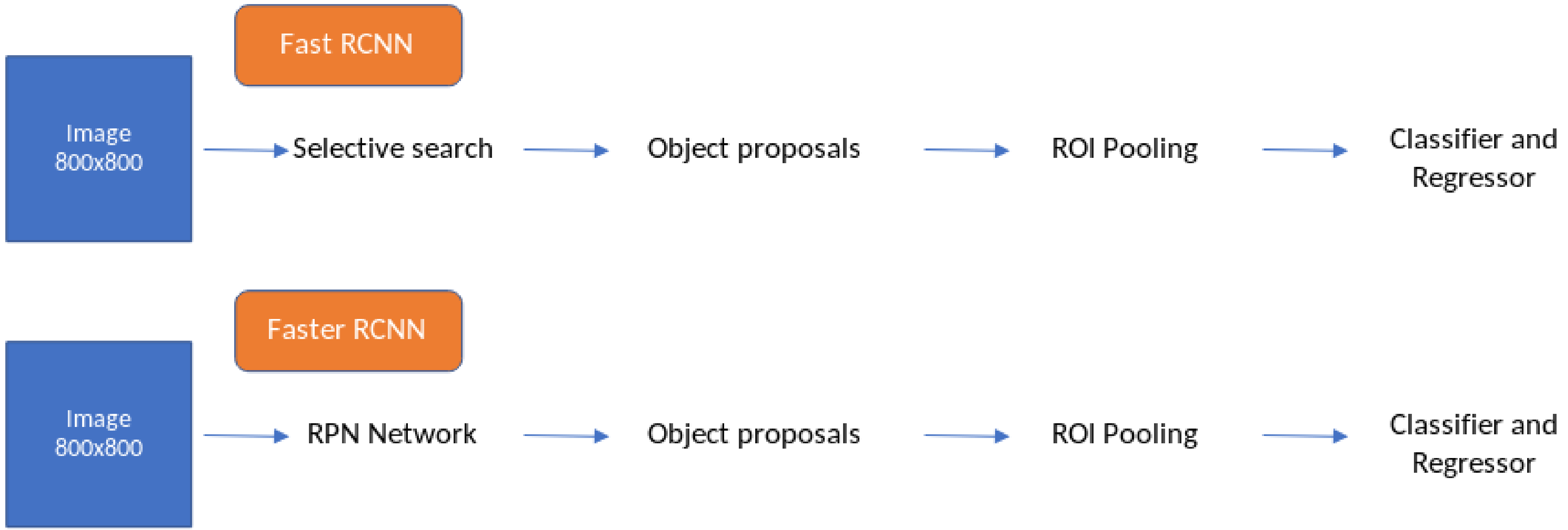
  - $L_{cls}(p, c) = -\log p_c$: log loss for true class c
  - Second: L_{loc} is defined over a tuple of bounding-box regression offsets $t^c = \left(t_x^c, t_y^c, t_w^c, t_h^c\right)$ for class c and the target t^*

$$L_{loc}(t^c, t^*) = \sum_{i \in \{x,y,w,h\}} \mathrm{smooth}_{L_1}(t_i^c - t_i^*)$$

$$\mathrm{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

Faster RCNN

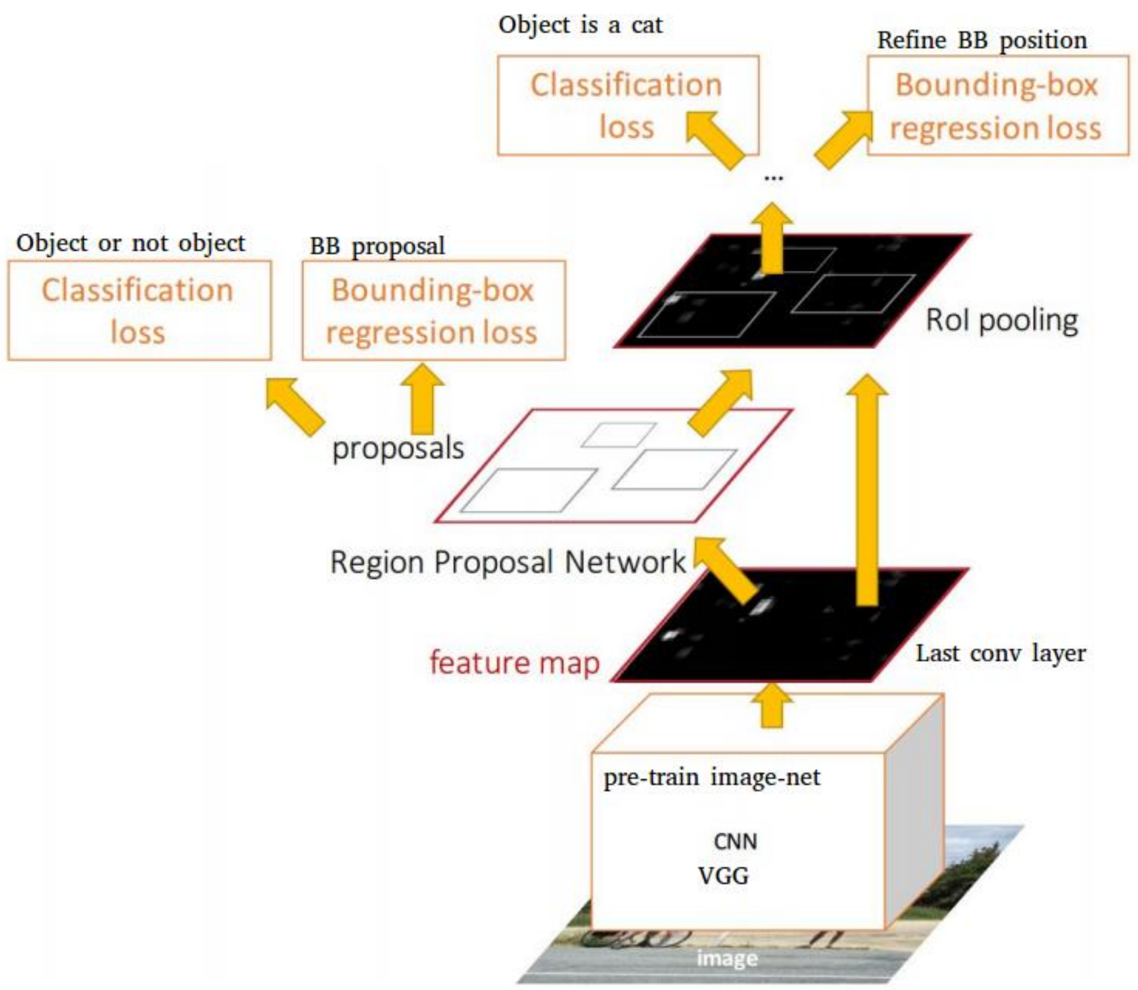
Fast RCNN vs Faster RCNN in one picture



Faster-RCNN

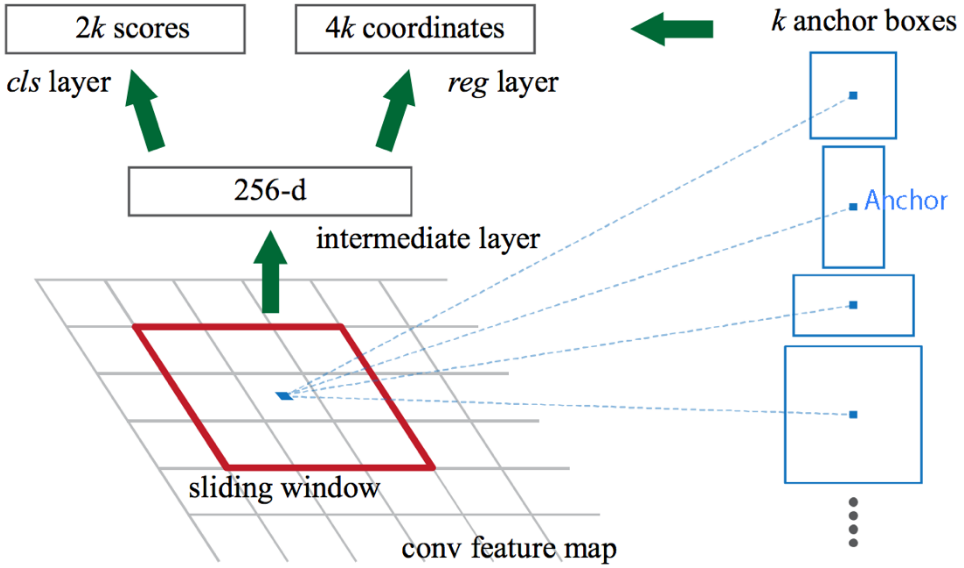
- Idea: Integrate the Bounding Box Proposals as part of the CNN predictions

- Original paper (Ren et al. NIPS 2015): https://arxiv.org/abs/1506.01497



Multiple anchor boxes

- To account for varying sizes of anchors, a set of k bounding-box regressors are learned (one regressor for each anchor).
- Each regressor is responsible for one scale and one aspect ratio, and the k regressors do not share weights.
- As such, it is still possible to predict boxes of various sizes even though the features are of a fixed size/scale, thanks to the design of anchors.



Train the RPN

- The mini-batch is the set $\{(\{p_i\},\{t_i\})\}_{i=1,\ldots,M}$; The outputs of the *cls* and *reg* layers consist of $\{p_i\}$ and $\{t_i\}$ respectively.
- We minimize the objective function

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_{i=1}^{M} L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}}\sum_{i=1}^{M} p_i^* L_{reg}(t_i, t_i^*)$$

- p_i is the predicted probability of anchor i being an object.
- The ground-truth label p_i^* is 1 if the anchor is positive, and is 0 if the anchor is negative
- The classification loss L_{cls} is log loss over two classes (object vs. not object).
- For the regression loss, we use $L_{reg}(t_i, t_i^*) = R(t_i - t_i^*)$ where R is the robust loss function (smooth L1)
- The term $p_i^* L_{reg}$ means the regression loss is activated only for positive anchors ($p_i^* = 1$) and is disabled otherwise ($p_i^* = 0$).
- The two terms are normalized by N_{cls} and N_{reg} and weighted by a balancing parameter λ.

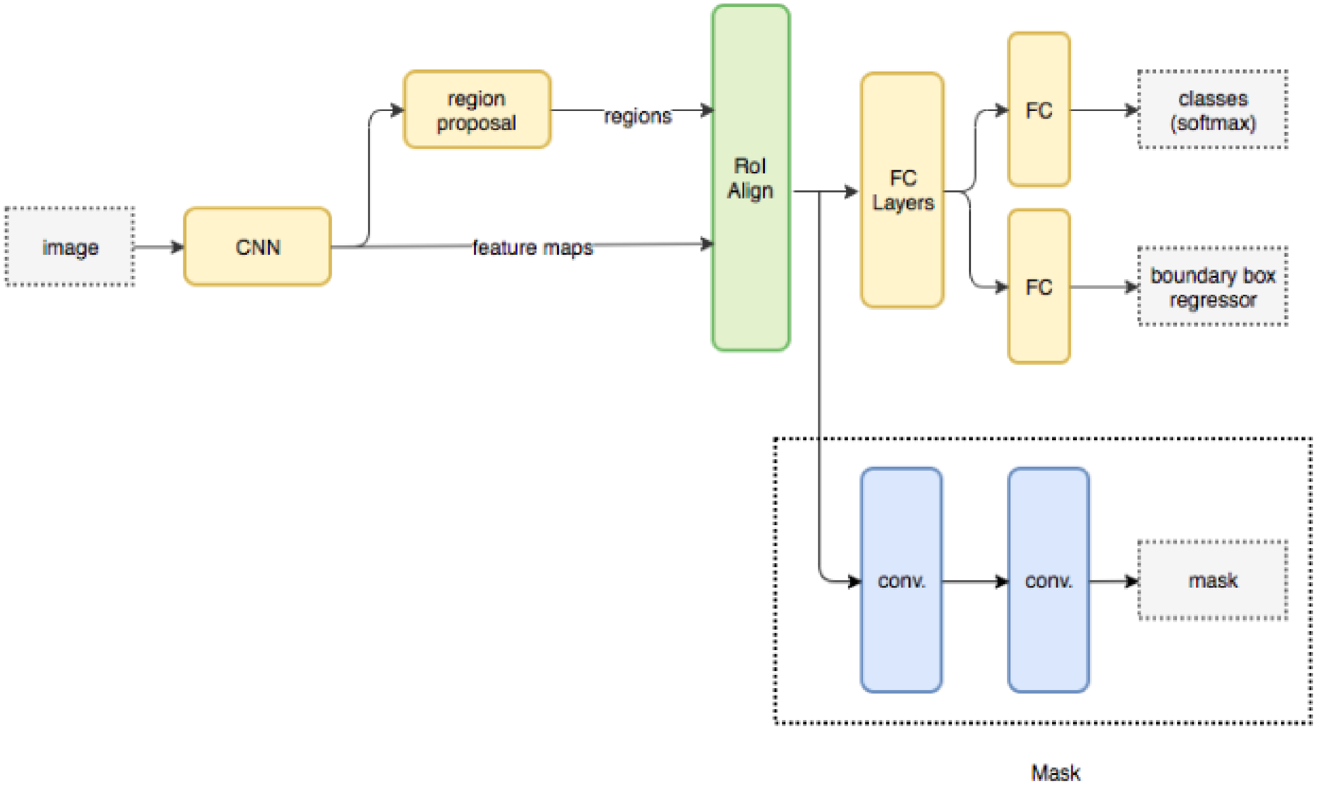
Example

https://jhui.github.io/2017/03/15/Fast-R-CNN-and-Faster-R-CNN/

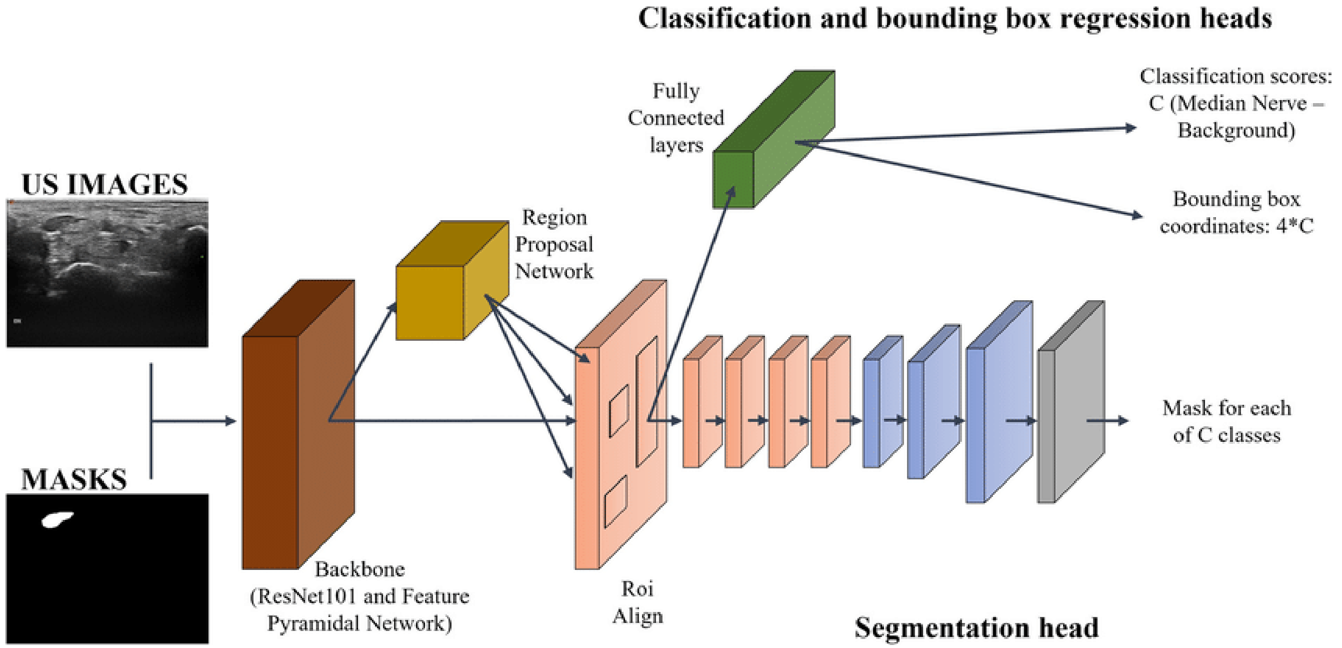
Mask RCNN

Mask R-CNN

- Mask R-CNN (regional convolutional neural network) is a two stage framework:
  - The first stage scans the image and generates *proposals* (areas likely to contain an object).
  - The second stage classifies the proposals and generates bounding boxes and masks.
- Both stages are connected to the backbone structure.



Backbone network



- What is backbone?
  - This is a standard convolutional neural network (typically, ResNet50 or ResNet101) that serves as a feature extractor.
  - The early layers detect low-level features (edges and corners), and later layers successively detect higher level features (car, person, sky).
  - Passing through the backbone network, the image is converted to a feature map. This feature map becomes the input for the following stages.

https://www.researchgate.net/figure/Mask-R-CNN-is-a-CNN-made-of-backbone-composed-by-a-ResNet101-and-a-feature-pyramid_fig1_358458237

Multi-task loss

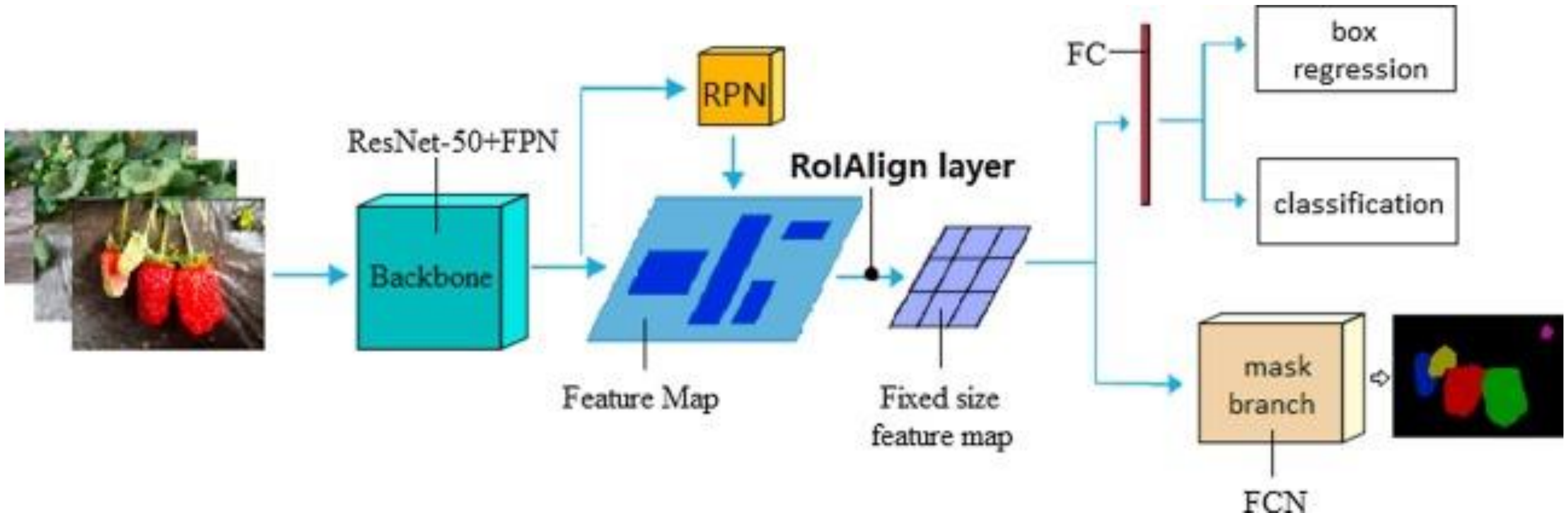
- Formally, during training, we define a multi-task loss on each sampled RoI as

$$L = L_{cls} + L_{box} + L_{mask}$$

- The classification loss L_{cls} and bounding-box loss L_{box} are identical as those defined in Faster-RCNN.
- The mask branch has a Km^2-dimensional output for each RoI, which encodes K binary masks of resolution $m \times m$, one for each of the K classes.
  - To this we apply a per-pixel sigmoid, and define L_{mask} as the average binary cross-entropy loss.
  - For an RoI associated with ground-truth class k, L_{mask} is only defined on the k-th mask (other mask outputs do not contribute to the loss).
- The network generates masks for every class without competition among classes
  - it relies on the dedicated classification branch to predict the class label used to select the output mask.
  - This decouples mask and class prediction.

Mask representation

- A mask encodes an input object's spatial layout.
- Thus, unlike class labels or box offsets that come from short output vectors of fully-connected layers, extracting the spatial structure of masks can be addressed naturally by the pixel-to-pixel correspondence provided by convolutions.
- Specifically, we predict an $m \times m$ mask from each RoI using an FCN (Fully convolutional network). This allows each layer in the mask branch to maintain the explicit $m \times m$ object spatial layout without collapsing it into a vector representation that lacks spatial dimensions.



An Example

mashroom6
person 0.989
person 0.989
laptop 0.980
laptop 0.998
chair 0.746
book 0.894

Conclusion

Conclusion

- Many architectures for object detection
- Different architectures can be combined
- Many improvements each year (YOLO9000, etc.)
- Deep neural networks are still the heart of the architectures
- The criterion to optimize plays a very important role