

Computer Vision

Lecture 10 – Recognition

Prof. Dr.-Ing. Andreas Geiger

Autonomous Vision Group

University of Tübingen / MPI-IS



e l l i s
European Laboratory for Learning and Intelligent Systems

Agenda

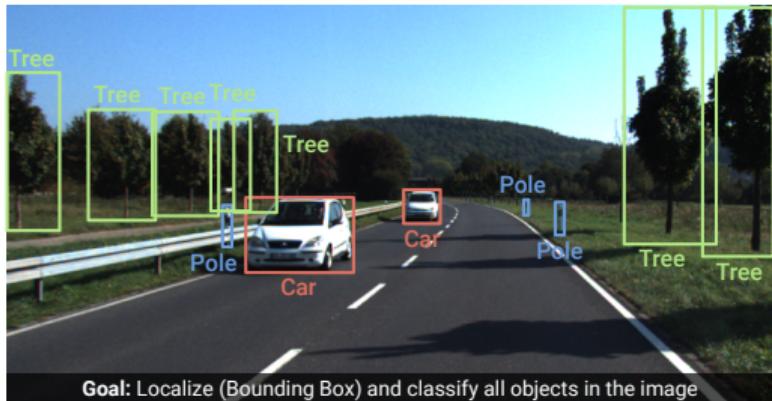
10.1 Image Classification

10.2 Semantic Segmentation

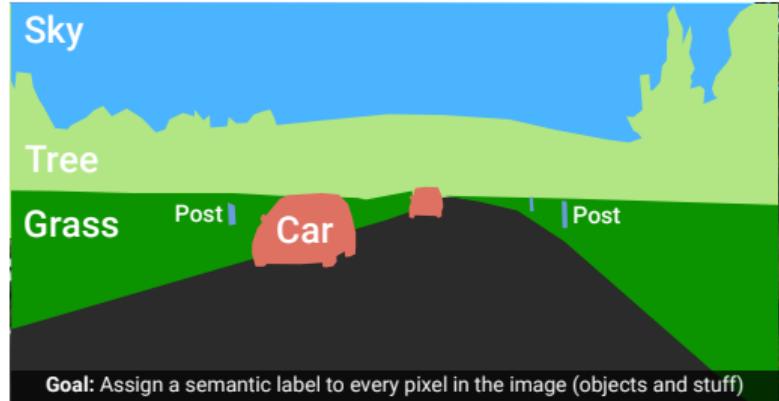
10.3 Object Detection and Segmentation



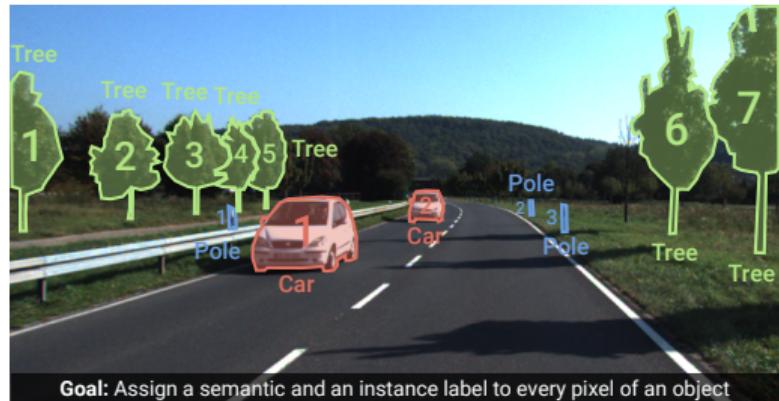
Image Classification



Object Detection



Semantic Segmentation



Instance Segmentation

10.1

Image Classification

Image Classification



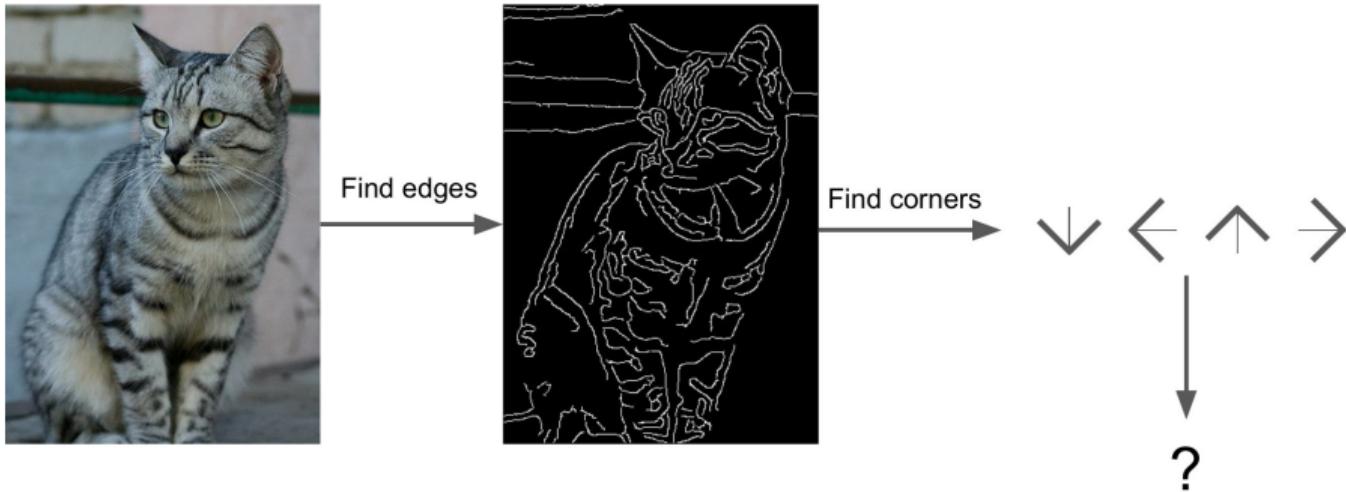
Goal: Assign a single class label (image category) to the image

Image Classification

Why Image Classification?

- ▶ Useful task on its own (e.g., handwritten digits / image search)
- ▶ Simple input/output specification, standard loss functions
- ▶ For long time, one of the difficult gold-standard tasks in computer vision
- ▶ Useful high-level proxy-task to learn good representations
which transfer to new tasks and domains

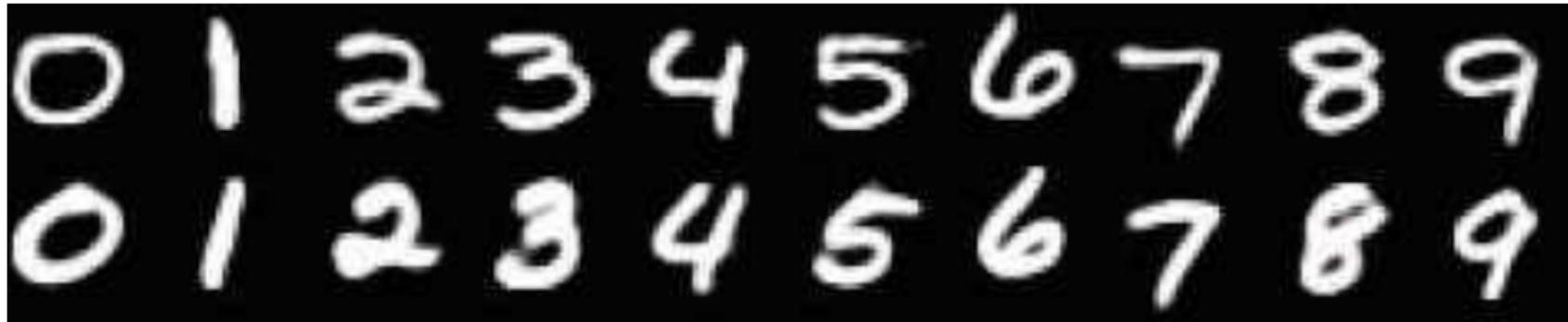
Early Attempts



- ▶ Early attempts of **hand-engineering** object detection algorithms **failed** as object shapes and appearances are simply too hard to describe
- ▶ We need **machine learning** and labeled datasets to learn these relationships!

Datasets

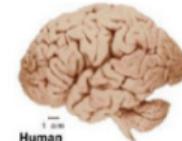
MNIST



MNIST Handwritten Digits:

- ▶ One of the most popular datasets in ML (many variants, still in use today)
- ▶ Based on a data from the National Institute of Standards and Technology
- ▶ Hand written by Census Bureau employees and high-school children
- ▶ Resolution: 28 x 28 pixels, 60k training samples with labels, 10k test samples

Caltech101



Caltech101:

- ▶ Caltech101 was the first major object recognition dataset, collected in 2003
- ▶ 101 object categories
- ▶ Hand-curated from Google Image Search
- ▶ Biased: canonical size and location



ImageNet



IMAGENET

22,000 categories

:

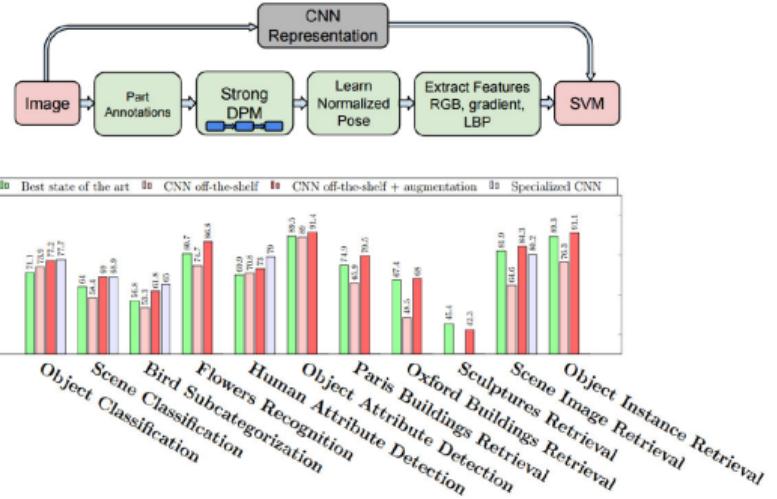
15,000,000 images

- ▶ AlexNet results on ImageNet challenge started the **deep learning revolution**
- ▶ Still one of the primary datasets for **pre-training** generic representations today

ImageNet Pre-Training

Transfer Learning:

- Deep representations generalize well despite large number of parameters
- Pre-train CNN on large amounts of data on generic task (e.g., ImageNet)
- Fine-tune (re-train) only last layers or all layers on little data of a new task leads to state-of-the-art performance
- Standard paradigm today



Challenges

Challenges: Large Number of Image Categories



~10,000 to 30,000

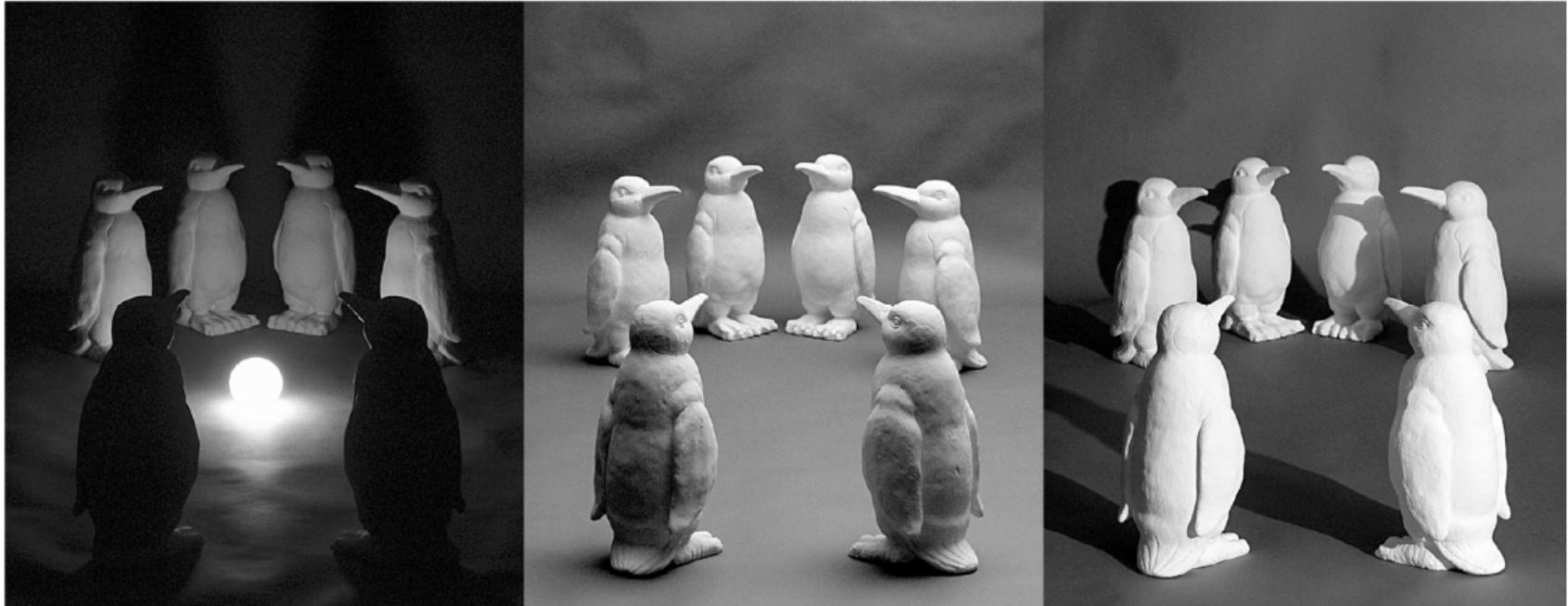
Challenges: Intra-class Variation



Challenges: Viewpoint Variation



Challenges: Illumination Changes



Challenges: Background Clutter



Challenges: Deformation



Challenges: Occlusion



Simple Models

Nearest Neighbor

Nearest Neighbor Classifier:

- ▶ Choose image distance:

$$d(\mathbf{I}_1, \mathbf{I}_2) = \sum_{\mathbf{p}} |I_1(\mathbf{p}) - I_2(\mathbf{p})|_1$$

- ▶ Given \mathbf{I} , find nearest neighbor:

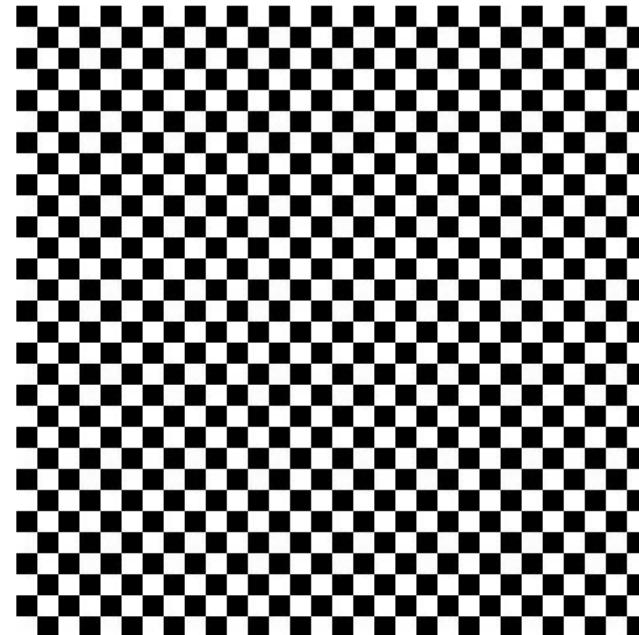
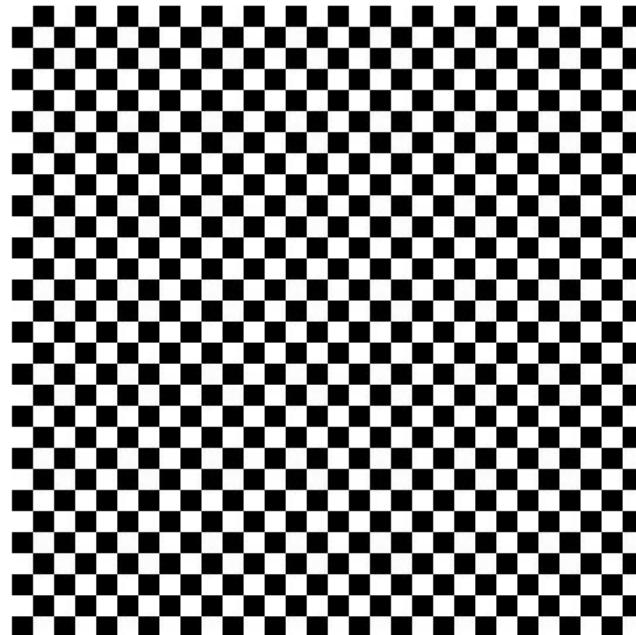
$$\mathbf{I}^* = \operatorname{argmin}_{\mathbf{I}' \in \mathcal{D}} d(\mathbf{I}, \mathbf{I}')$$

- ▶ Return class of \mathbf{I}^*



- ▶ This is a **slow** (NN at test time) and **bad** (pixel distances uninformative) **classifier**

Nearest Neighbor



Thought Experiment:

- ▶ Two checkerboards, horizontally displaced by one field \Rightarrow What is $d(\mathbf{I}_1, \mathbf{I}_2)$?

Bag-of-Words (BoW)



- Idea: Obtain spatial invariance by comparing **histograms of local features**

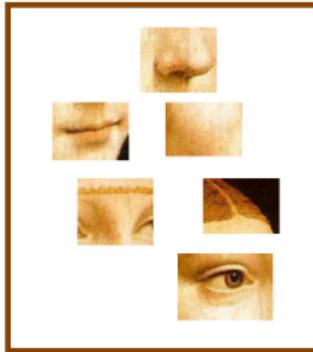
Bag-of-Words (BoW)



- The idea of bag-of-word models originates from natural language processing
- **Orderless** document representation: **frequencies of words** from a dictionary

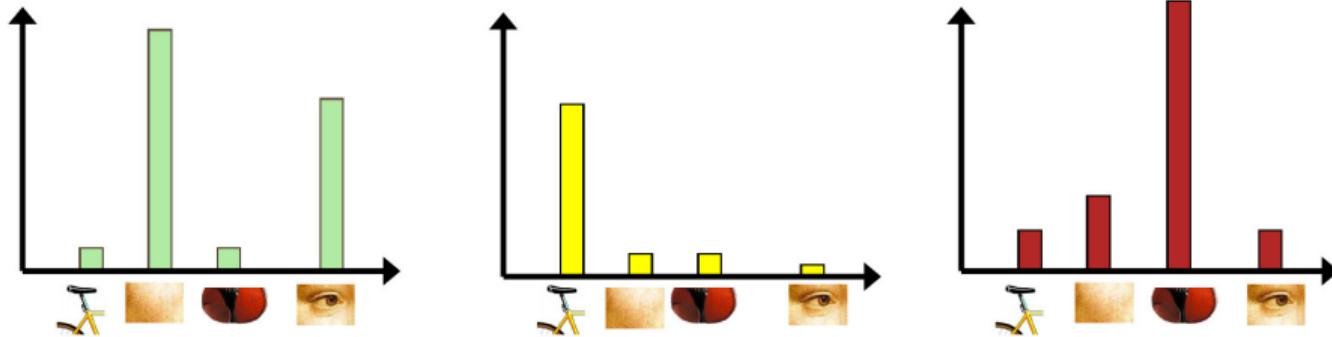
Bag-of-Words (BoW)

1. **Extract features** (e.g., SIFT detector)
2. Learn **visual vocabulary** (e.g., k-means on SIFT feature vectors)
3. **Quantize features** into visual words using vocabulary (nearest neighbors)
4. Represent images by **histograms** of visual word frequencies
5. **Train classifier** (e.g., k-NN, SVM, random forest, neural network)

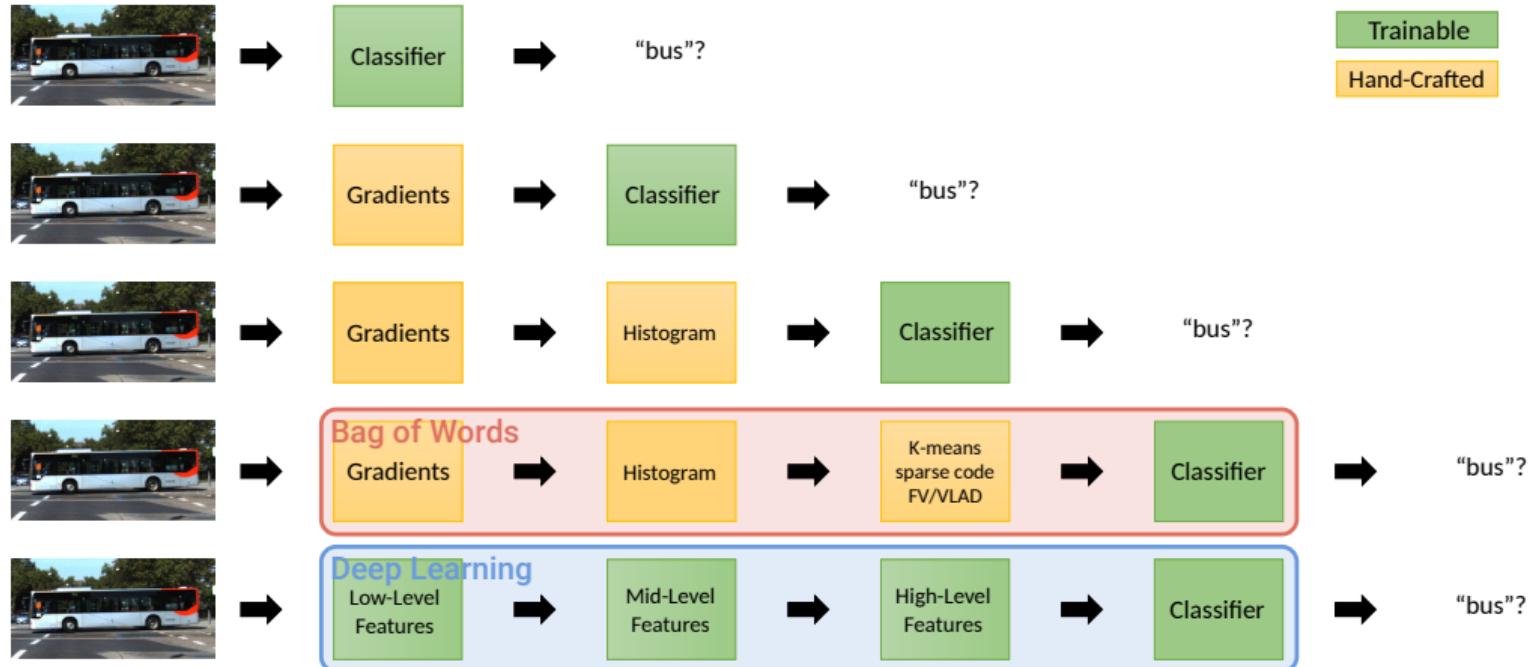


Bag-of-Words (BoW)

1. **Extract features** (e.g., SIFT detector)
2. Learn **visual vocabulary** (e.g., k-means on SIFT feature vectors)
3. **Quantize features** into visual words using vocabulary (nearest neighbors)
4. Represent images by **histograms** of visual word frequencies
5. **Train classifier** (e.g., k-NN, SVM, random forest, neural network)

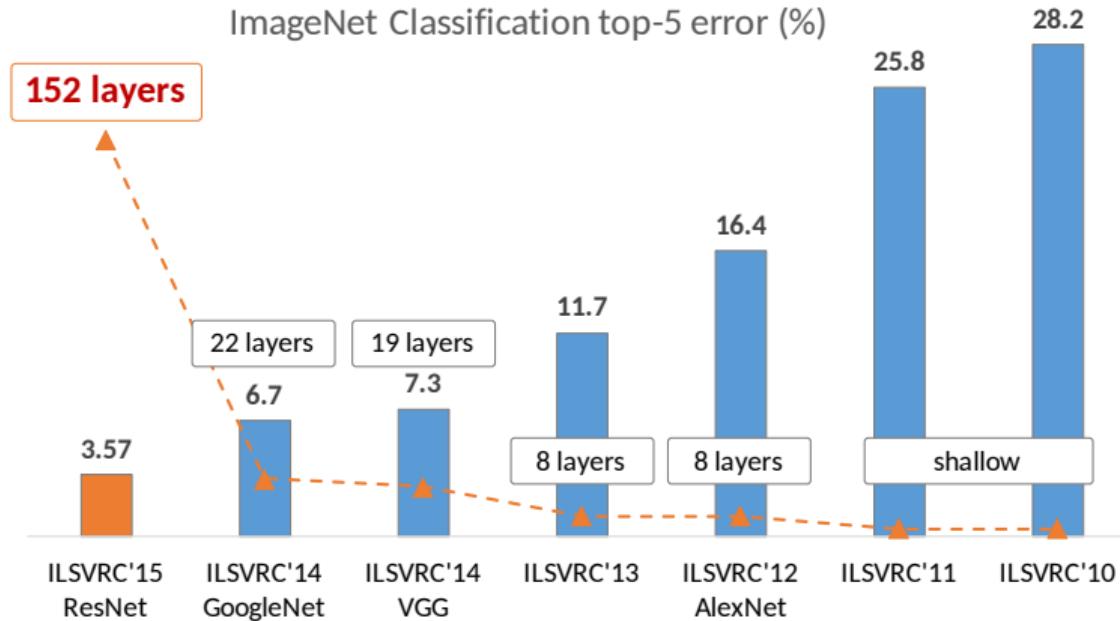


Representation Learning



- ▶ Problem with BoW: Still **many hand-designed components**, little learning
- ▶ Solution: Learn entire model (from image to decision) **end-to-end** from data

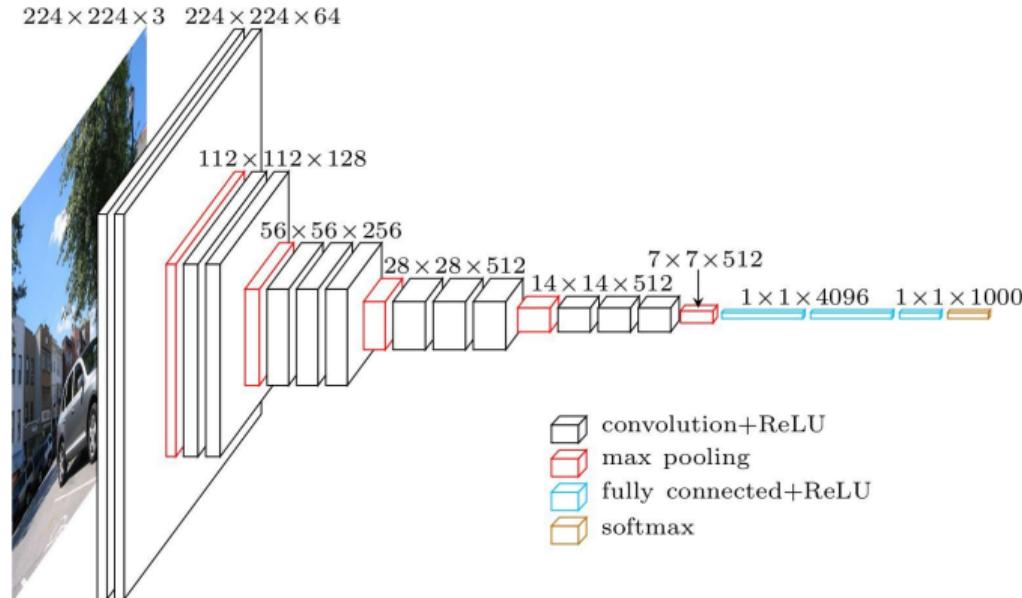
ImageNet Large Scale Visual Recognition Challenge



- ▶ Classification into 1000 object categories. Current state-of-the-art: 1.3 %

Convolutional Neural Networks

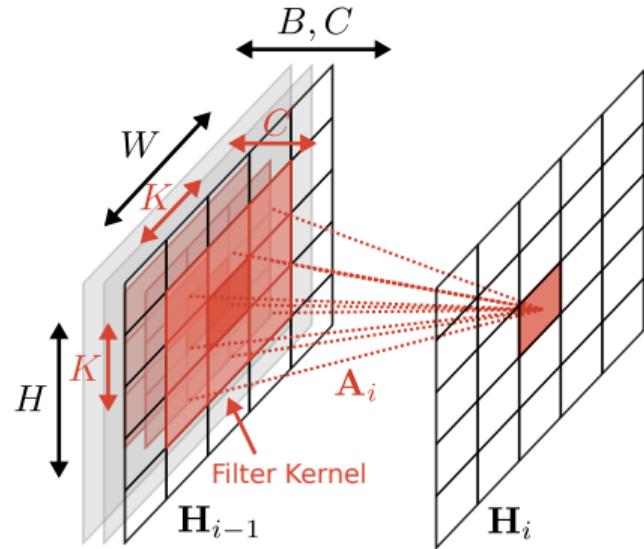
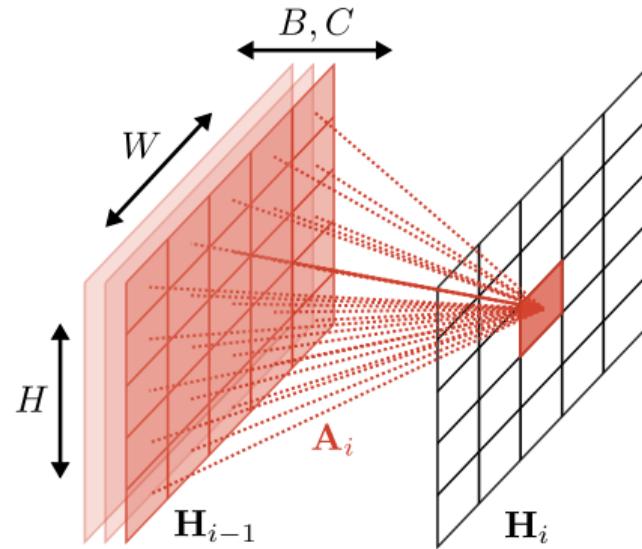
VGG Network



3 Types of Layers:

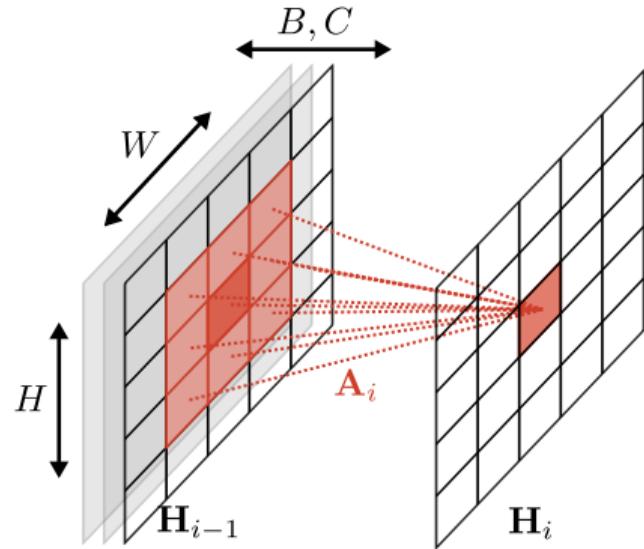
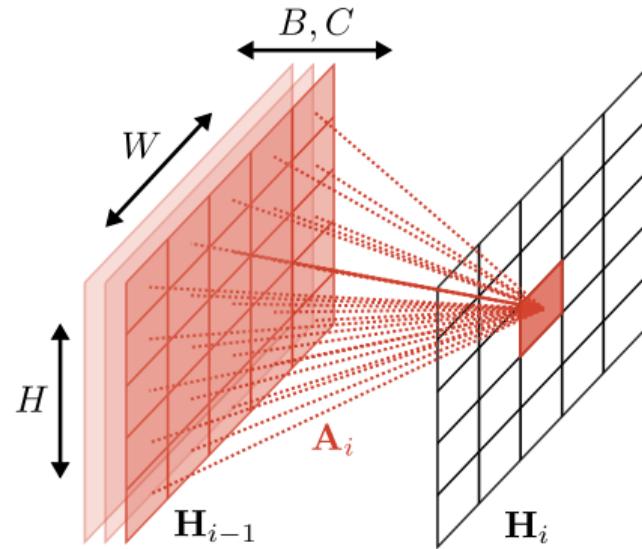
- ▶ Convolution layers, downsampling layers and fully connected layers

Convolutional Layers



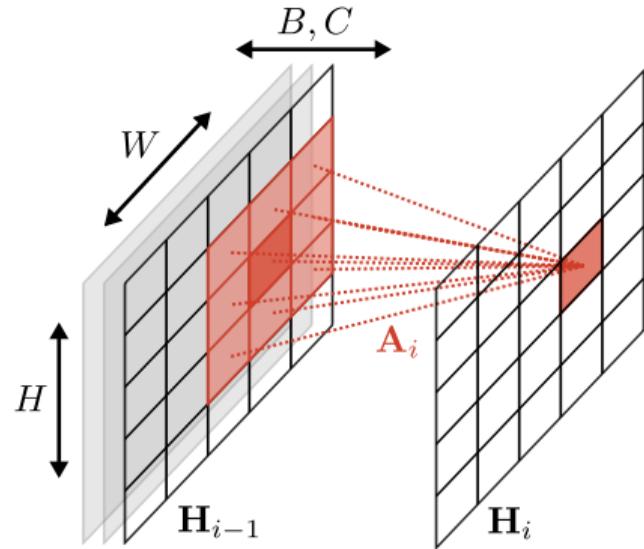
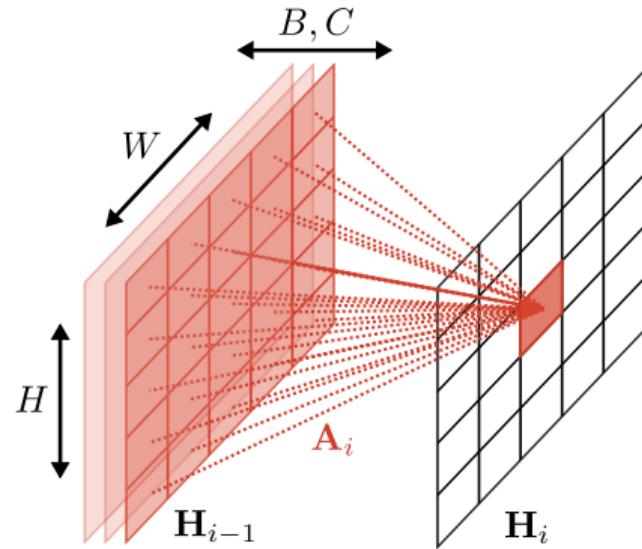
- ▶ Compared to fully conn. layers (left), convolutional layers (right) **share weights**
- ▶ Convolutional layers are **translation equivariant**: $\mathcal{T}_\theta[f](\mathbf{H}) = f(\mathcal{T}_\theta[\mathbf{H}])$

Convolutional Layers



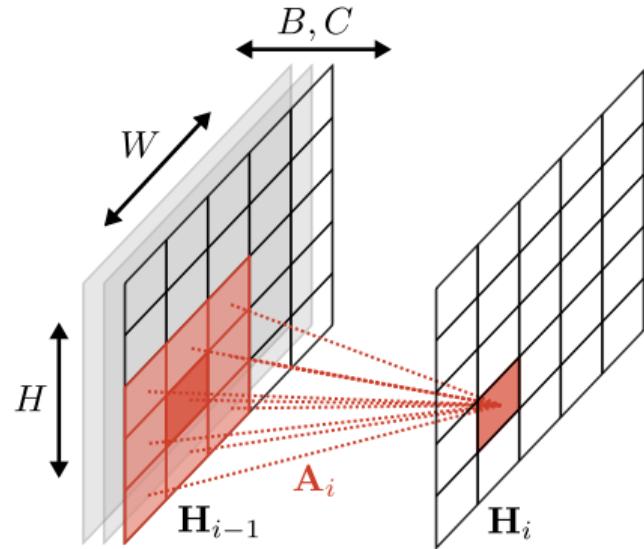
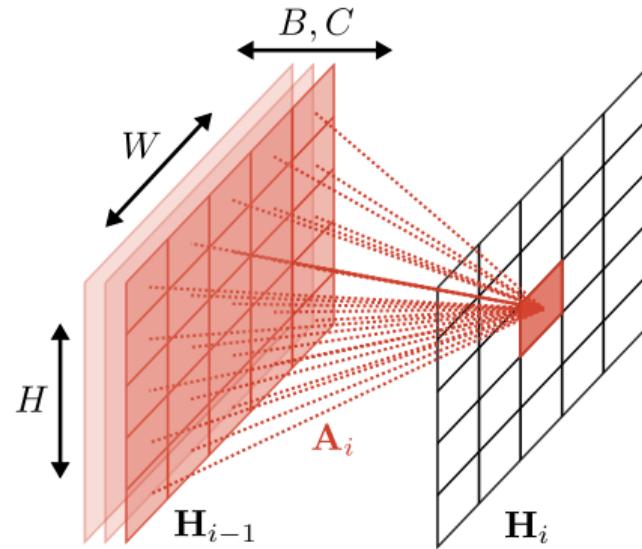
- ▶ Compared to fully conn. layers (left), convolutional layers (right) **share weights**
- ▶ Convolutional layers are **translation equivariant**: $\mathcal{T}_\theta[f](\mathbf{H}) = f(\mathcal{T}_\theta[\mathbf{H}])$

Convolutional Layers



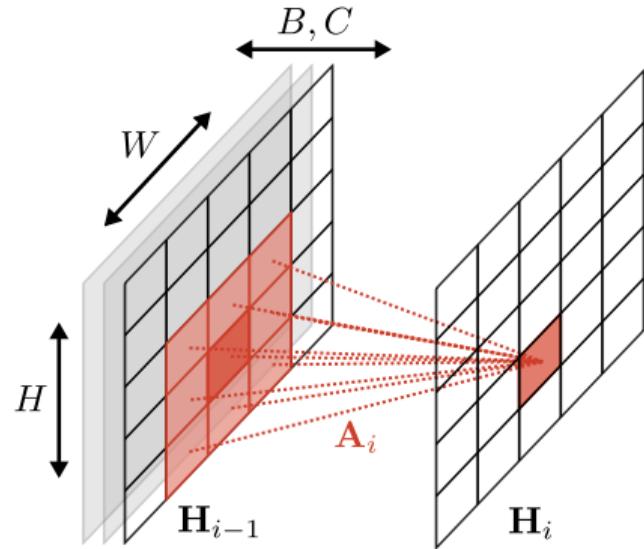
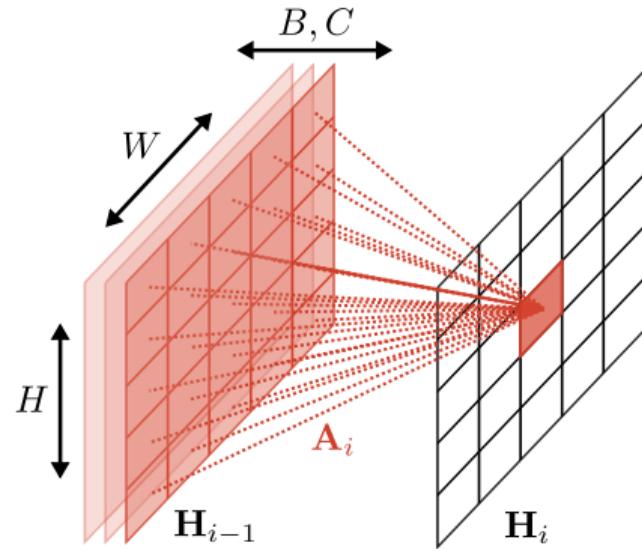
- ▶ Compared to fully conn. layers (left), convolutional layers (right) **share weights**
- ▶ Convolutional layers are **translation equivariant**: $\mathcal{T}_\theta[f](\mathbf{H}) = f(\mathcal{T}_\theta[\mathbf{H}])$

Convolutional Layers



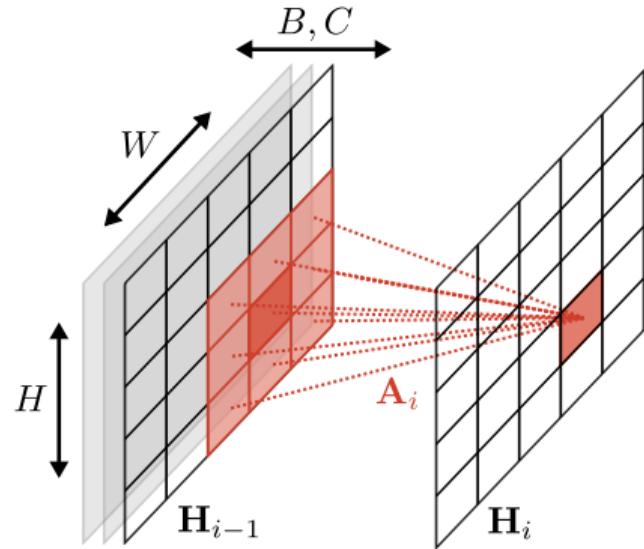
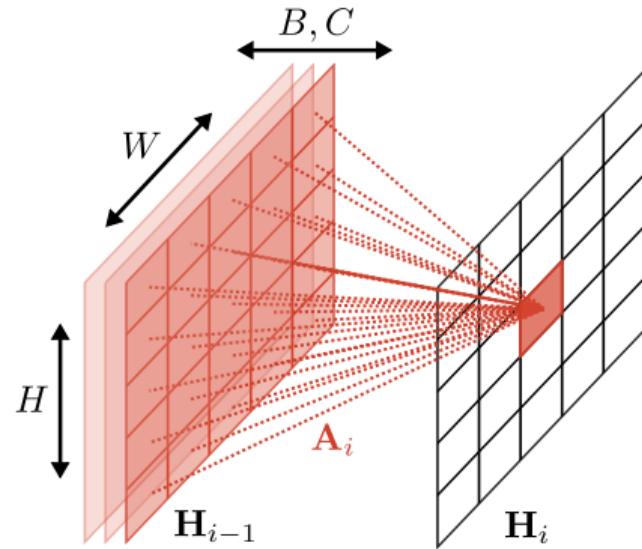
- ▶ Compared to fully conn. layers (left), convolutional layers (right) **share weights**
- ▶ Convolutional layers are **translation equivariant**: $\mathcal{T}_\theta[f](\mathbf{H}) = f(\mathcal{T}_\theta[\mathbf{H}])$

Convolutional Layers



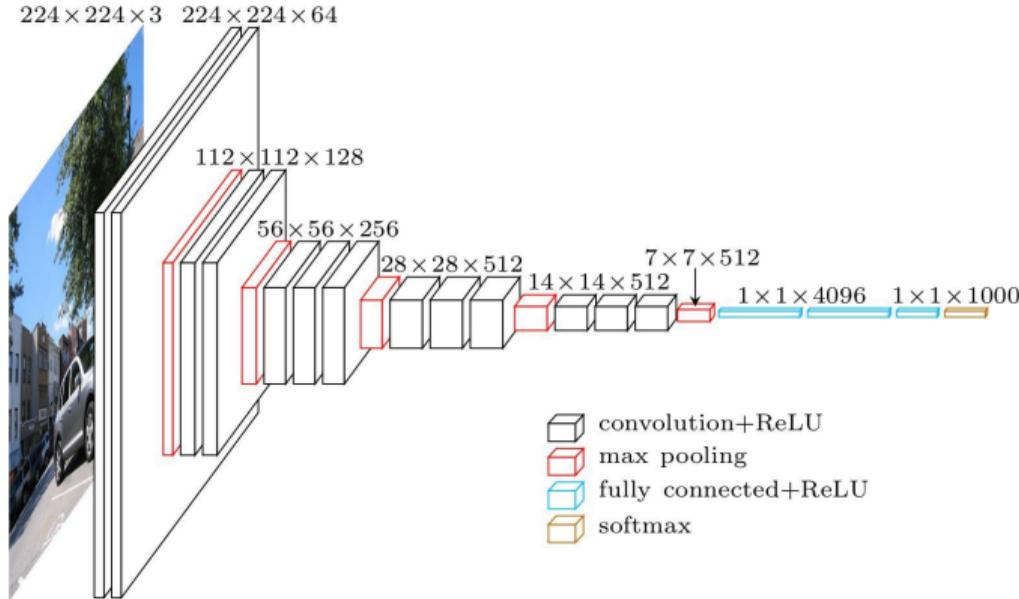
- ▶ Compared to fully conn. layers (left), convolutional layers (right) **share weights**
- ▶ Convolutional layers are **translation equivariant**: $\mathcal{T}_\theta[f](\mathbf{H}) = f(\mathcal{T}_\theta[\mathbf{H}])$

Convolutional Layers



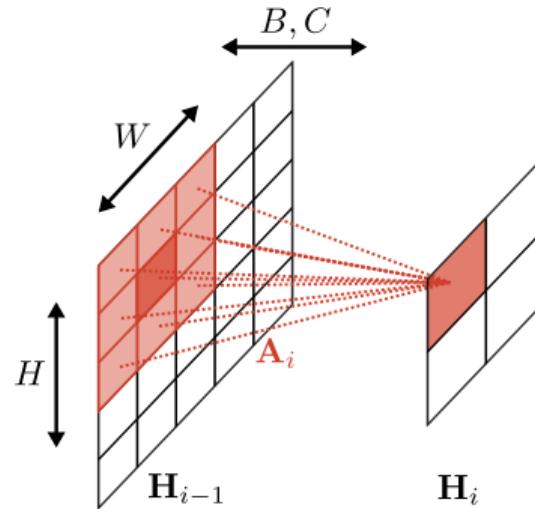
- ▶ Compared to fully conn. layers (left), convolutional layers (right) **share weights**
- ▶ Convolutional layers are **translation equivariant**: $\mathcal{T}_\theta[f](\mathbf{H}) = f(\mathcal{T}_\theta[\mathbf{H}])$

Downsampling



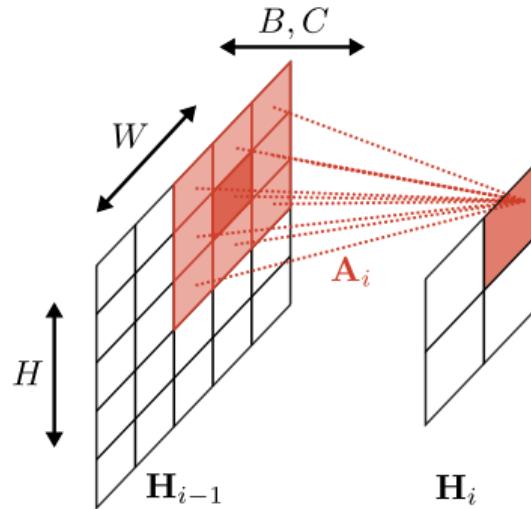
- Downsampling **reduces the spatial resolution** (e.g., for image level predictions)
- Downsampling **increases the receptive field** (which pixels influence a neuron)

Pooling



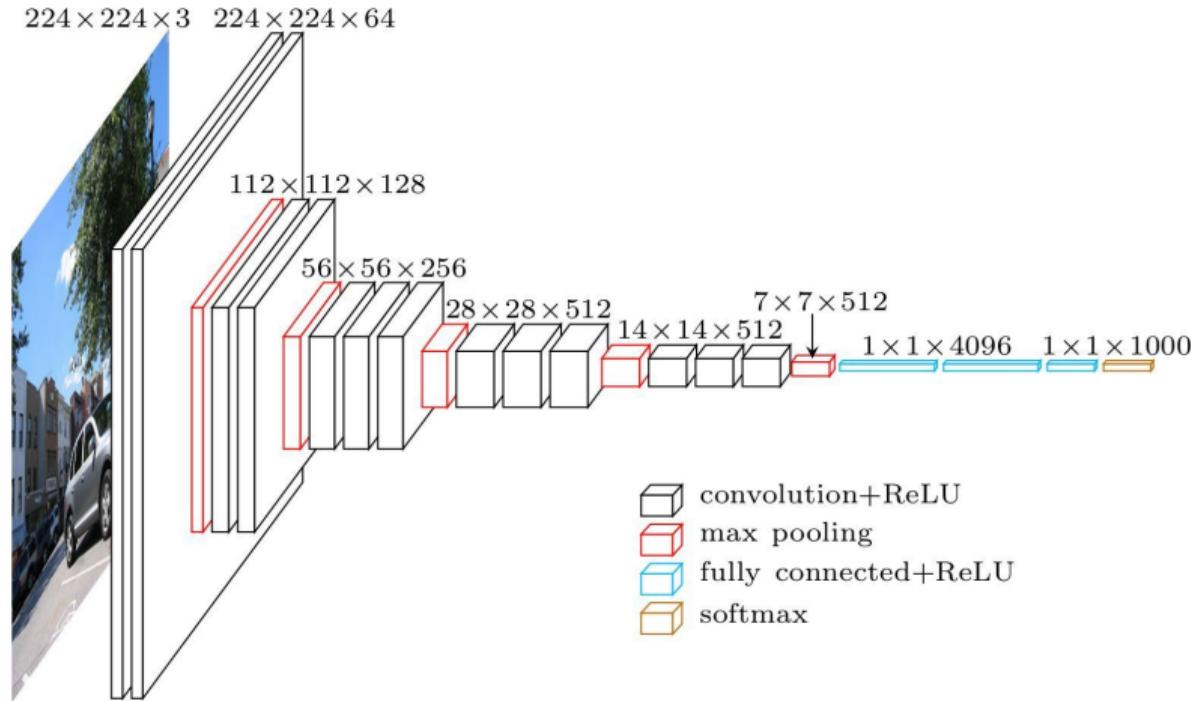
- ▶ Typically, stride $s = 2$ and kernel size $2 \times 2 \Rightarrow$ **reduces spatial dimensions** by 2
- ▶ Pooling has **no parameters** (typical pooling operations: max, min, mean)
- ▶ Pooling is **applied to each channel separately** \Rightarrow keeps number of channels

Pooling



- ▶ Typically, stride $s = 2$ and kernel size $2 \times 2 \Rightarrow$ **reduces spatial dimensions** by 2
- ▶ Pooling has **no parameters** (typical pooling operations: max, min, mean)
- ▶ Pooling is **applied to each channel separately** \Rightarrow keeps number of channels

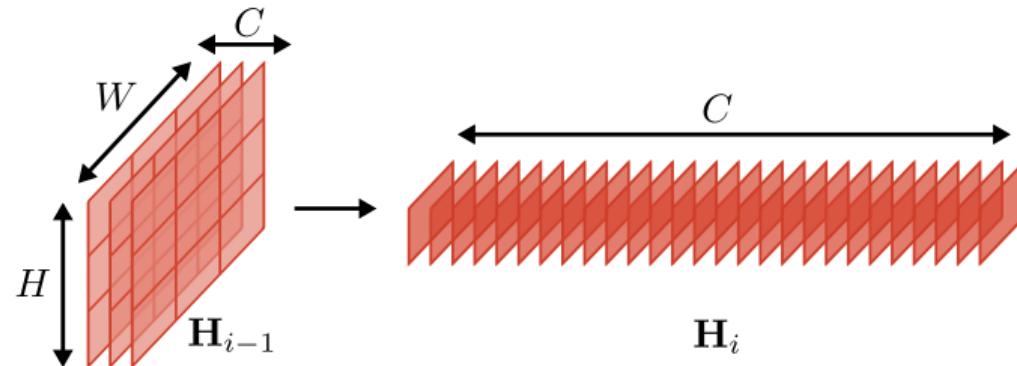
Fully Connected Layers



- ▶ Fully connected layers are most **memory intensive** part of VGG architecture

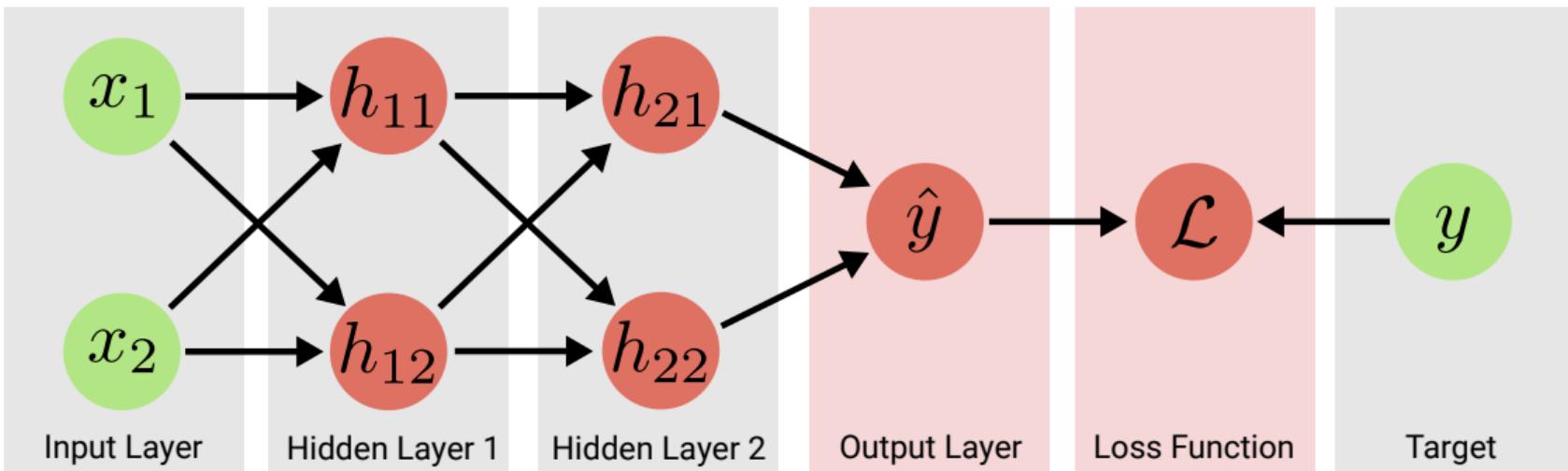
Fully Connected Layers

- Reshape $H_{i-1}[B, X, Y, C]$ into $H_i[B, C]$



- Now X and Y are **reshaped** into the feature channel dimension C

Output and Loss Functions



- The **output layer** is the last layer in a neural network which computes the output
- The **loss function** compares the result of the output layer to the target value(s)
- In image classification, we use a **softmax** output layer and a **cross entropy** loss

Categorical Distribution

Categorical distribution:

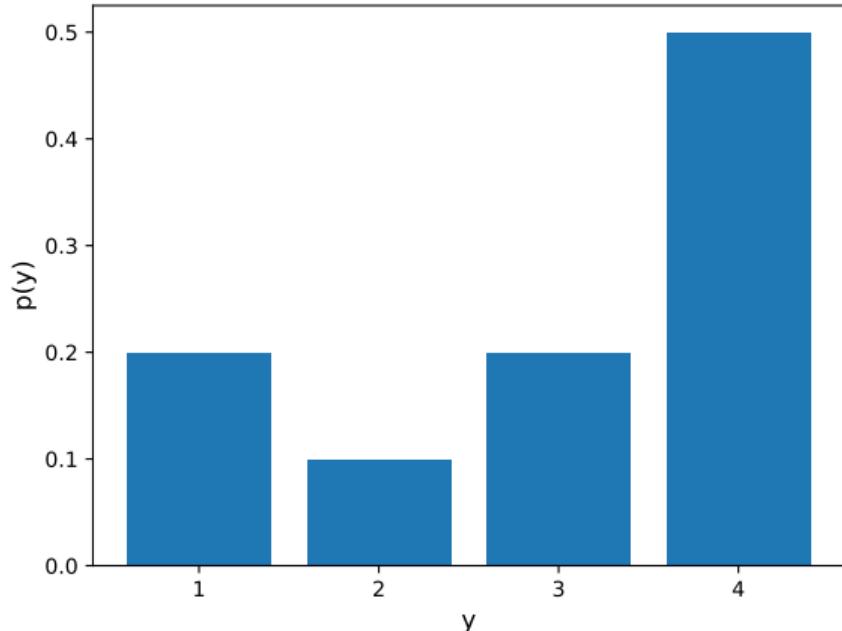
$$p(y = c) = \mu_c$$

- μ_c : probability for class c

Alternative notation:

$$p(\mathbf{y}) = \prod_{c=1}^C \mu_c^{y_c}$$

- \mathbf{y} : “one-hot” vector with $y_c \in \{0, 1\}$
- $\mathbf{y} = (0, \dots, 0, 1, 0, \dots, 0)^\top$ with all zeros except for one (the true class)



One-Hot Vector Representation

class	y	\mathbf{y}
	1	$(1, 0, 0, 0)^\top$
	2	$(0, 1, 0, 0)^\top$
	3	$(0, 0, 1, 0)^\top$
	4	$(0, 0, 0, 1)^\top$

- ▶ One-hot vector \mathbf{y} with binary elements $y_c \in \{0, 1\}$
- ▶ Index c where $y_c = 1$ determines the correct class
- ▶ Often used in ML as it can make formalism more convenient

Categorical Distribution / CE Loss

Let $p_{model}(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \prod_{c=1}^C f_{\mathbf{w}}^{(c)}(\mathbf{x})^{y_c}$ be a **Categorical distribution**.

Maximizing the (log-)likelihood leads to the **cross-entropy (CE)** loss:

$$\begin{aligned}\hat{\mathbf{w}}_{ML} &= \operatorname{argmax}_{\mathbf{w}} \sum_{i=1}^N \log p_{model}(\mathbf{y}_i | \mathbf{x}_i, \mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{w}} \sum_{i=1}^N \log \prod_{c=1}^C f_{\mathbf{w}}^{(c)}(\mathbf{x}_i)^{y_{i,c}} \\ &= \operatorname{argmin}_{\mathbf{w}} \underbrace{\sum_{i=1}^N \sum_{c=1}^C -y_{i,c} \log f_{\mathbf{w}}^{(c)}(\mathbf{x}_i)}_{\text{CE Loss}}\end{aligned}$$

The target $\mathbf{y} = (0, \dots, 0, 1, 0, \dots, 0)^\top$ is a “one-hot” vector with y_c its c 'th element.

Softmax

How can we ensure that $f_{\mathbf{w}}^{(c)}(\mathbf{x})$ predicts a **valid Categorical distribution?**

- We must guarantee (1) $f_{\mathbf{w}}^{(c)}(\mathbf{x}) \in [0, 1]$ and (2) $\sum_{c=1}^C f_{\mathbf{w}}^{(c)}(\mathbf{x}) = 1$
- The **softmax function** guarantees both (1) and (2):

$$\text{softmax}(\mathbf{x}) = \left(\frac{\exp(x_1)}{\sum_{k=1}^C \exp(x_k)}, \dots, \frac{\exp(x_C)}{\sum_{k=1}^C \exp(x_k)} \right)$$

- Let the score vector \mathbf{s} denote the network output after the last affine layer. Then:

$$f_{\mathbf{w}}^{(c)}(\mathbf{x}) = \frac{\exp(s_c)}{\sum_{k=1}^C \exp(s_k)} \quad \Rightarrow \quad \log f_{\mathbf{w}}^{(c)}(\mathbf{x}) = s_c - \log \sum_{k=1}^C \exp(s_k)$$

Log Softmax

Intuition: Assume c is the correct class. Our goal is to maximize the log softmax:

$$\log f_{\mathbf{w}}^{(c)}(\mathbf{x}) = s_c - \log \sum_{k=1}^C \exp(s_k)$$

- ▶ The first term encourages the score s_c for the correct class c to increase
- ▶ The second term encourages all scores in \mathbf{s} to decrease (negative sign)
- ▶ The second term can be approximated by: $\log \sum_{k=1}^C \exp(s_k) \approx \max_k s_k$ as $\exp(s_k)$ is insignificant for all $s_k < \max_k s_k$
- ▶ Thus, the loss always strongly penalizes the most active incorrect prediction
- ▶ If the correct class already has the largest score (i.e., $s_c = \max_k s_k$), both terms roughly cancel and the example will contribute little to the overall training cost

Softmax

- ▶ Softmax responds to differences between inputs
- ▶ It is invariant to adding the same scalar to all its inputs:

$$\text{softmax}(\mathbf{x}) = \text{softmax}(\mathbf{x} + c)$$

- ▶ We can therefore derive a numerically more stable variant:

$$\text{softmax}(\mathbf{x}) = \text{softmax}\left(\mathbf{x} - \max_{k=1..L} x_k\right)$$

- ▶ Allows accurate computation even when \mathbf{x} is large

Cross Entropy Loss Example

Putting it together: Cross Entropy Loss for a single training sample $(\mathbf{x}, \mathbf{y}) \in \mathcal{X}$:

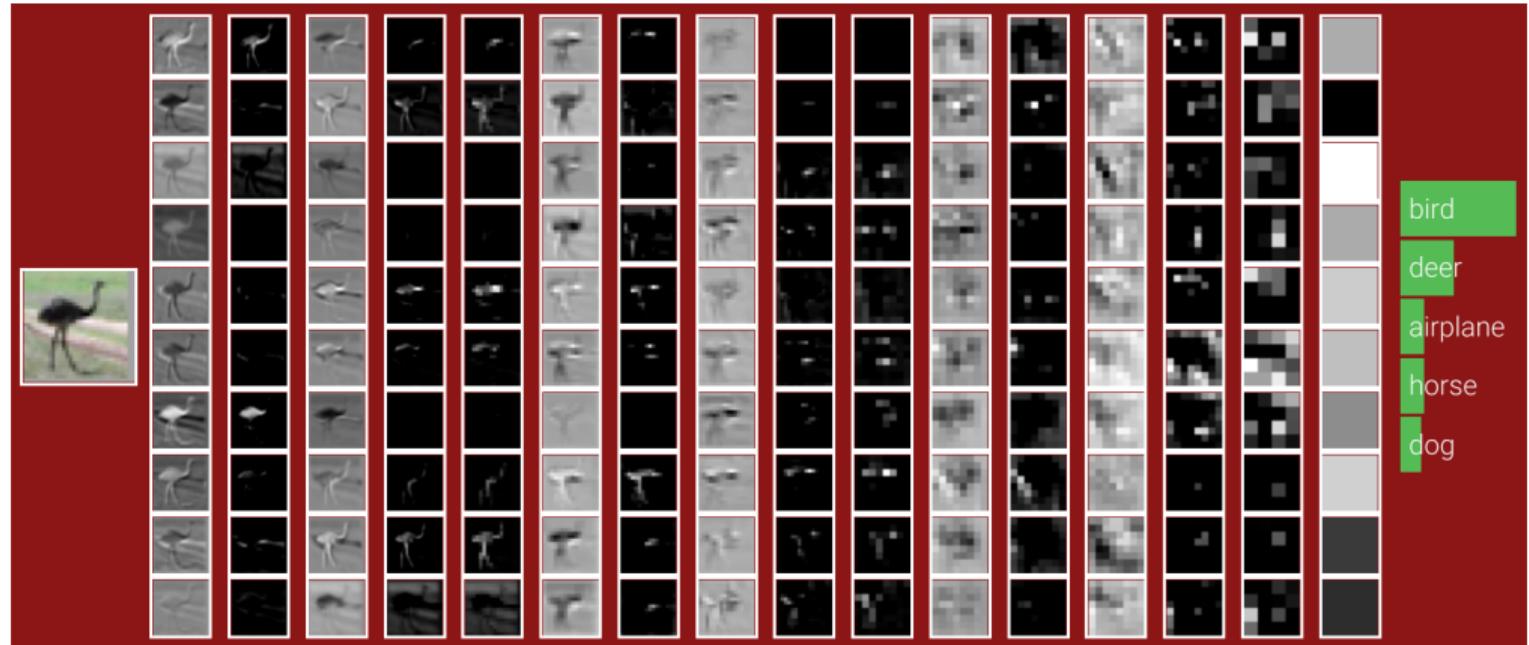
$$\text{CE Loss: } \sum_{c=1}^C -y_c \log f_{\mathbf{w}}^{(c)}(\mathbf{x})$$

Example: Suppose 4 classes and 4 training samples:

Input \mathbf{x}	Label \mathbf{y}	Predicted scores \mathbf{s}	softmax(\mathbf{s})	CE Loss
	$(1, 0, 0, 0)^T$	$(+3, +1, -1, -1)^T$	$(0.85, 0.12, 0.02, 0.02)^T$	0.16
	$(0, 1, 0, 0)^T$	$(+3, +3, +1, +0)^T$	$(0.46, 0.46, 0.06, 0.02)^T$	0.78
	$(0, 0, 1, 0)^T$	$(+1, +1, +1, +1)^T$	$(0.25, 0.25, 0.25, 0.25)^T$	1.38
	$(0, 0, 0, 1)^T$	$(+3, +2, +3, -1)^T$	$(0.42, 0.16, 0.42, 0.01)^T$	4.87

We observe that sample 4 contributes most strongly to the loss function!

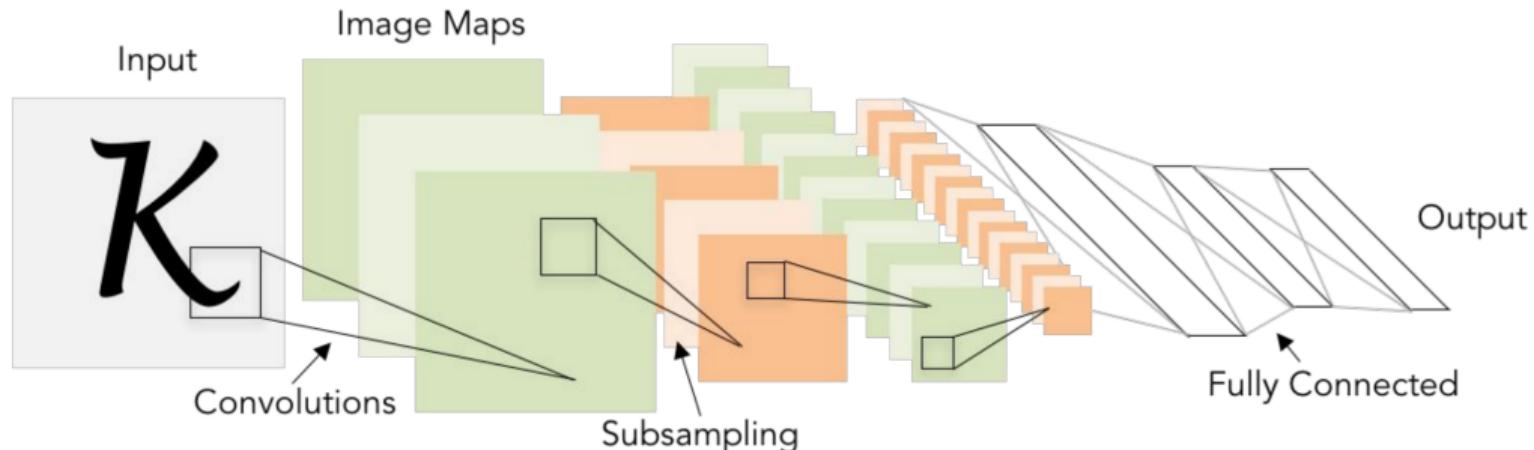
Convolutional Neural Network Example



<http://cs231n.stanford.edu/2017/index.html>

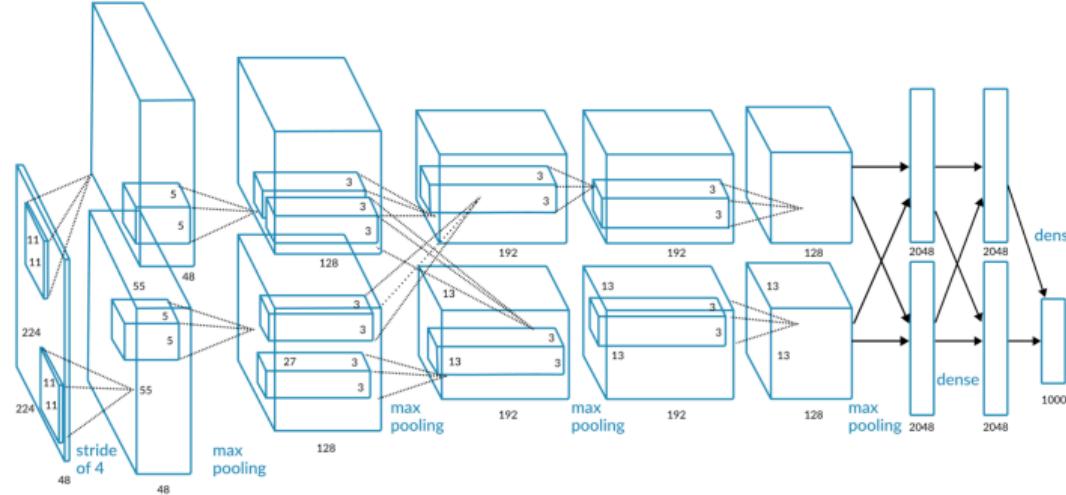
Network Architectures

1998: LeNet-5



- ▶ 2 convolution layers (5×5), 2 pooling layers (2×2), 2 fully connected layers
- ▶ Achieved state-of-the-art accuracy on MNIST (prior to ImageNet)

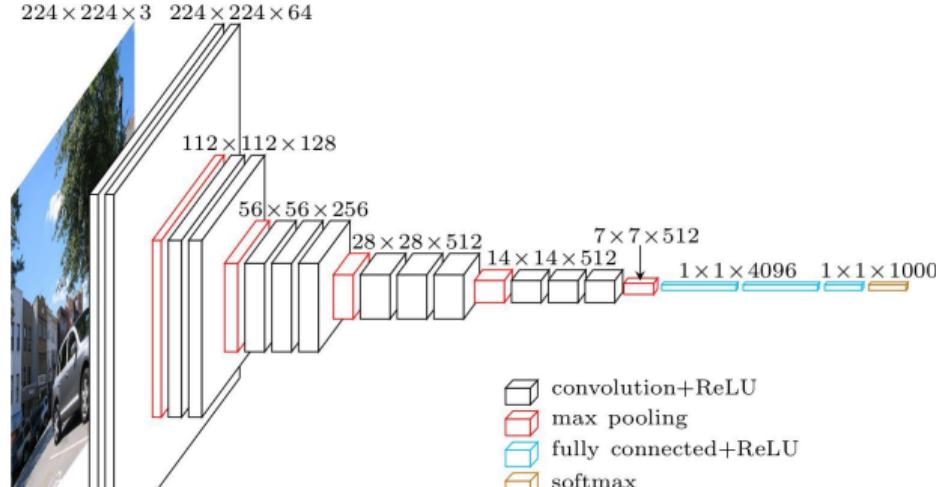
2012: AlexNet



AlexNet:

- ▶ 8 layers, ReLUs, dropout, data augmentation, trained on 2 GTX 580 GPUs
- ▶ Number of feature channels increase with depth, spatial resolution decreases
- ▶ Triggered deep learning revolution, showed that CNNs work well in practice

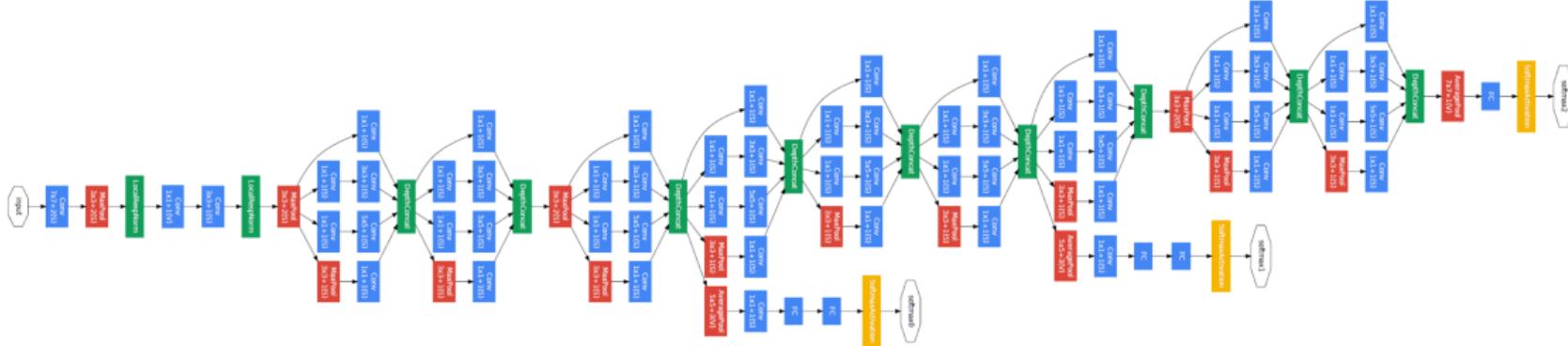
2015: VGG



VGG:

- ▶ Uses 3×3 convolutions everywhere (same expressiveness, fewer parameters)
- ▶ Three 3×3 layers: same receptive field as one 7×7 layer, but less parameters
- ▶ 2 Variants: 16 and 19 Layers. Showed that deeper networks are better.

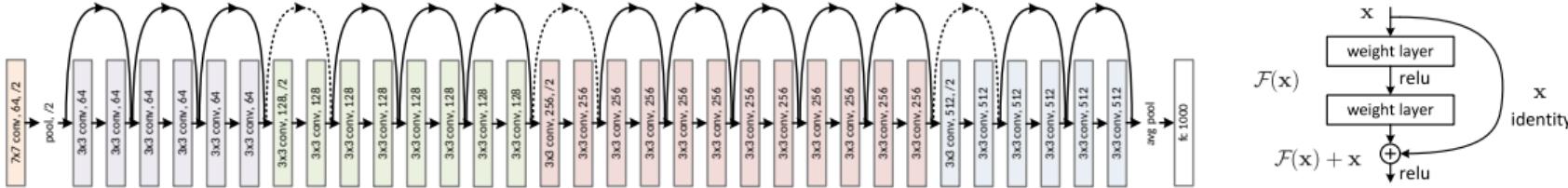
2015: Inception / GoogLeNet



Inception:

- ▶ 22 layers. Inception modules utilize conv/pool operations with varying filter size
- ▶ Multiple intermediate classification heads to improve gradient flow
- ▶ Global average pooling (no FC layers), 27x less parameters (5 million) than VGG-16

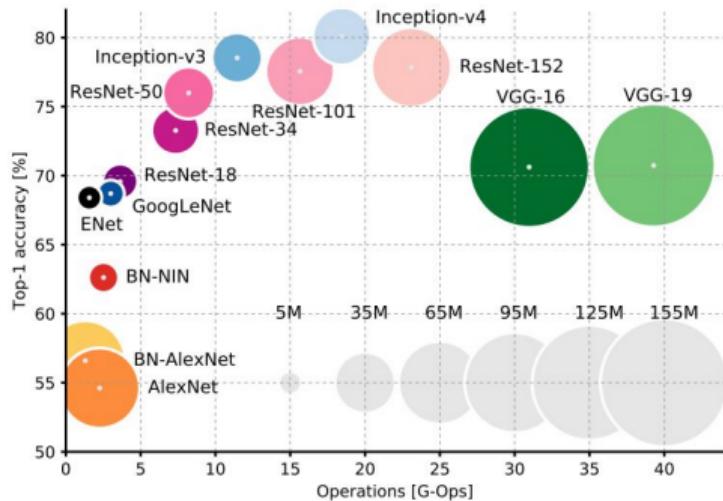
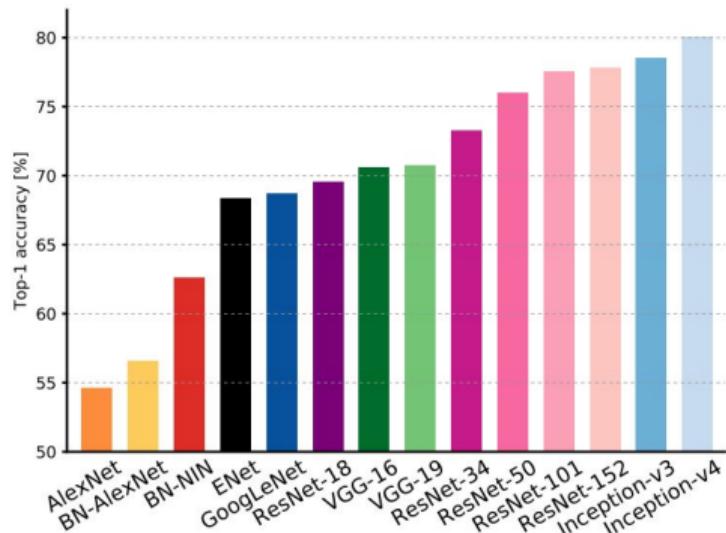
2016: ResNet



ResNet:

- ▶ Residual connections allow for training deeper networks (up to 152 layers)
- ▶ Very simple and regular network structure with 3×3 convolutions
- ▶ Uses strided convolutions for downsampling
- ▶ ResNet and ResNet-like architectures are dominating today

Accuracy vs. Complexity

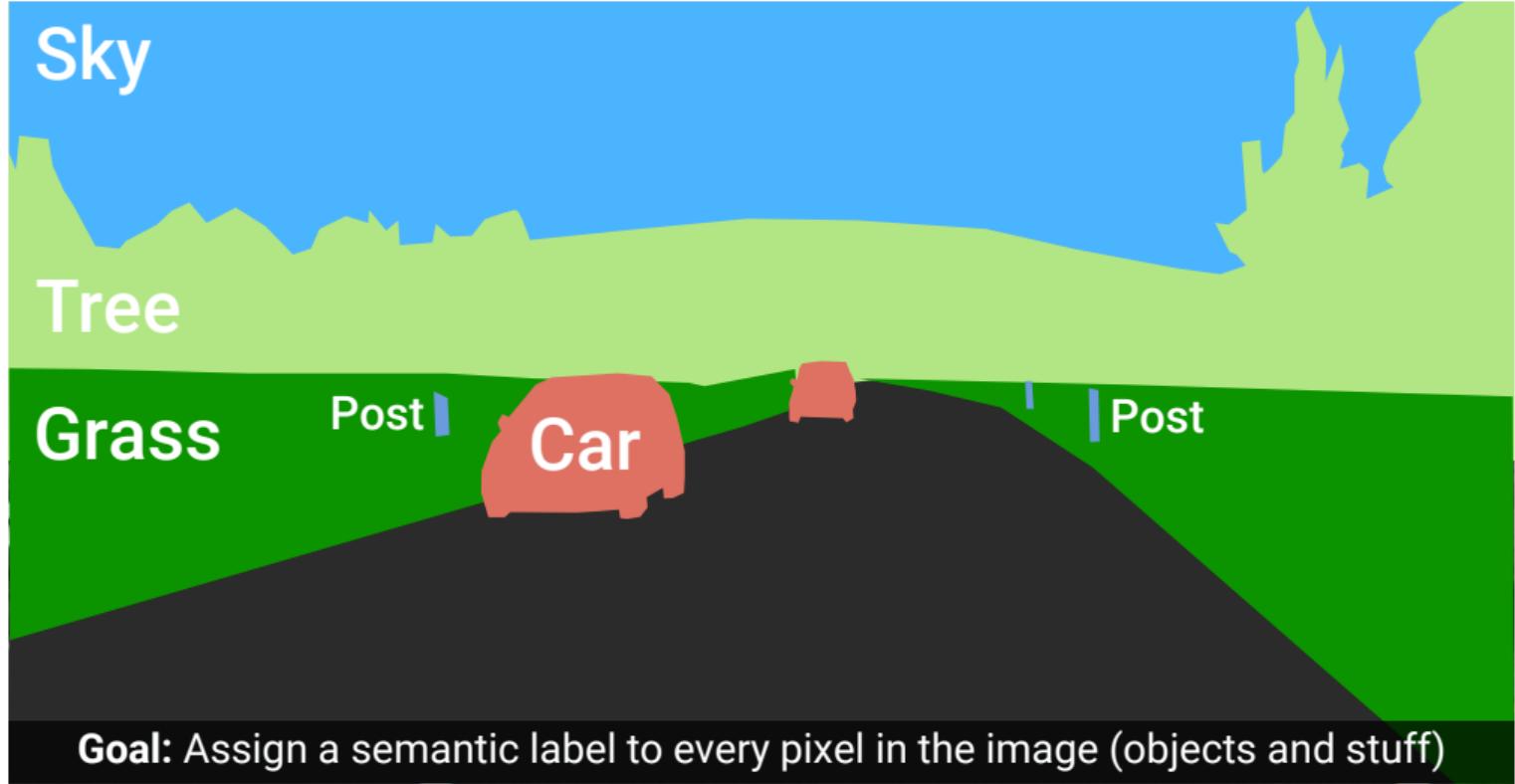


- ▶ Performance: Top-1 or Top-5 accuracy (target label in top N predictions)
- ▶ VGG has most parameters and is slowest
- ▶ ResNet/Inception/GoogLeNet are faster and have fewer parameters

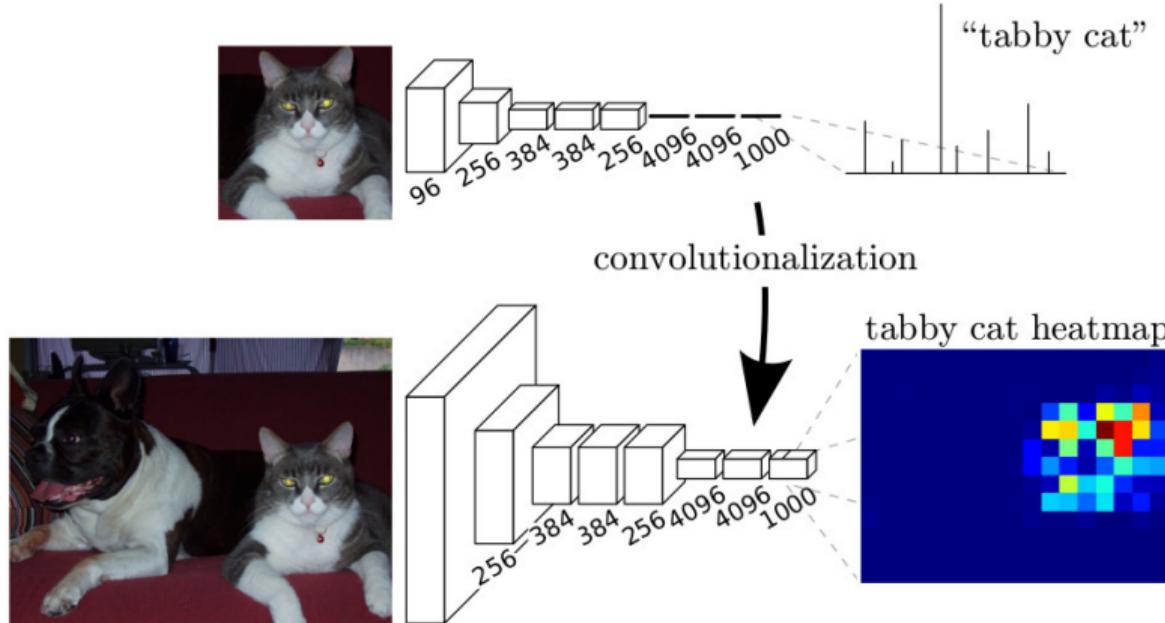
10.2

Semantic Segmentation

Semantic Segmentation



Fully Convolutional Networks



- ▶ Apply classification network in convolutional fashion to obtain class heatmaps
- ▶ Use cross entropy loss at every pixel of the output (sum over classification losses)

Fully Convolutional Networks

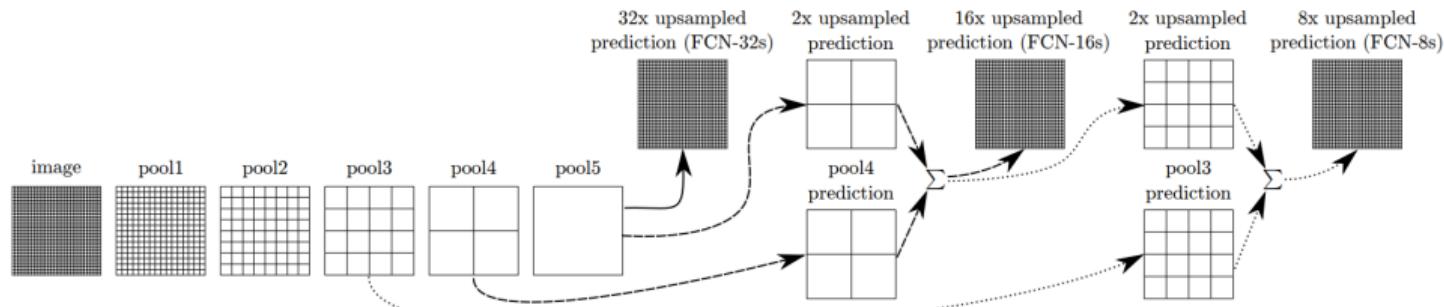
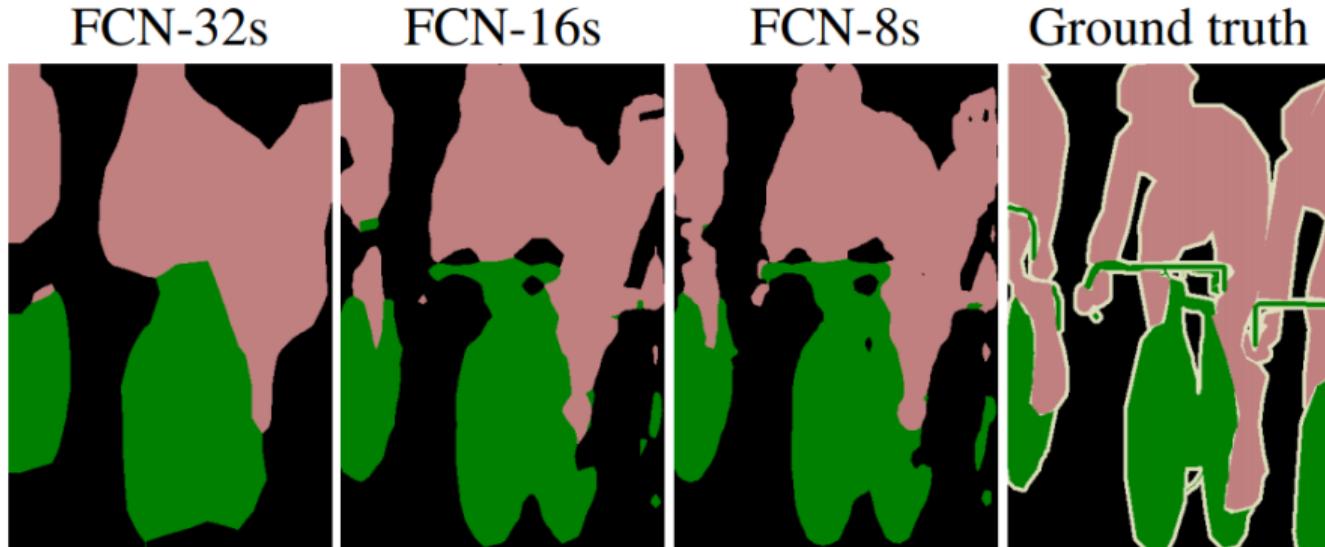


Figure 3. Our DAG nets learn to combine coarse, high layer information with fine, low layer information. Layers are shown as grids that reveal relative spatial coarseness. Only pooling and prediction layers are shown; intermediate convolution layers (including our converted fully connected layers) are omitted. Solid line (FCN-32s): Our single-stream net, described in Section 4.1, upsamples stride 32 predictions back to pixels in a single step. Dashed line (FCN-16s): Combining predictions from both the final layer and the pool4 layer, at stride 16, lets our net predict finer details, while retaining high-level semantic information. Dotted line (FCN-8s): Additional predictions from pool3, at stride 8, provide further precision.

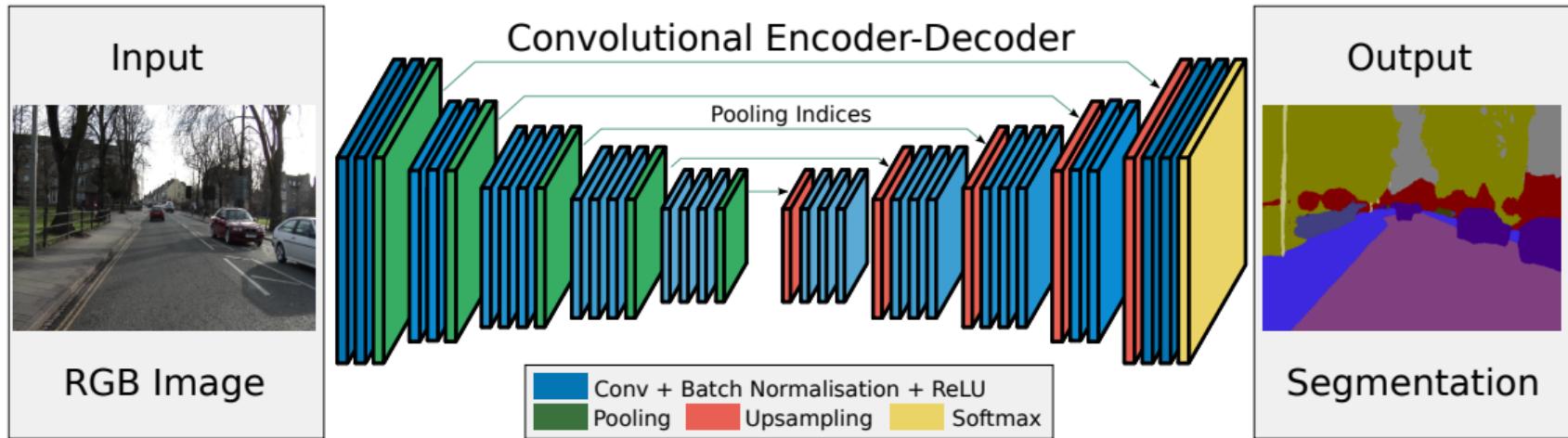
- ▶ Problem: Output heatmaps are **low resolution** due to downsampling operations
- ▶ But using only convolutional filters does not work due to small receptive field
- ▶ Idea: **Learn upsampling** operation and combine **low- and high-level** information

Fully Convolutional Networks



- ▶ Fusing information from layers with different strides (FCN-8s) improves detail
- ▶ However, output of early 2015 results still relatively coarse

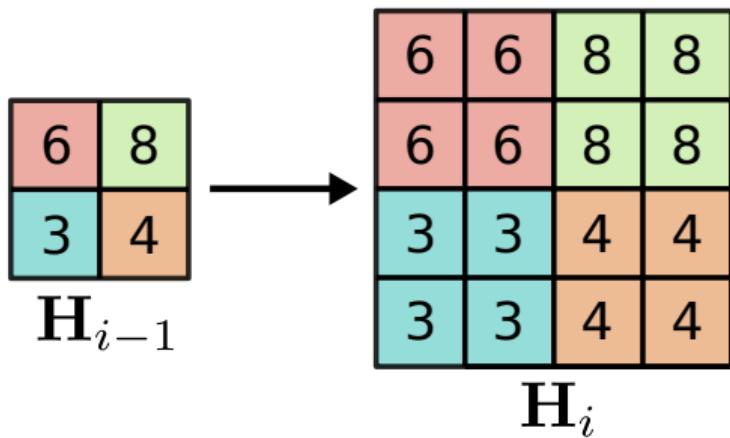
SegNet



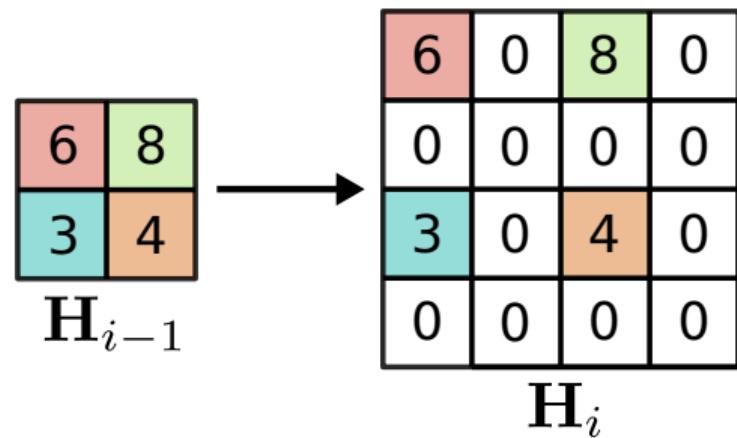
- ▶ SegNet is another model that uses the idea of down- and upsampling
- ▶ Downsampling provides strong features with large receptive field
- ▶ Upsampling yields output at the same resolution as input

Upsampling

Nearest Neighbor Upsampling

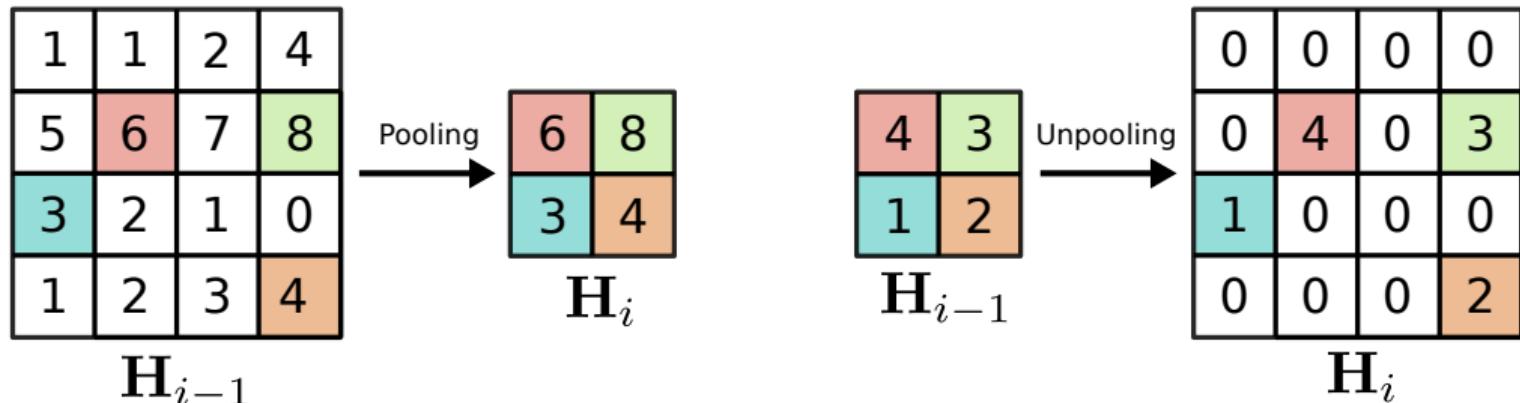


Bed of Nails Upsampling



- ▶ **Nearest neighbor:** scale each channel using nearest neighbor interpolation
- ▶ **Bilinear:** scale each channel using bilinear neighbor interpolation
- ▶ **Bed of nails:** insert elements at sparse location (followed by convolution)

Max Unpooling



- ▶ For unpooling, **remember** which element was **maximum** during pooling
- ▶ Requires corresponding pairs of downsampling and upsampling layers
- ▶ This approach has been used in SegNet

SegNet Results

Input



Segnet



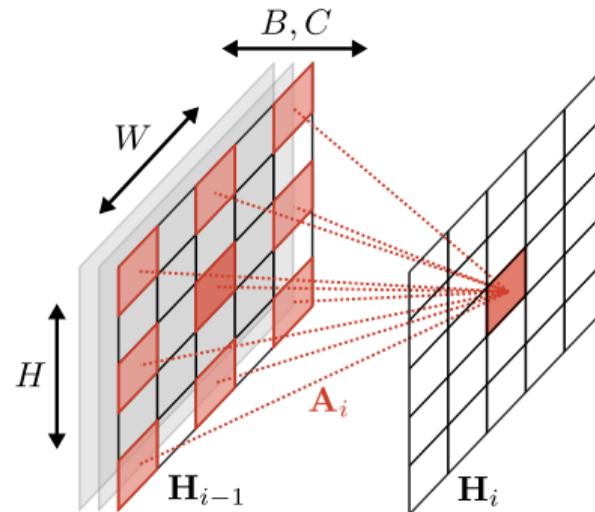
FCN



Dilated Convolution

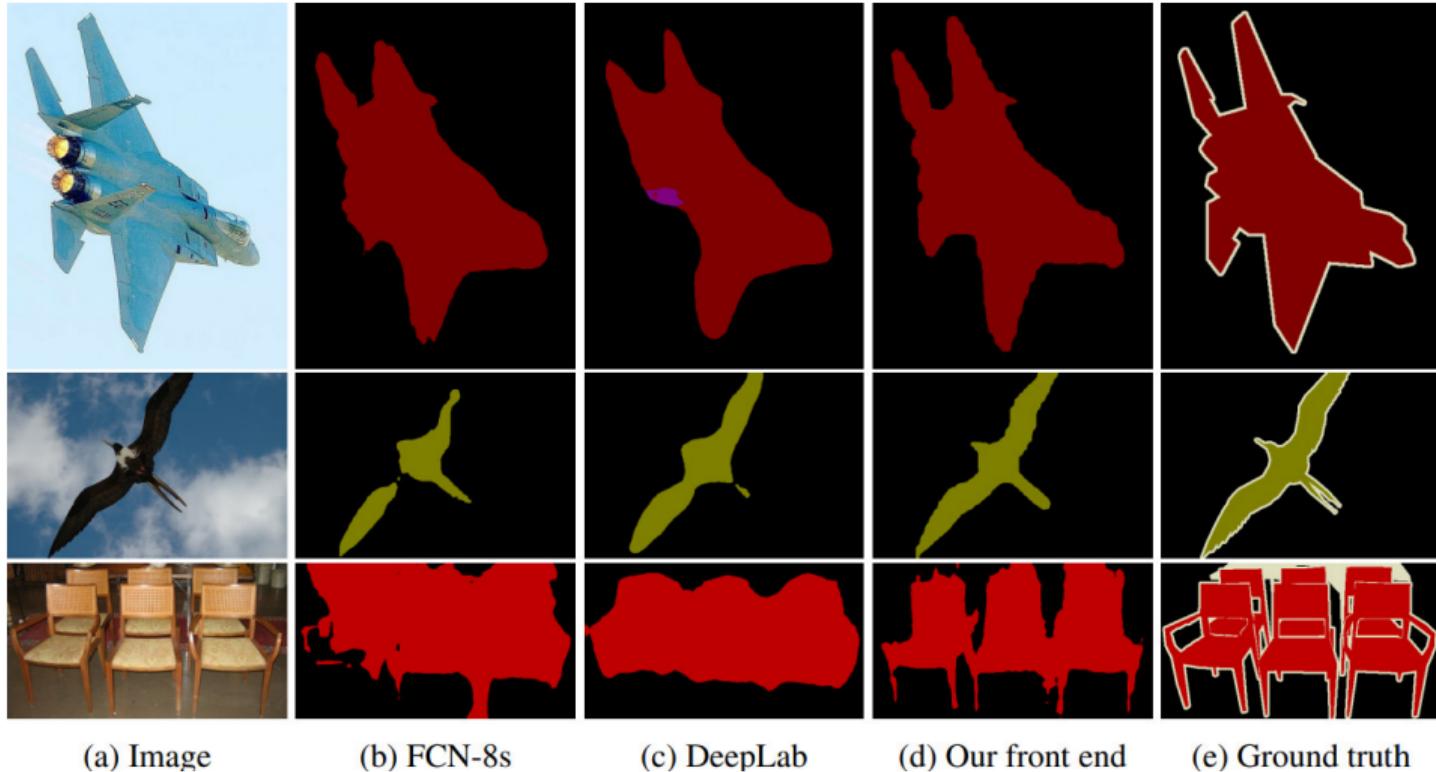
- ▶ Dilated convolutions are an alternative to combining downsampling and upsampling operations in order to reach a large receptive field size quickly
- ▶ Dilated convolutions increase the receptive field of standard convolutions without increasing the number of parameters
- ▶ Thus, a network with dilated convolutions is able to perform image-level predictions (e.g., semantic segmentation) without upsampling and downsampling
- ▶ However, in practice, dilated convolutions combined with standard backbone

Dilated Convolution



- ▶ Dilation factor d (here $d = 2$) leads to more “distributed” sampling of the input
- ▶ **Increases receptive field** without increasing parameters or reducing spatial dim.

Dilated Convolution



(a) Image

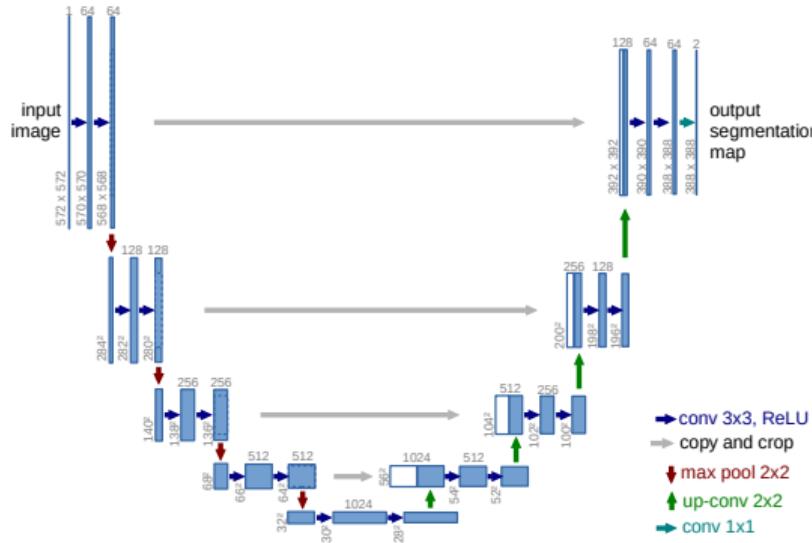
(b) FCN-8s

(c) DeepLab

(d) Our front end

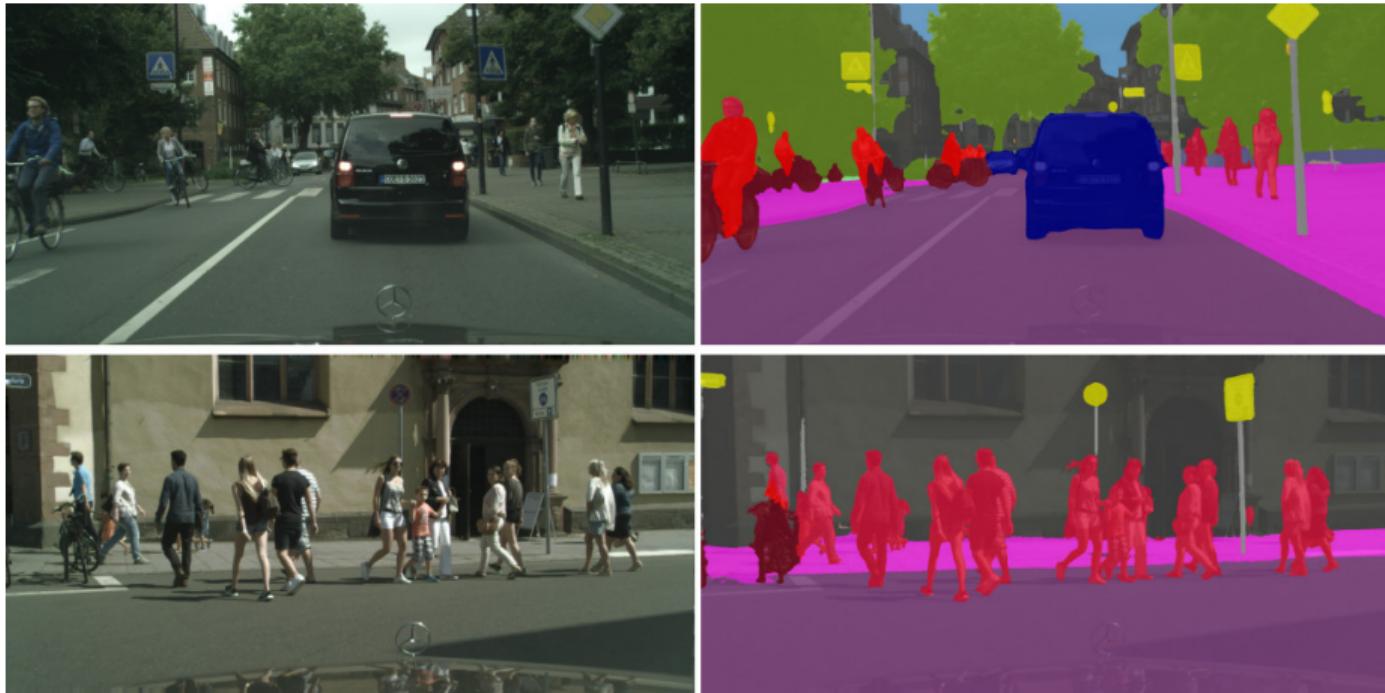
(e) Ground truth

U-Networks



- ▶ U-Nets use max-pooling, up-convolutions and skip-connections (concatenation)
 - ▶ Defacto standard for many tasks with image output (e.g., depth, segmentation)

Feature Space Optimization



- ▶ Temporally consistent semantic segmentation in video sequences

Cityscapes Leaderboard

Evaluation Metrics:

- ▶ Per-pixel accuracy
- ▶ Per-class Jaccard Index / Intersection-over-Union
- ▶ IoU weighted by inv. instance size



CITYSCAPES
DATASET

name	fine	coarse	16-bit	depth	video	sub	IoU class	IoU class	IoU category	IoU category	Runtime [s]	code
							IoU class	IoU category	IoU category	Runtime [s]	code	
AntGroup-AI-VisionAlgo	yes	yes	no	no	no	no	86.1	71.3	93.2	84.7	n/a	no
DAHUA-ARI	yes	yes	no	no	no	no	85.8	70.6	93.2	85.4	n/a	no
Qualcomm AI Research	yes	yes	no	no	no	no	85.6	71.4	93.0	85.7	n/a	no
Hierarchical Multi-Scale Attention for Semantic Segmentation	yes	yes	no	no	no	no	85.4	70.4	93.2	85.4	n/a	yes
DCNAS+ASPP [Mapillary Vistas]	yes	yes	no	no	no	no	85.3	70.0	93.1	85.3	n/a	no
UJS-model	yes	no	no	no	no	no	85.3	70.5	93.1	85.1	n/a	no
Naive-Student (iterative semi-supervised learning with Panoptic-DeepLab)	yes	no	no	no	yes	no	85.2	68.8	92.9	82.0	n/a	no
Panoptic-DeepLab w/ SWideRNet [Mapillary Vistas + Pseudo-labels]	yes	no	no	no	yes	no	85.1	71.2	93.0	85.1	n/a	no
ddl_seg	yes	no	no	no	no	no	84.6	69.4	92.8	84.4	n/a	no
HRNetV2 + OCR + SegFix	yes	yes	no	no	no	no	84.5	65.9	92.7	83.9	n/a	yes
Panoptic-DeepLab [Mapillary Vistas]	yes	no	no	no	no	no	84.5	68.7	92.9	82.8	n/a	no

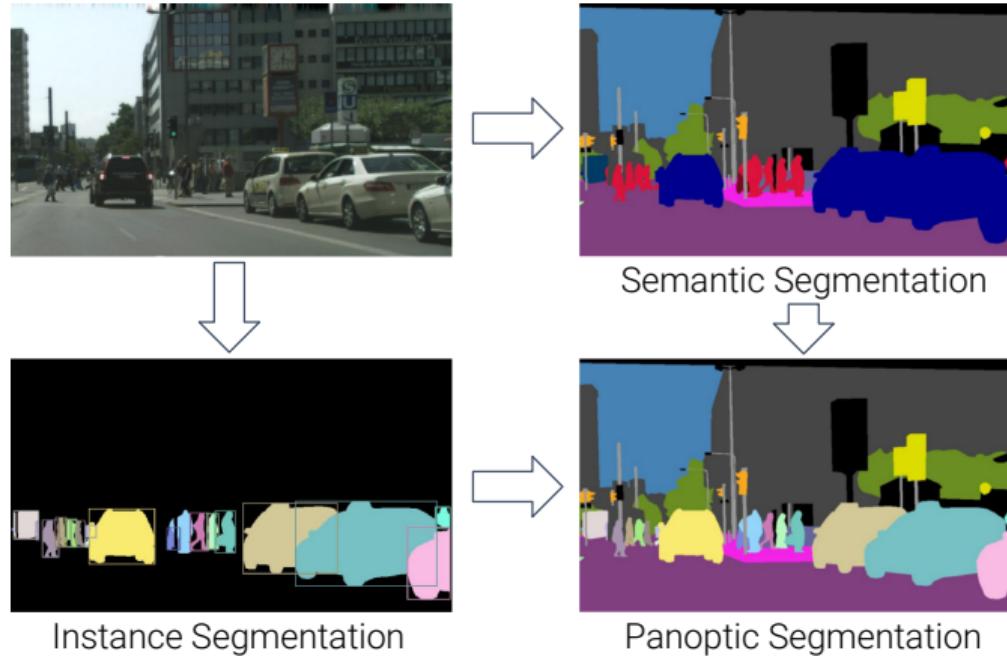
www.cityscapes-dataset.com

Hierarchical Multi-Scale Attention



- State-of-the-art multi-scale approach, currently ranked fourth on Cityscapes

Panoptic Segmentation

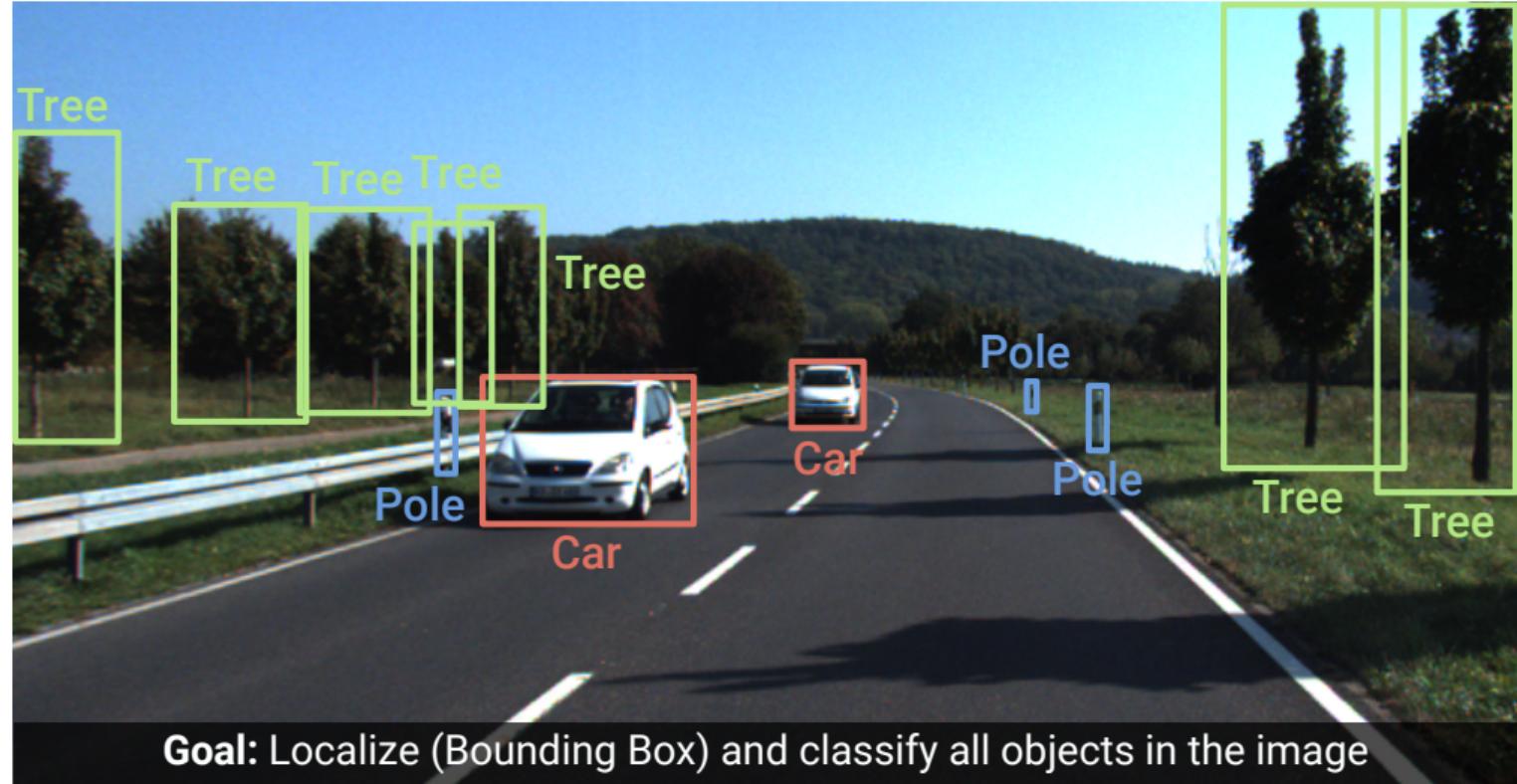


- ▶ Combines semantic and instance segmentation: predict semantic category for both stuff and objects, as well as instance label for each object

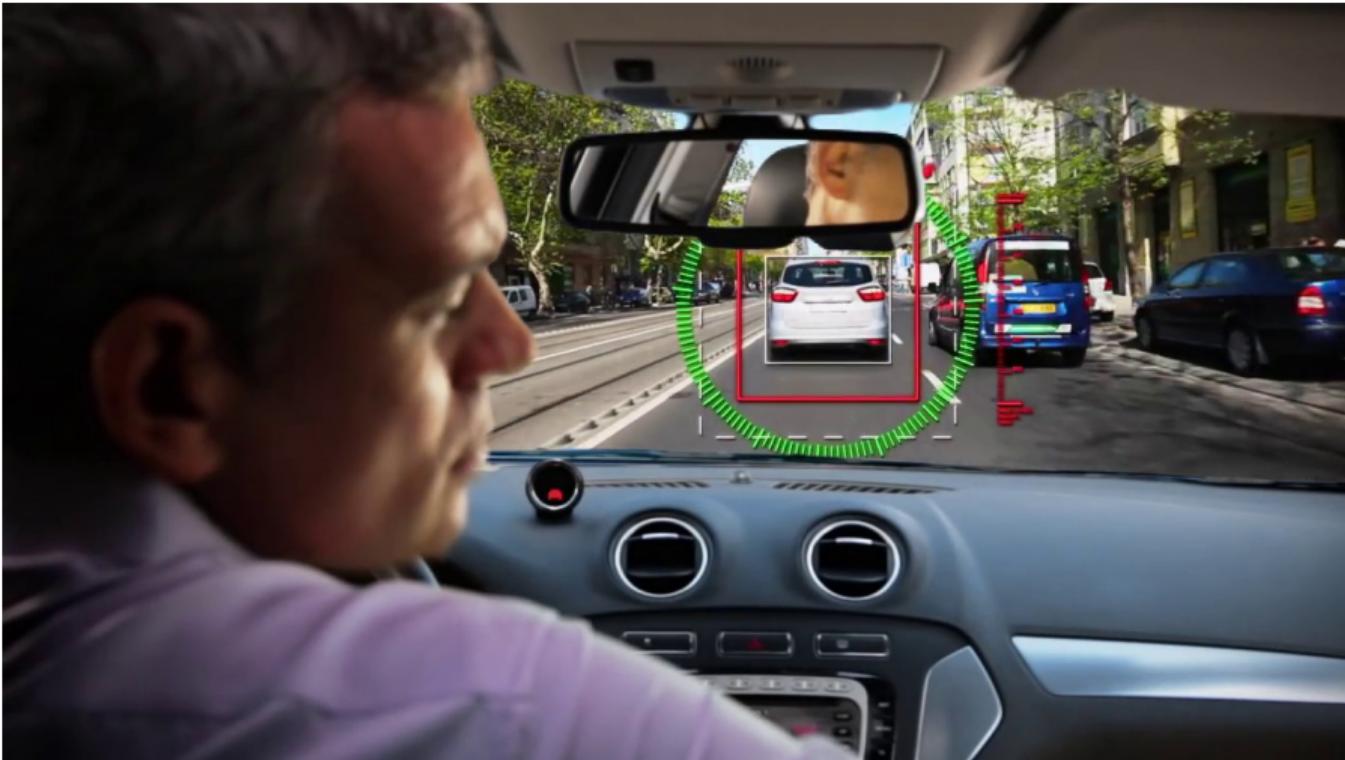
10.3

Object Detection and Segmentation

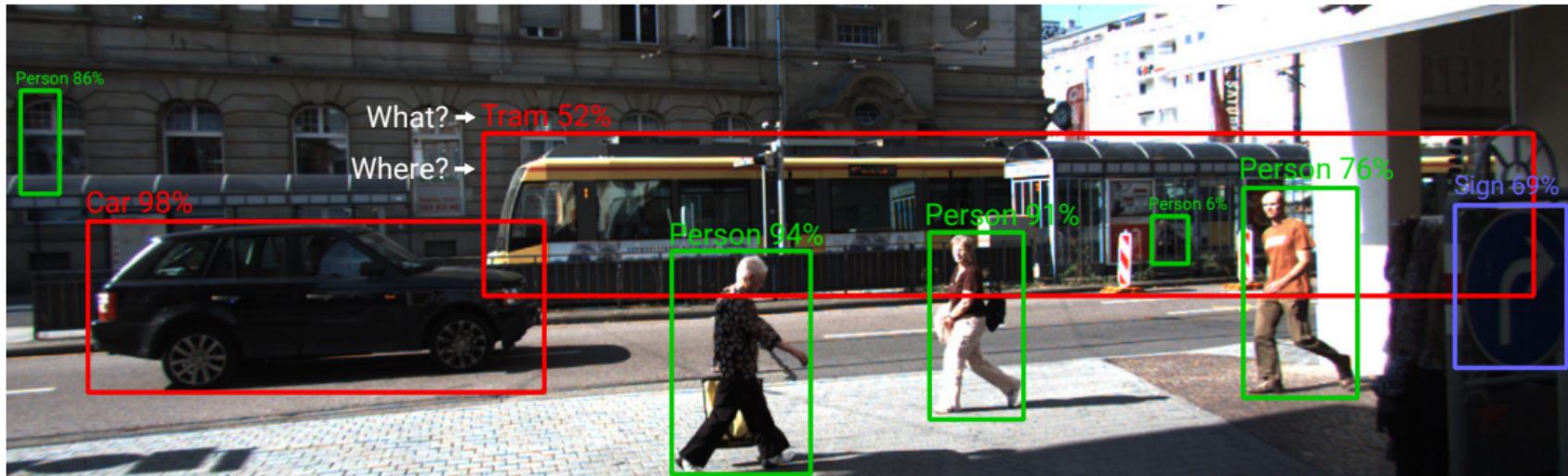
Object Detection



Motivation



Problem Setting



Problem Setting:

- ▶ **Input:** RGB Image or laser range scan.
- ▶ **Output:** Set of 2D/3D bounding boxes with category label and confidence
- ▶ Note that the number of objects and the object size not known a priori

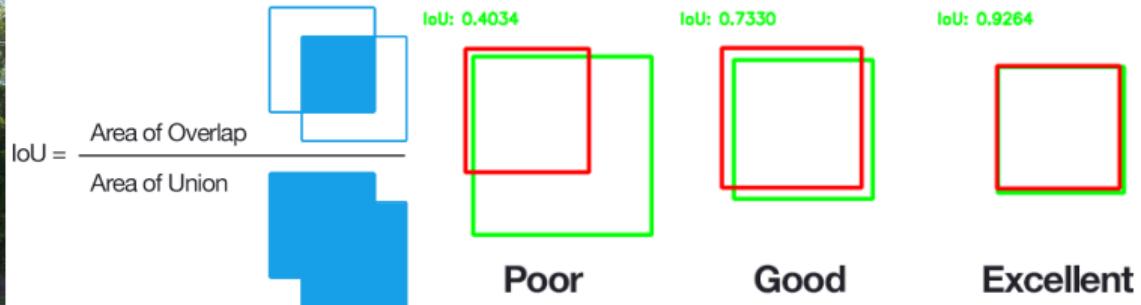
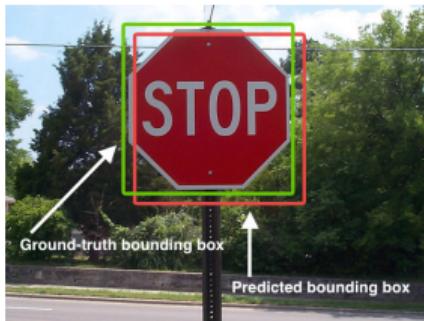
Problem Setting



Problem Setting:

- ▶ **Input:** RGB Image or laser range scan.
- ▶ **Output:** Set of 2D/3D bounding boxes with category label and confidence
- ▶ Note that the number of objects and the object size not known a priori

Performance Evaluation



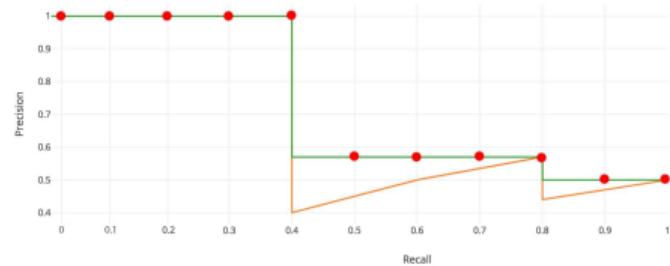
Performance Evaluation:

- ▶ IoU = Intersection-over-Union (**predicted bbox** vs. **true bbox**)
- ▶ Detection considered successful if $\text{IoU} > 0.5$
- ▶ How to fairly measure detection performance in case of multiple objects?
(Number of detections depends on detector threshold and is detector specific!)

Performance Evaluation

Average Precision Metric:

1. Run detector with varying thresholds
2. Assign detections to closest object
3. Count TP, FP, FN
4. Compute **Average Precision (AP)**



True Positives TP: Number of objects correctly detected ($\text{IoU} > 0.5$)

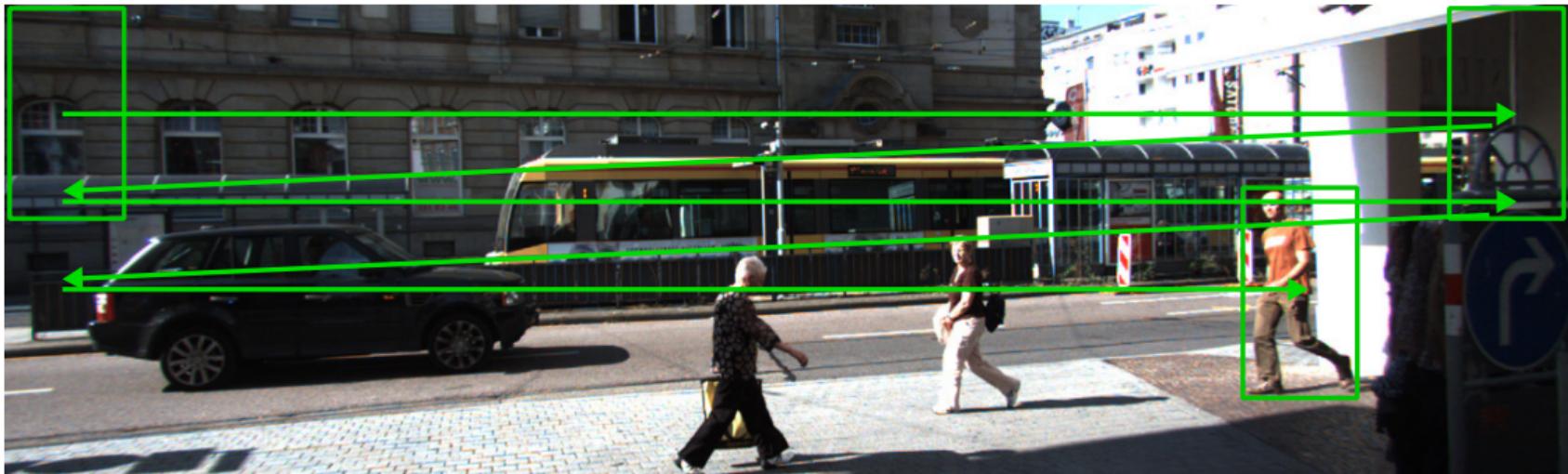
False Negatives FN: Number of objects not detected ($\text{IoU} < 0.5$)

False Positives FP: Wrong detections

$$\begin{aligned}\text{Precision } P &= \frac{TP}{TP + FP} \\ \text{Recall } R &= \frac{TP}{TP + FN} \\ \text{Avg. Prec. } AP &= \frac{1}{11} \sum_{R \in \{0, \dots, 1\}} \max_{R' \geq R} P(R')\end{aligned}$$

Sliding-Window Object Detection

Sliding Window Object Detection



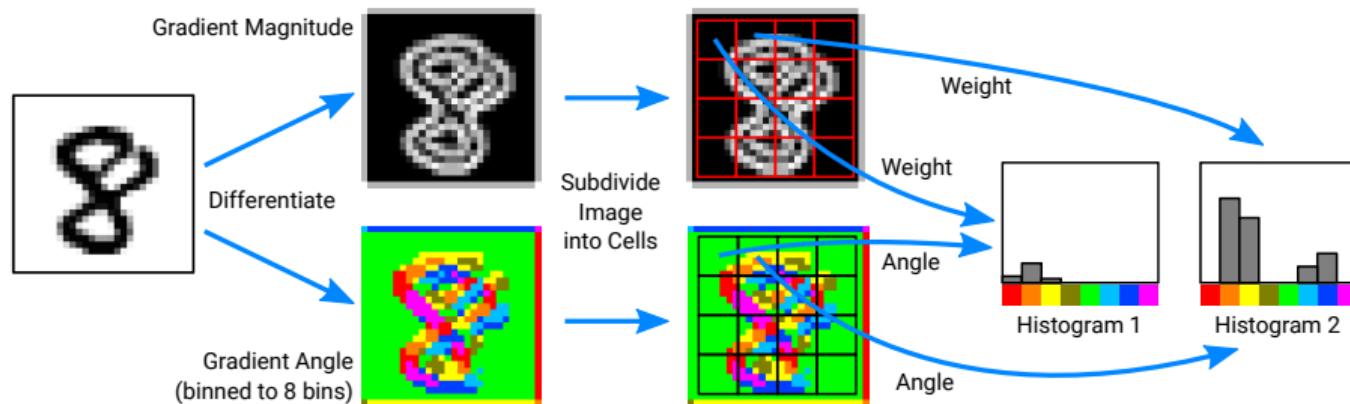
Sliding Window Detection:

- ▶ Run **sliding window** of fixed size over the image; extract features for each window
- ▶ **Classify each crop** (object vs. background) using SVM, random forest, boosting, ..
- ▶ Iterate over multiple aspect ratios/scales and perform **non-maxima suppression**

Histograms of Oriented Gradients

Which feature space to use?

- ▶ Simplest: RGB pixel space \Rightarrow neither viewpoint nor illumination invariant
- ▶ Better: Histograms of Oriented Gradients (HoG) (defacto standard for >10 years)
 - ▶ Idea: represent patches with histograms (gradient angles weighted by magnitude)

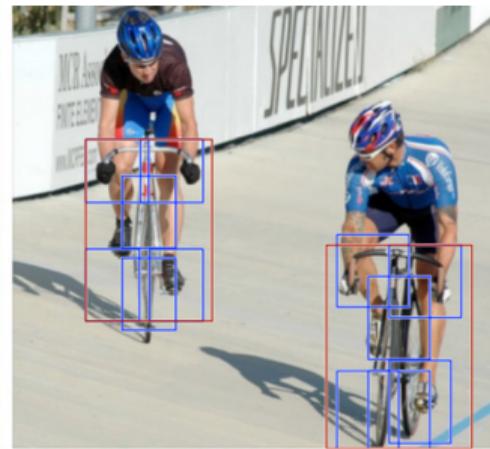
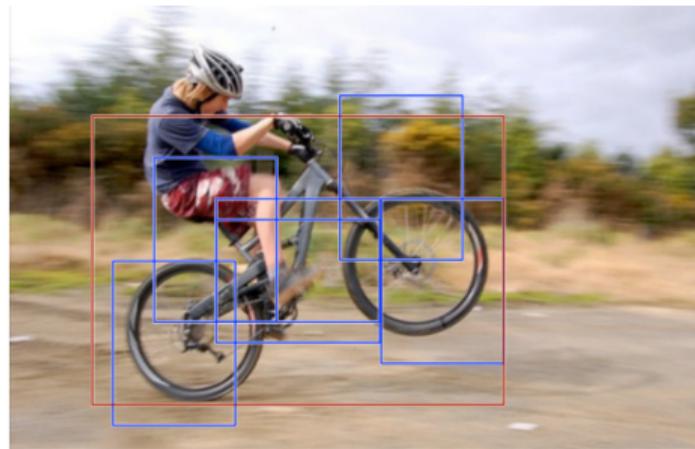
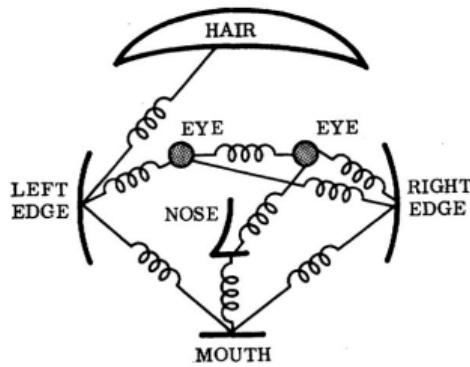


- ▶ HoG is invariant to small deformations (translation, scale, rotation, perspective)

Part Based Models

Part-based Models:

- Idea: Model object based on its parts, **model distribution of part configurations**
- Allows for even more invariance (non-rigid deformations etc.)
- However: **slower inference** and not much gain wrt. multi-view HoG models

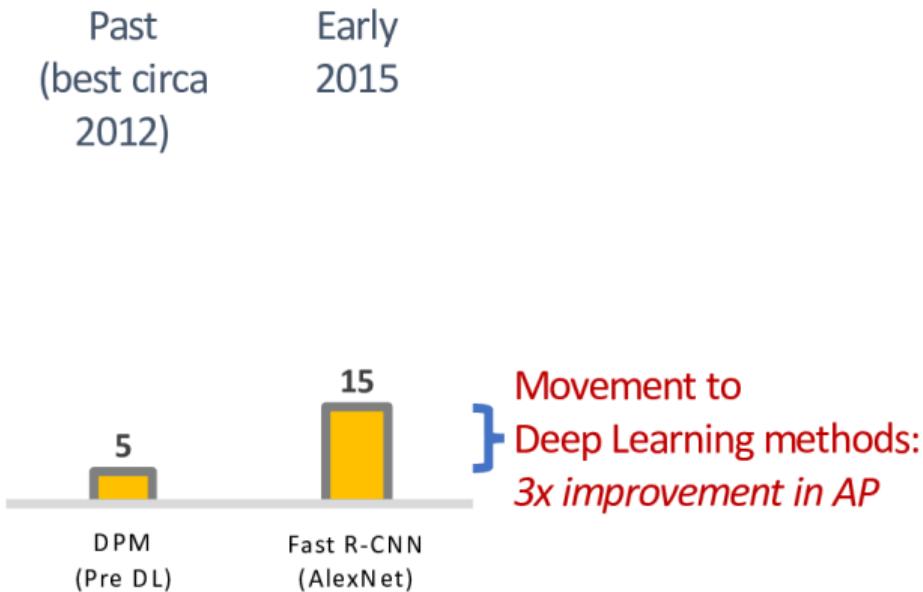


Recognition - A Success Story?

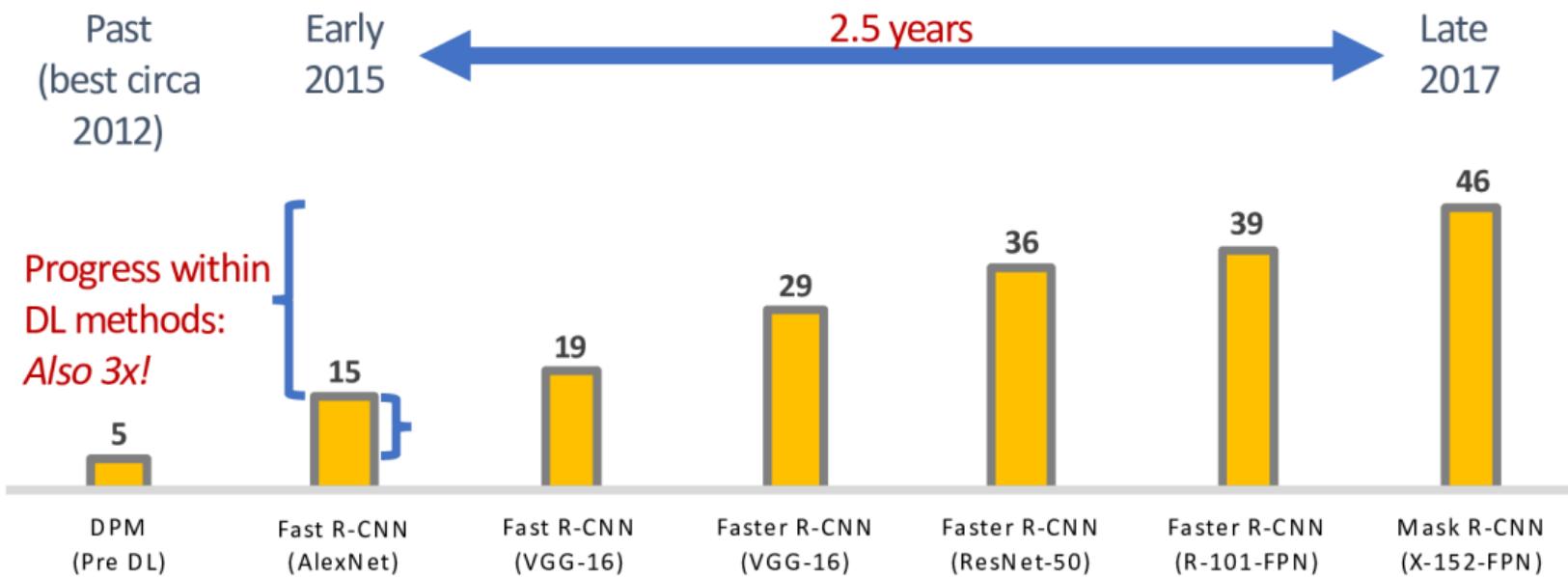
Classical approaches **haven't lead to major breakthroughs**: only for specific non-safety-critical tasks (e.g., face detection), they have seen great success. **Why?**

- ▶ Computer vision features are very difficult to hand-engineer
- ▶ It is unclear how a good representation should look like
- ▶ Furthermore, sliding window approaches are slow in practice
- ▶ Inference in complicated part-based models is even slower
- ▶ Computation is often not efficiently re-used
- ▶ But: deep learning has changed this ..

Hand-crafted Representations vs. Learned Representations



Hand-crafted Representations vs. Learned Representations



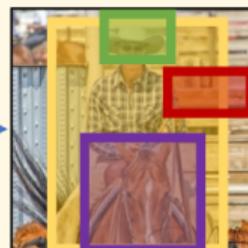
Object Detection with Deep Neural Networks

Slide Credits: Ross Girshick

R-CNN: Region-based Convolutional Neural Network

Per-image computation

$I:$

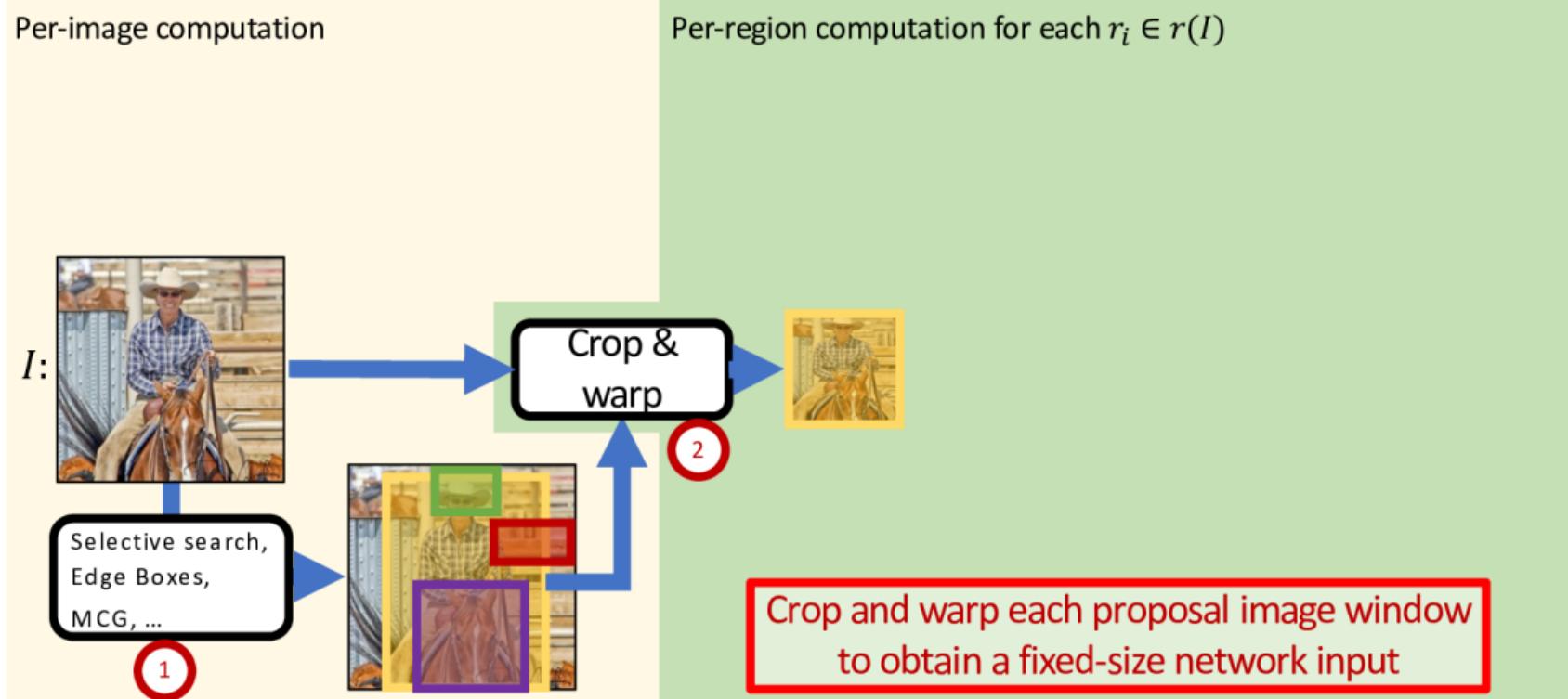


Selective search,
Edge Boxes,
MCG, ...

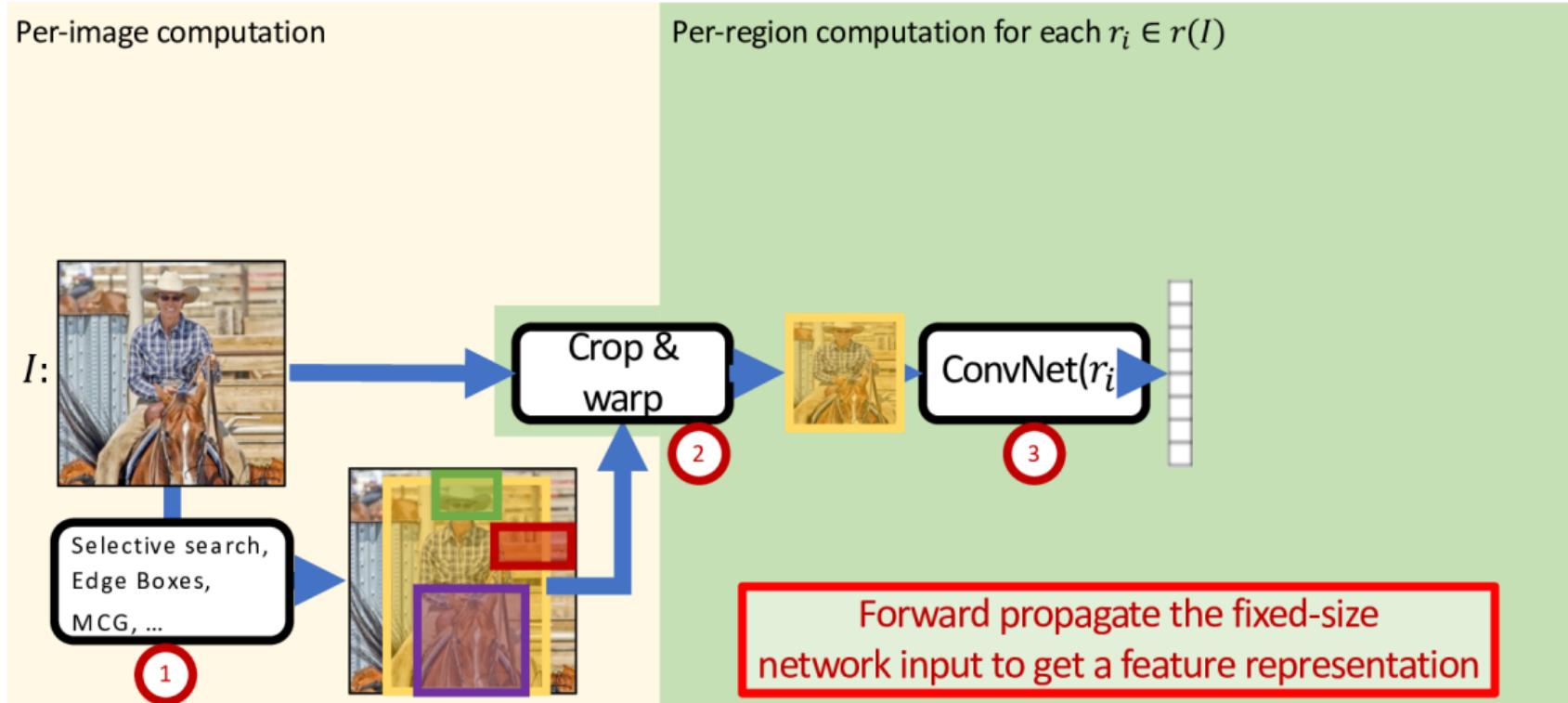
1

Use an off-the-shelf region/object/detection
proposal algorithm (~2k proposals per image)

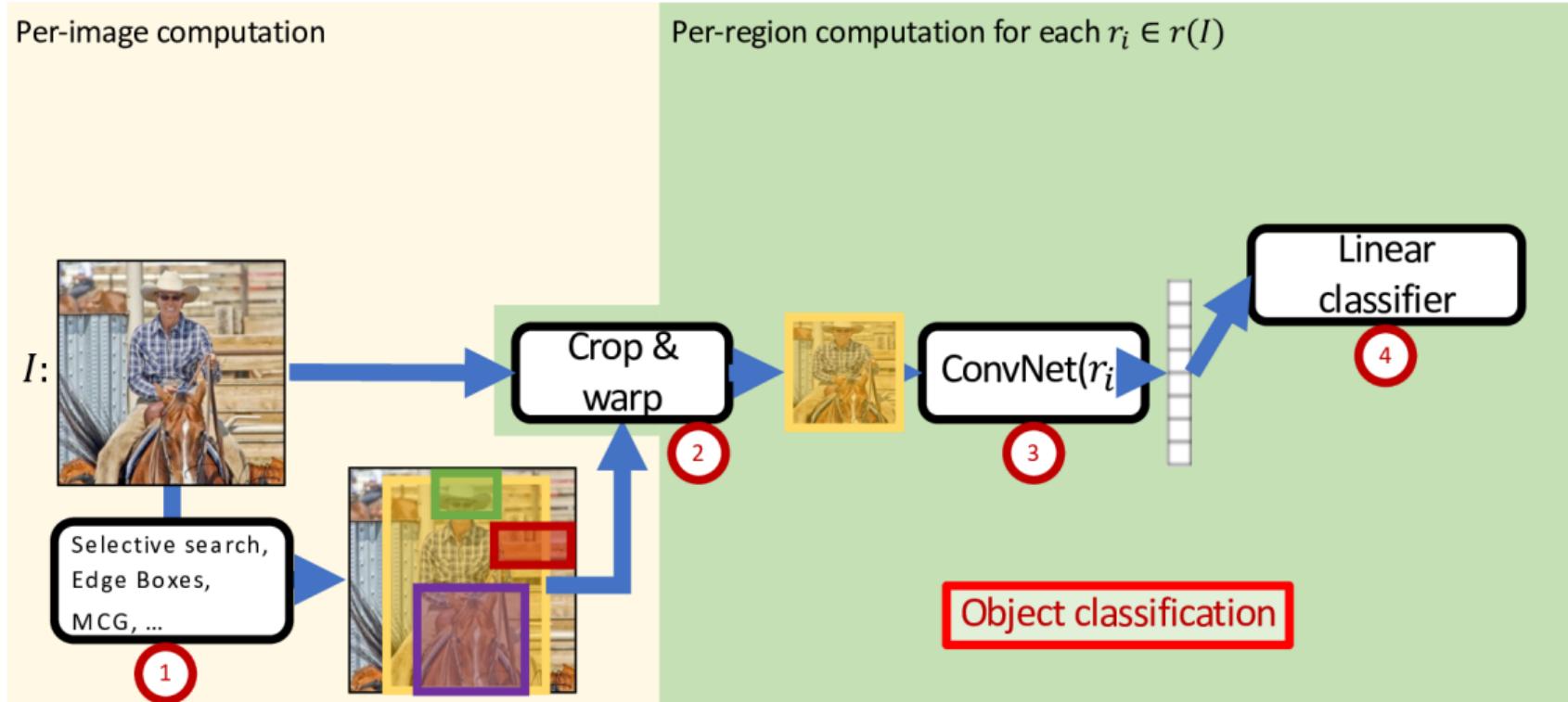
R-CNN: Region-based Convolutional Neural Network



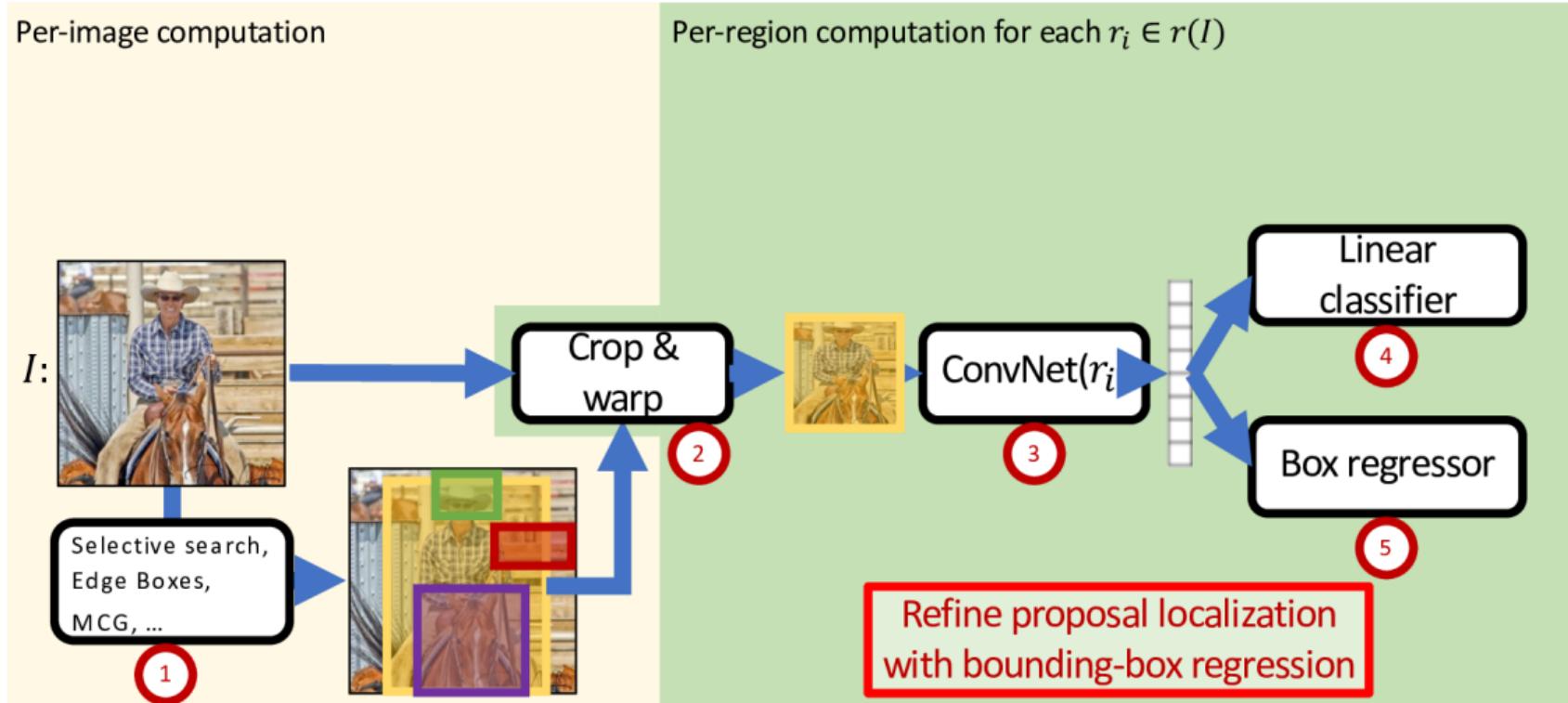
R-CNN: Region-based Convolutional Neural Network



R-CNN: Region-based Convolutional Neural Network

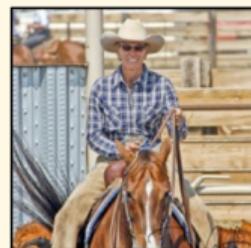


R-CNN: Region-based Convolutional Neural Network



Generalized Framework

Per-image computation



I :

Per-region computation for each $r_i \in r(I)$

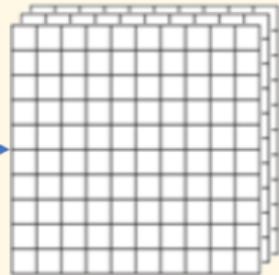
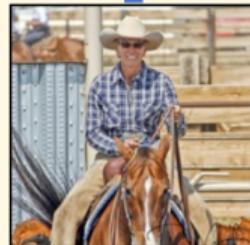
Input image
per-image operations | per-region operations

Generalized Framework

Per-image computation

$$f_I = f(I)$$

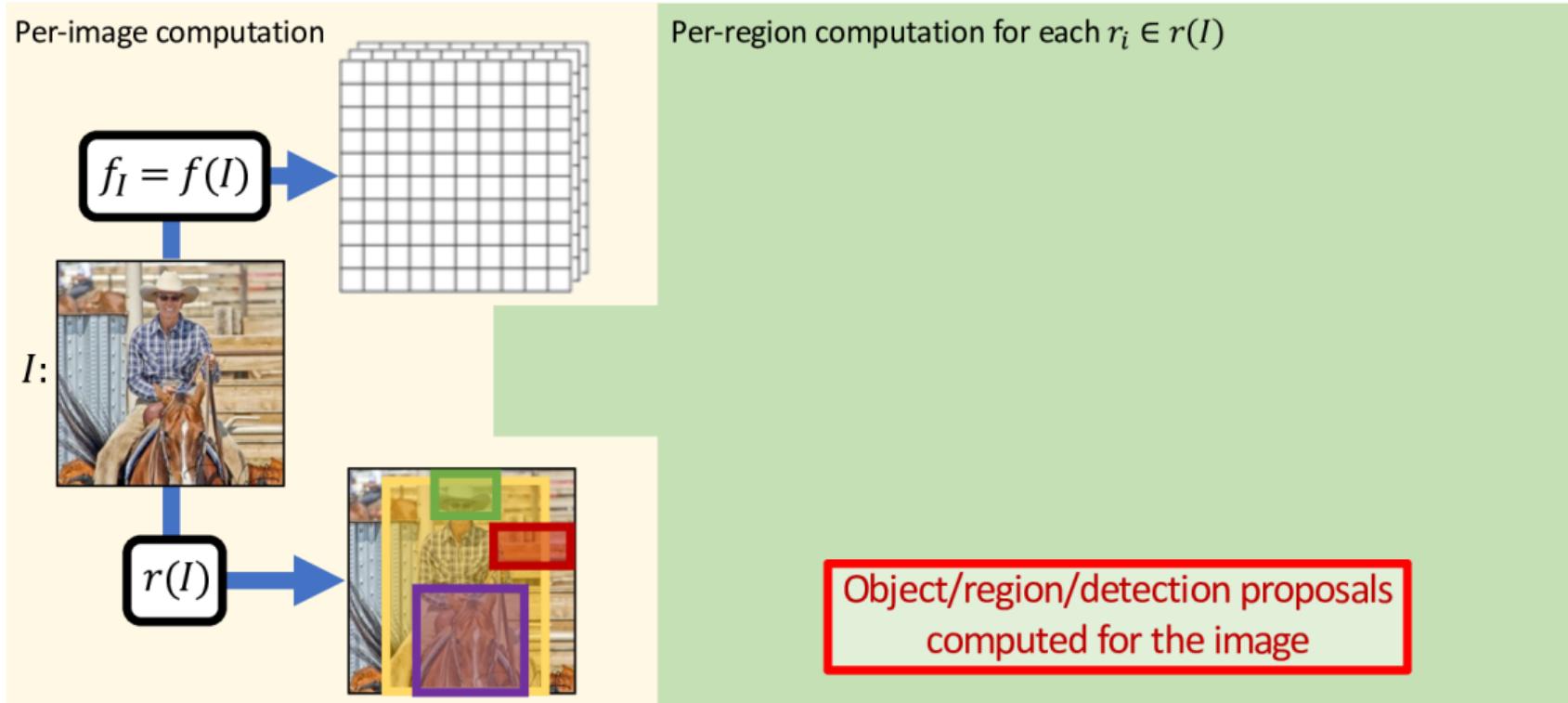
I :



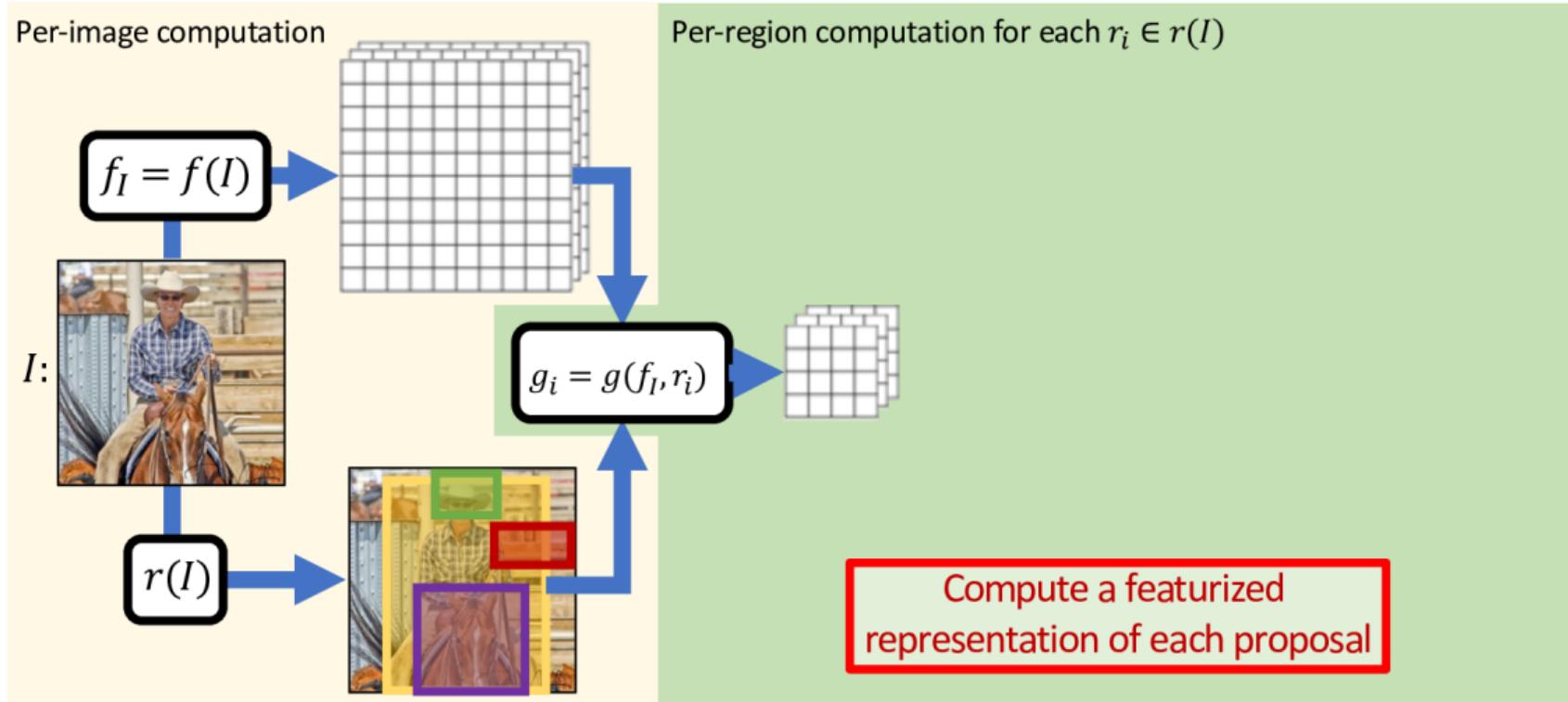
Per-region computation for each $r_i \in r(I)$

Transformation of the input image
into a featurized representation

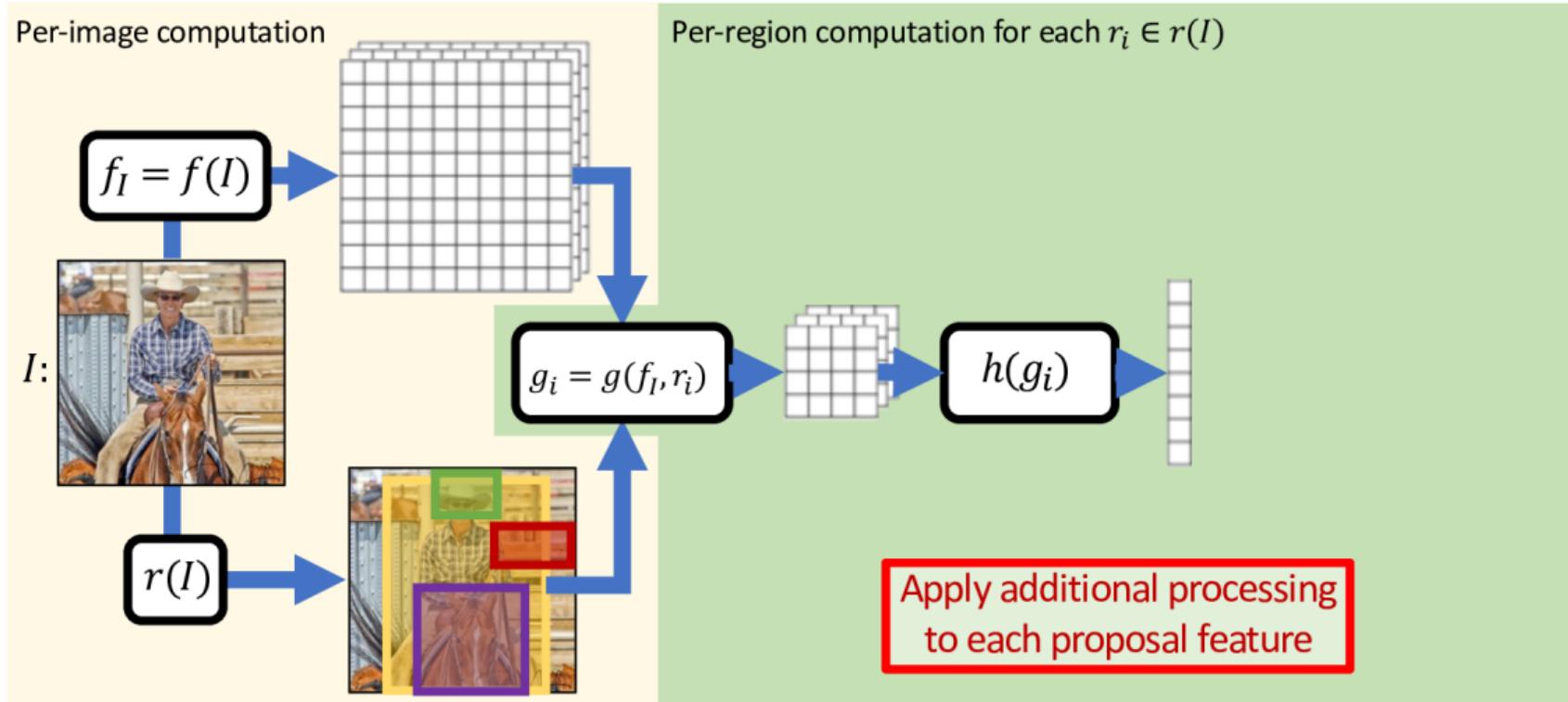
Generalized Framework



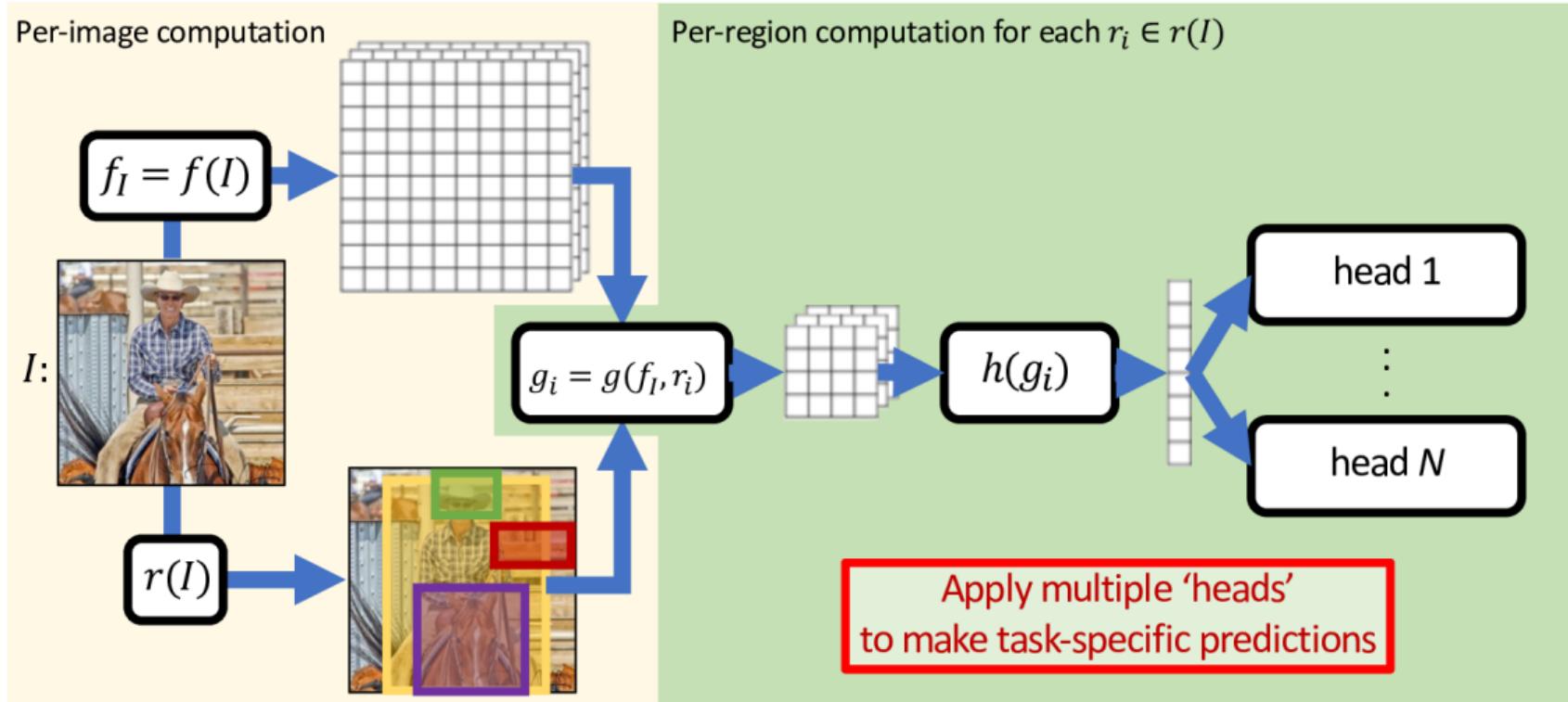
Generalized Framework



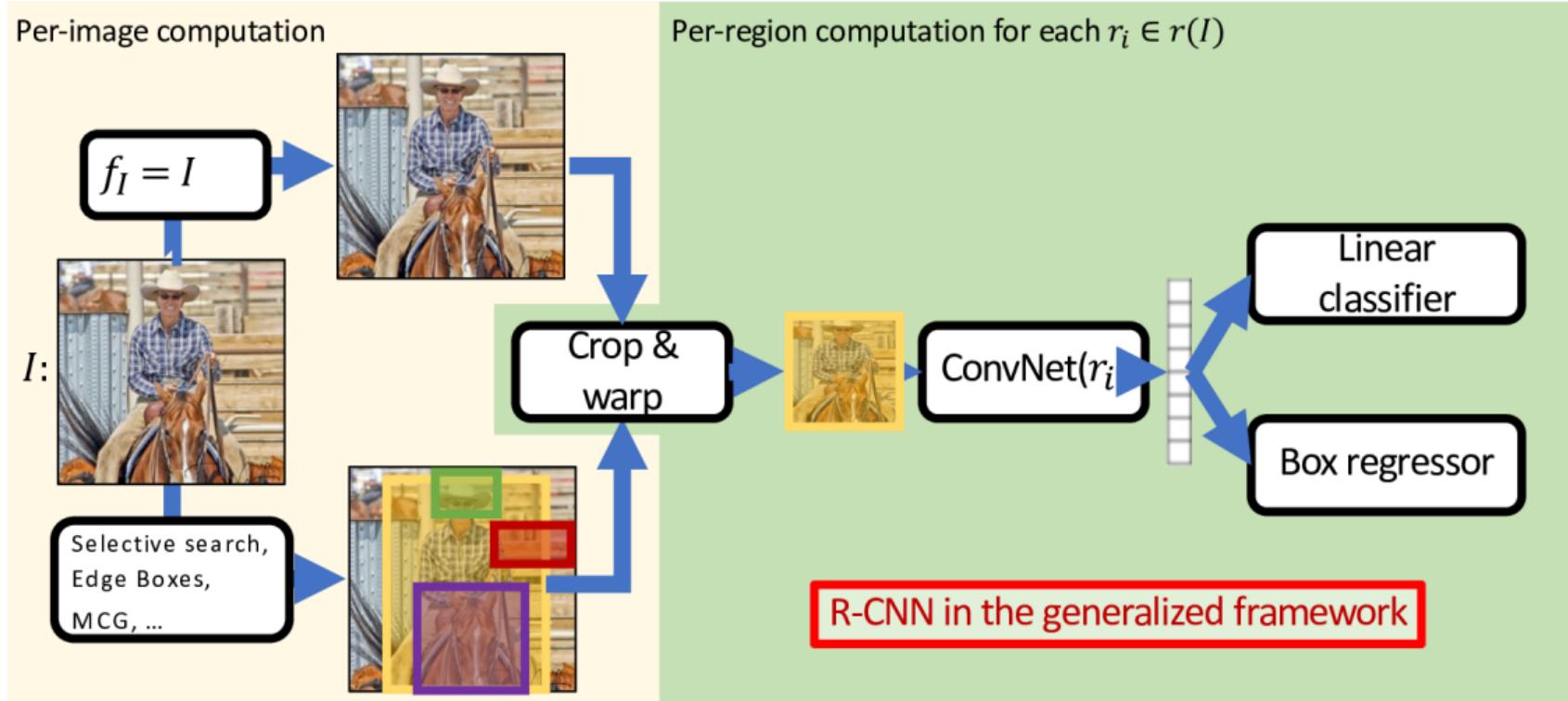
Generalized Framework



Generalized Framework



R-CNN in Generalized Framework

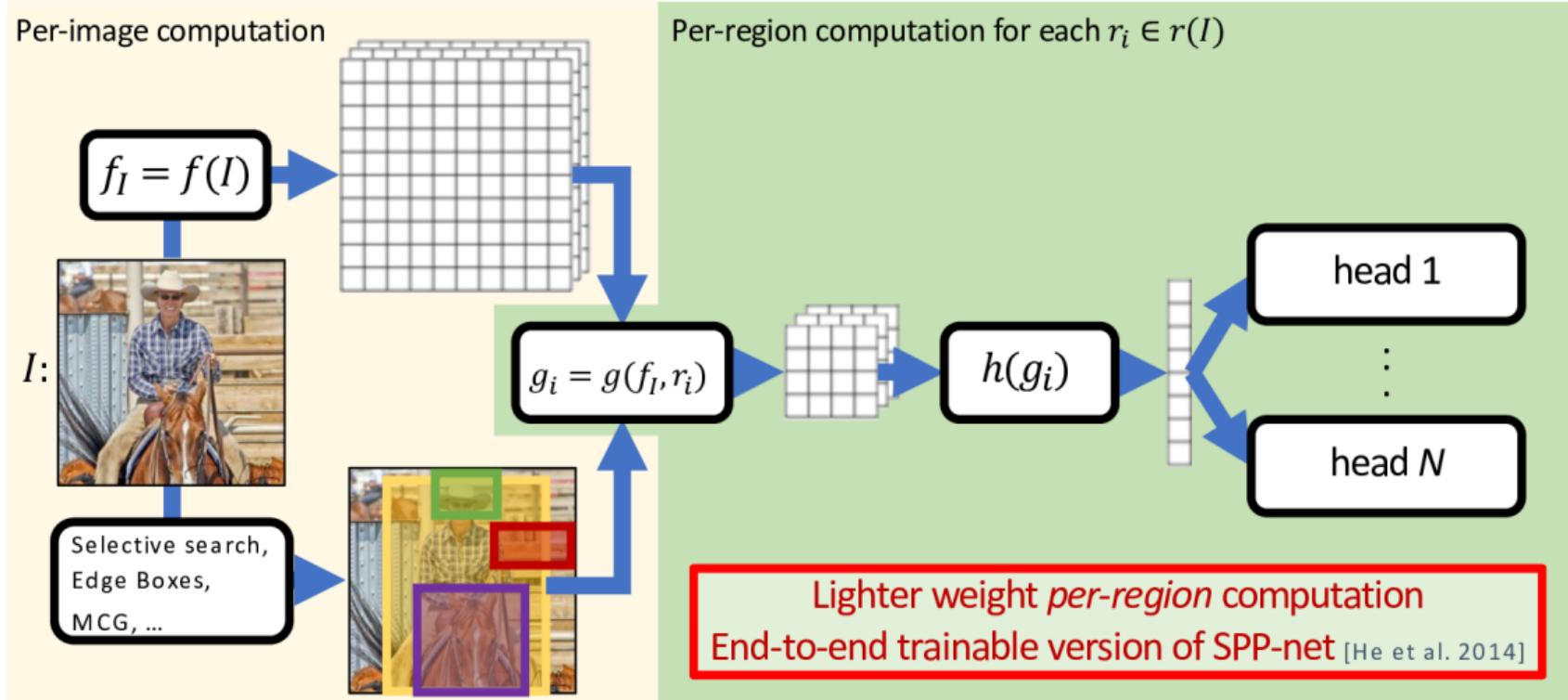


R-CNN in Generalized Framework

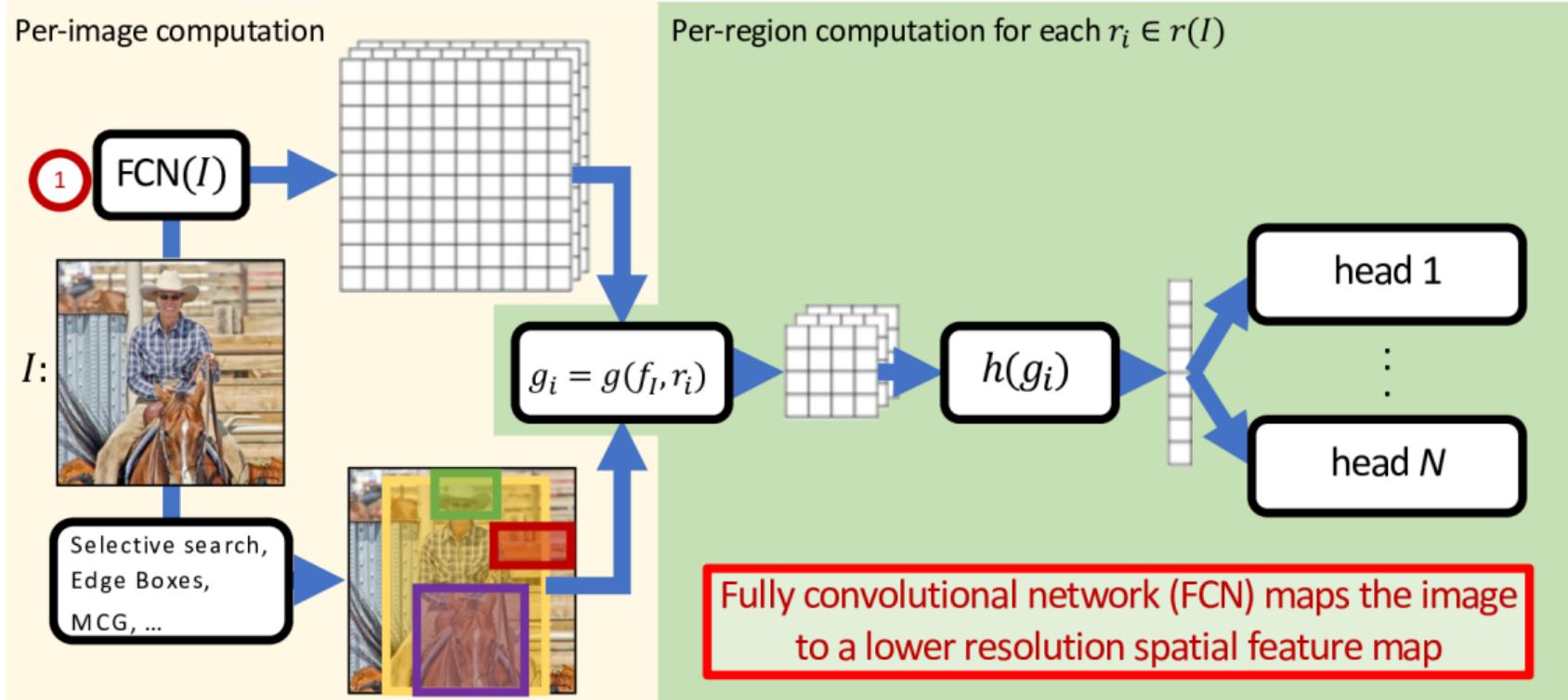
What is the problem with R-CNN?

- ▶ Heavy per-region computation (e.g., 2000 full network evaluations)
- ▶ No computation/feature sharing
- ▶ Slow region proposal method adds to runtime
- ▶ Generic region proposal techniques have limited recall

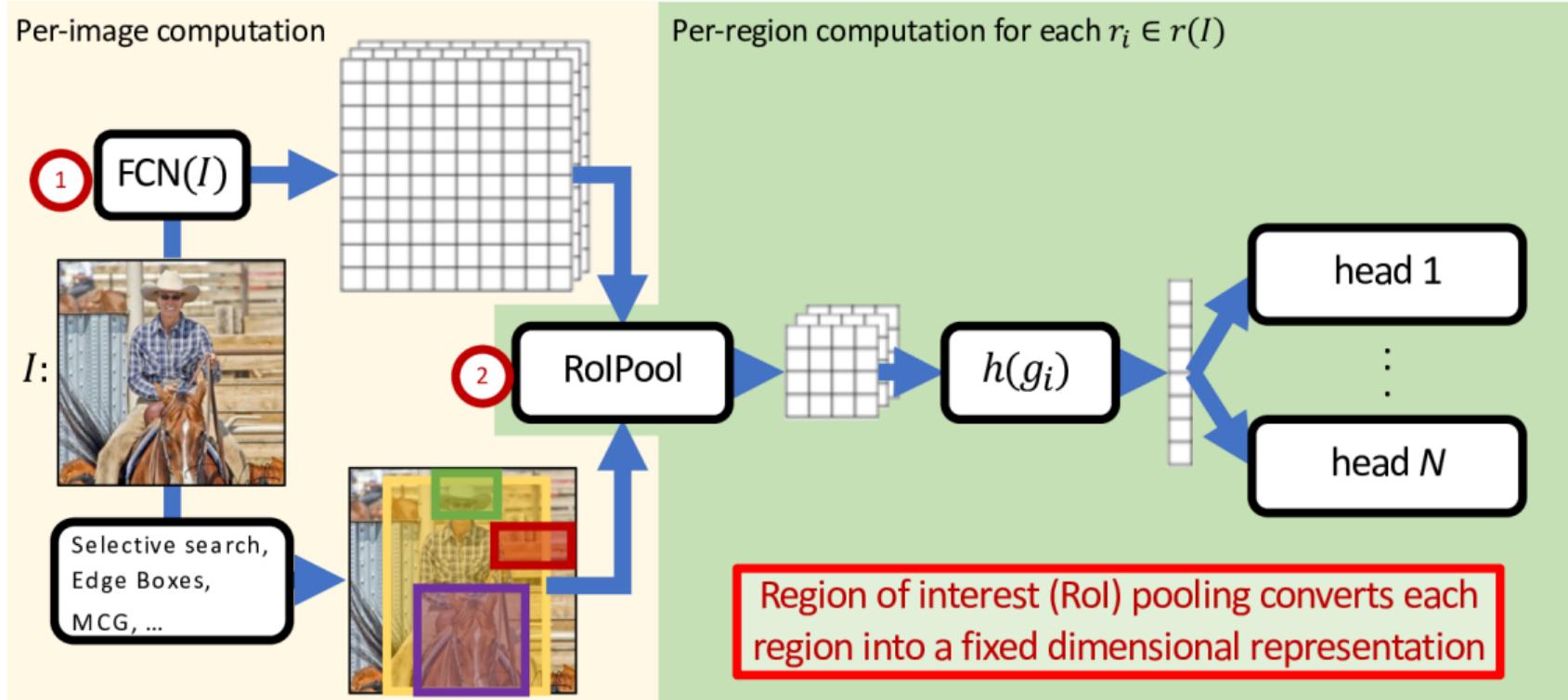
Fast R-CNN



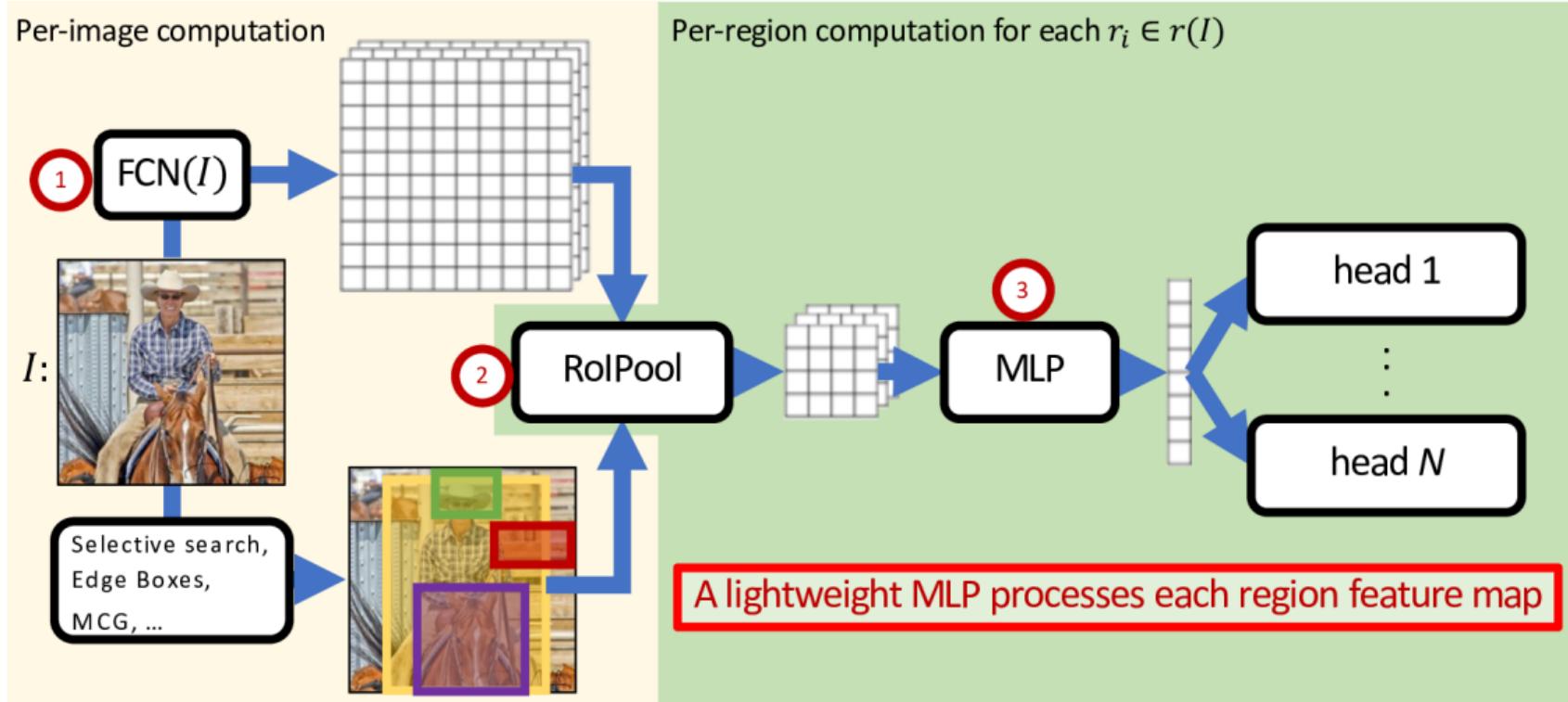
Fast R-CNN



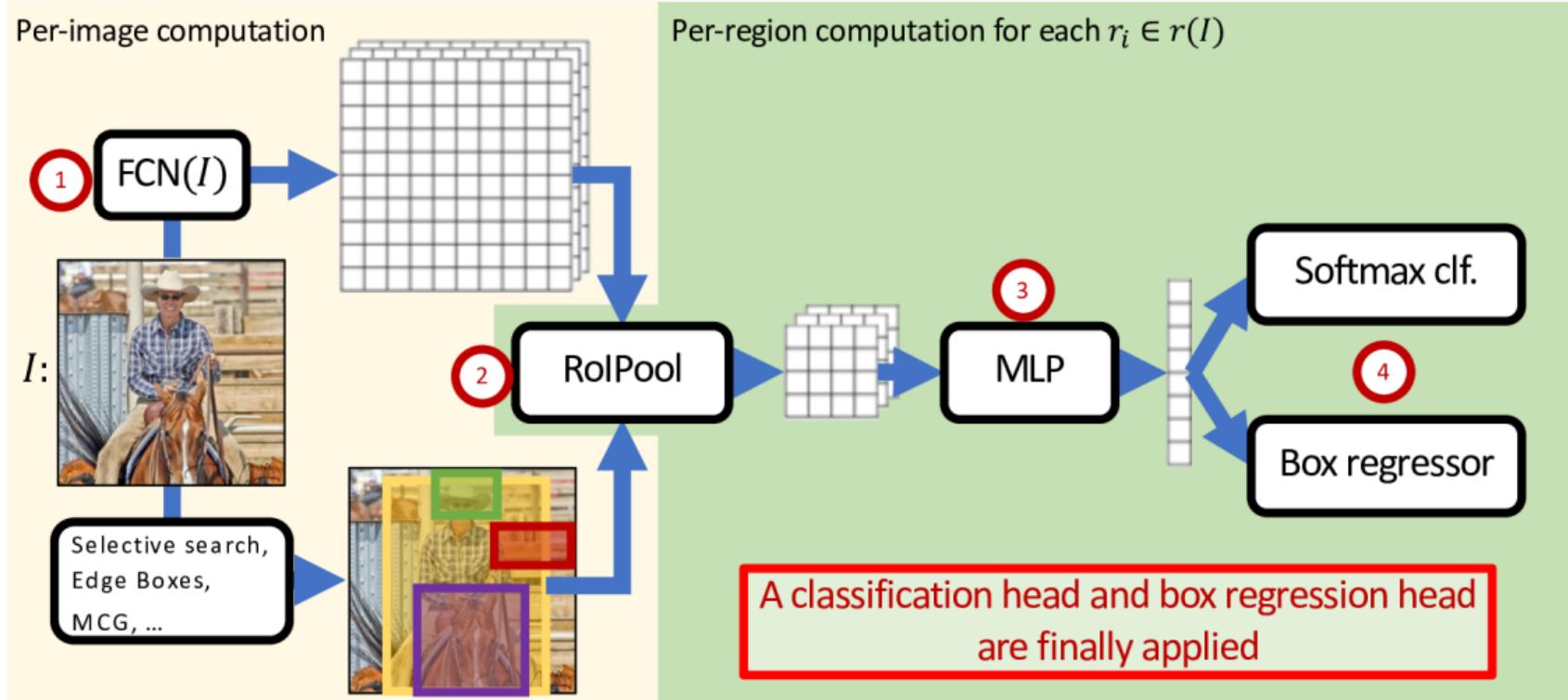
Fast R-CNN



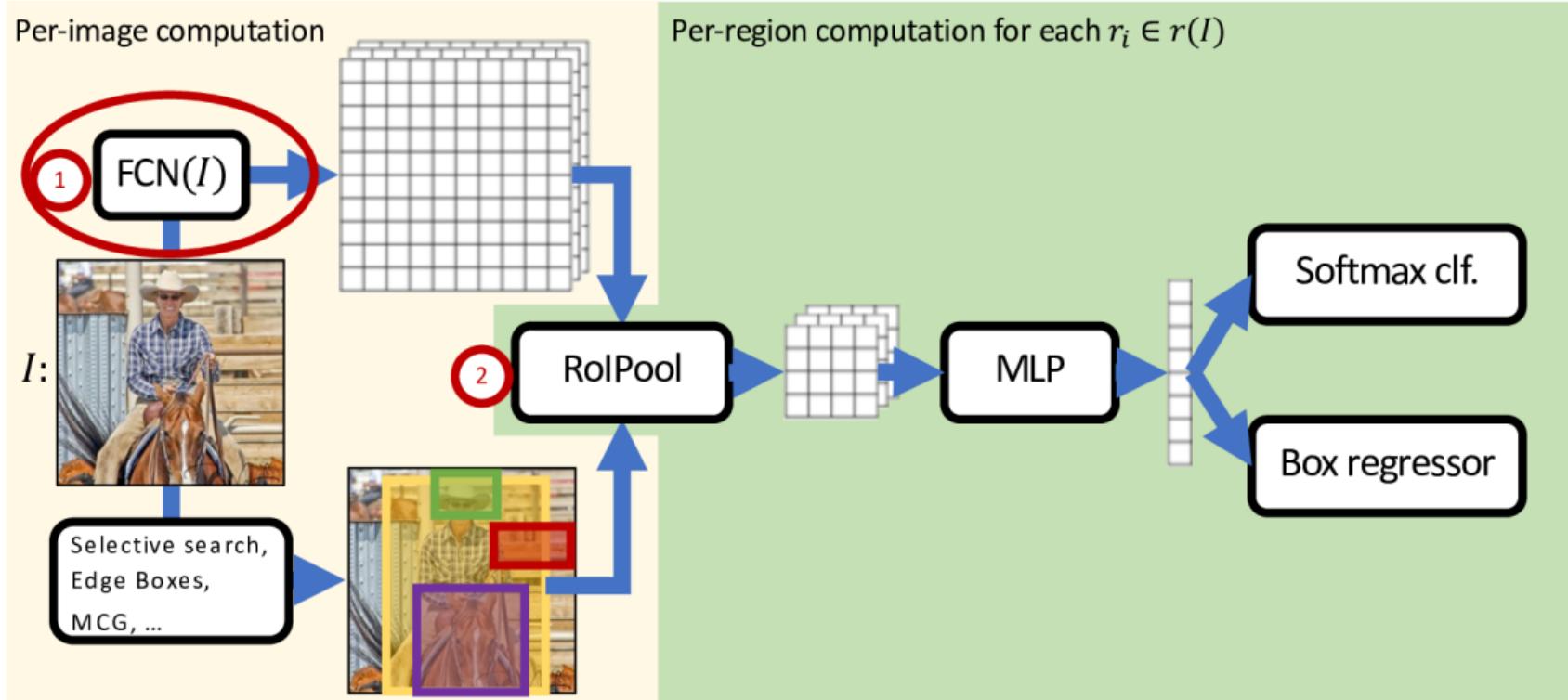
Fast R-CNN



Fast R-CNN



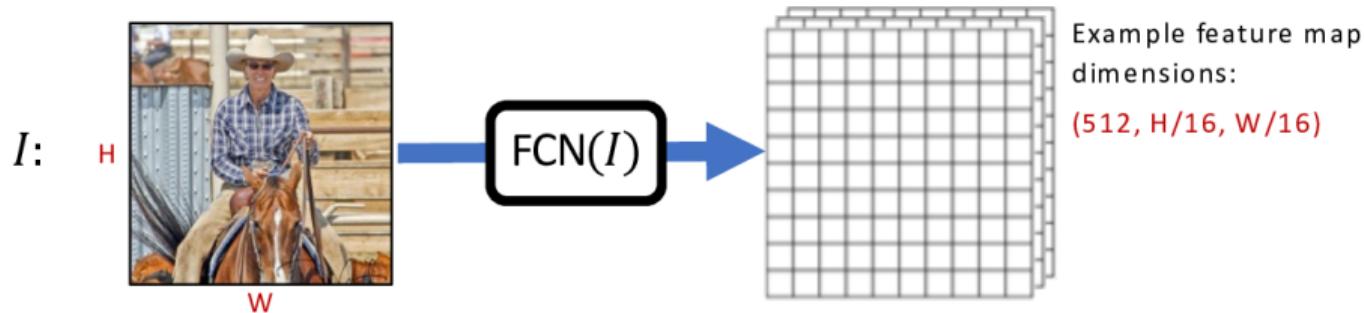
Fast R-CNN



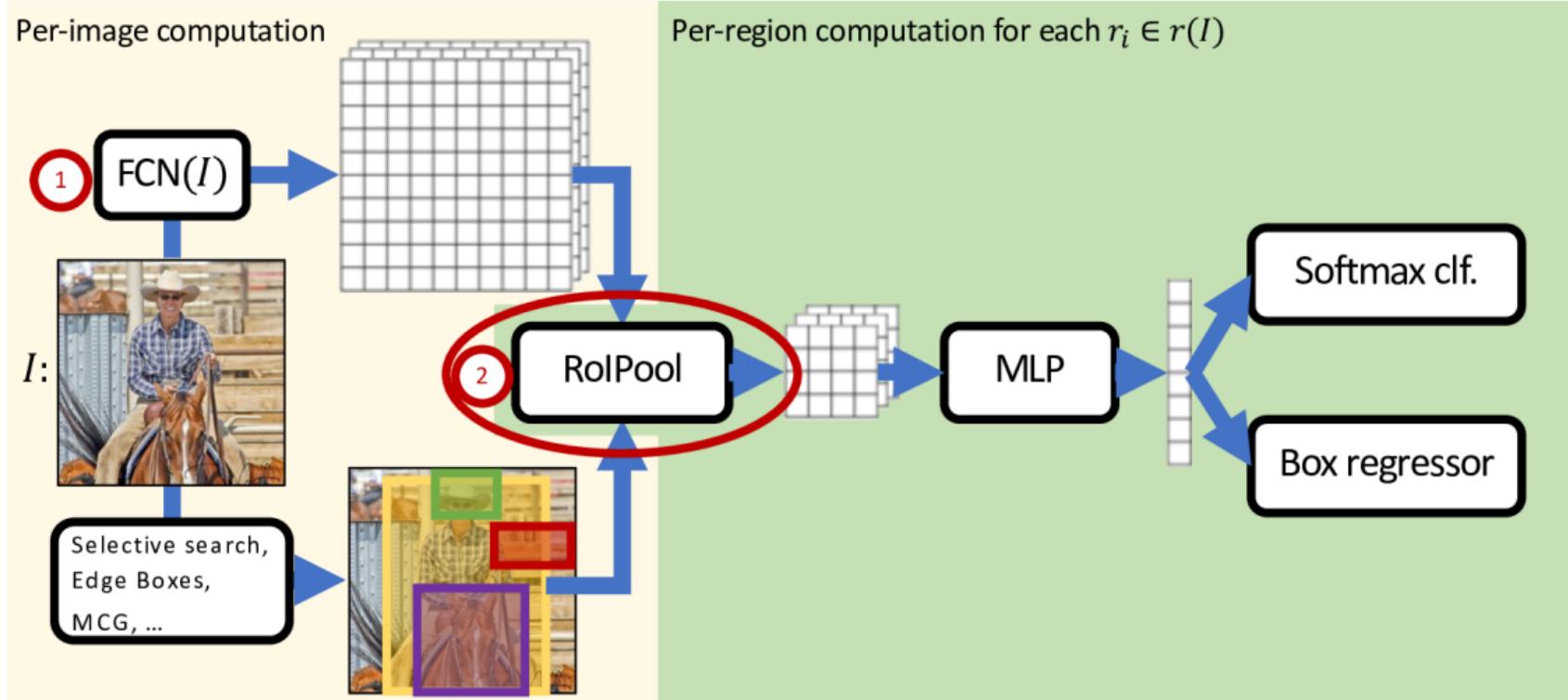
Fast R-CNN

Backbone:

- ▶ Use any standard convolutional network as “backbone” architecture
 - ▶ E.g.: AlexNet, VGG, ResNet, Inception, ResNeXt, DenseNet, ...
- ▶ **Remove global pooling** ⇒ output spatial dims proportional to input spatial dims
- ▶ A good network exploits the strongest recognition backbone (features matter!)

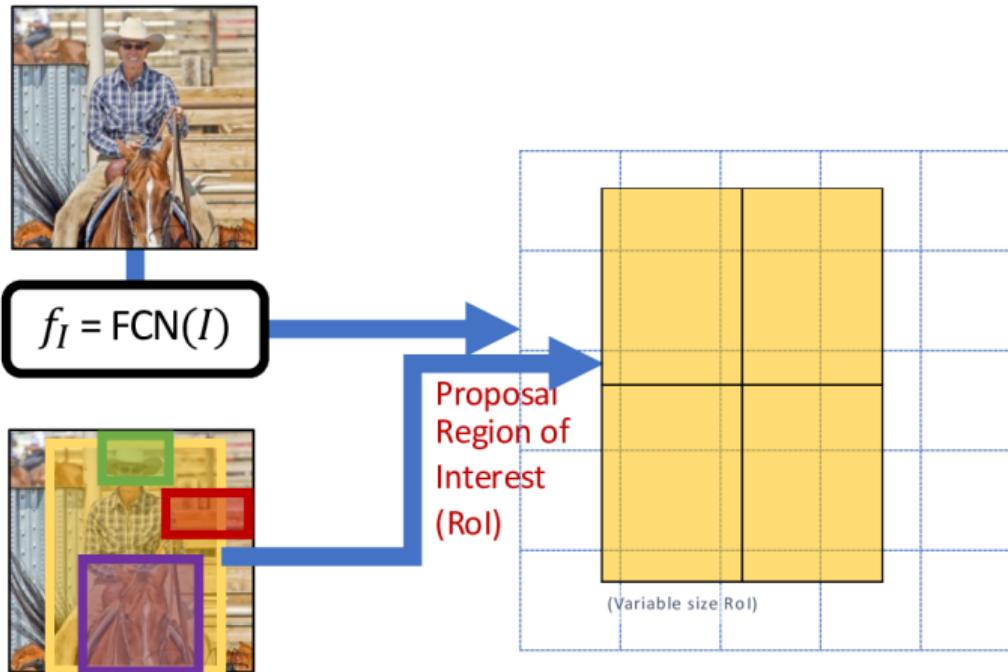


Fast R-CNN



Fast R-CNN

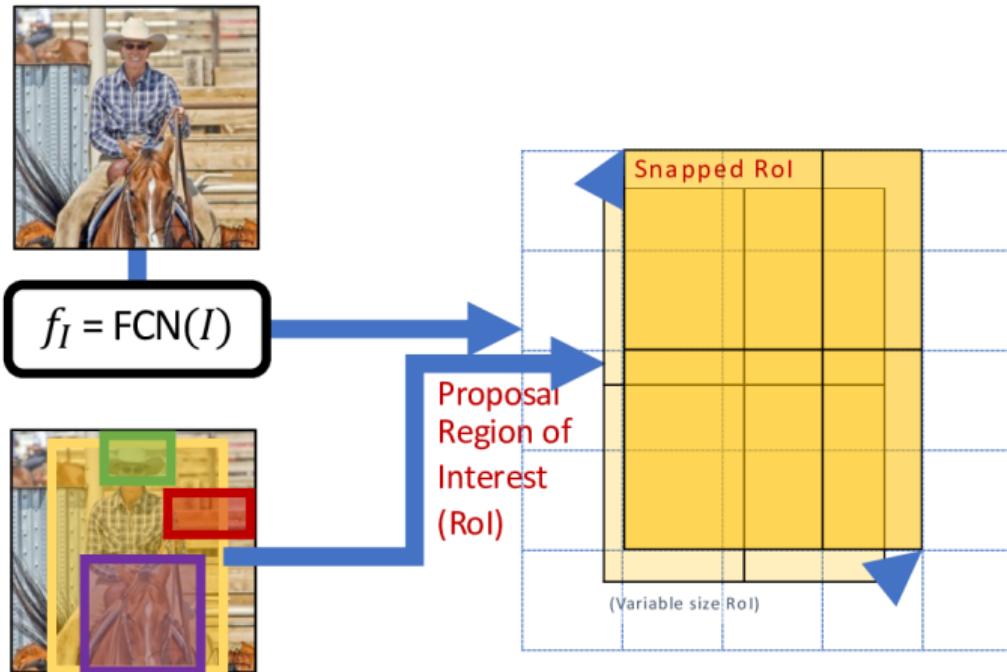
Region-of-Interest (RoI) Pooling on each Proposal:



Key innovation in SPP-net
[He et al. 2014]

Fast R-CNN

Region-of-Interest (RoI) Pooling on each Proposal:



Key innovation in SPP-net
[He et al. 2014]

Fast R-CNN

Region-of-Interest (RoI) Pooling on each Proposal:

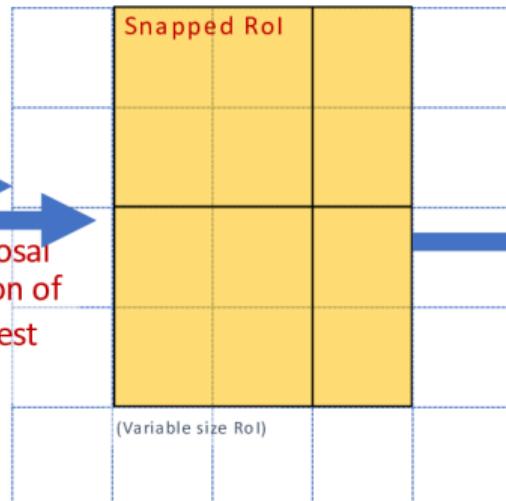


Transform arbitrary size proposal into a fixed-dimensional representation (e.g., 2x2)

$$f_I = \text{FCN}(I)$$



Proposal
Region of
Interest
(RoI)



RoIPool
transform

(Fixed dimensional
representation)

Feature value
is **max** over input
cells



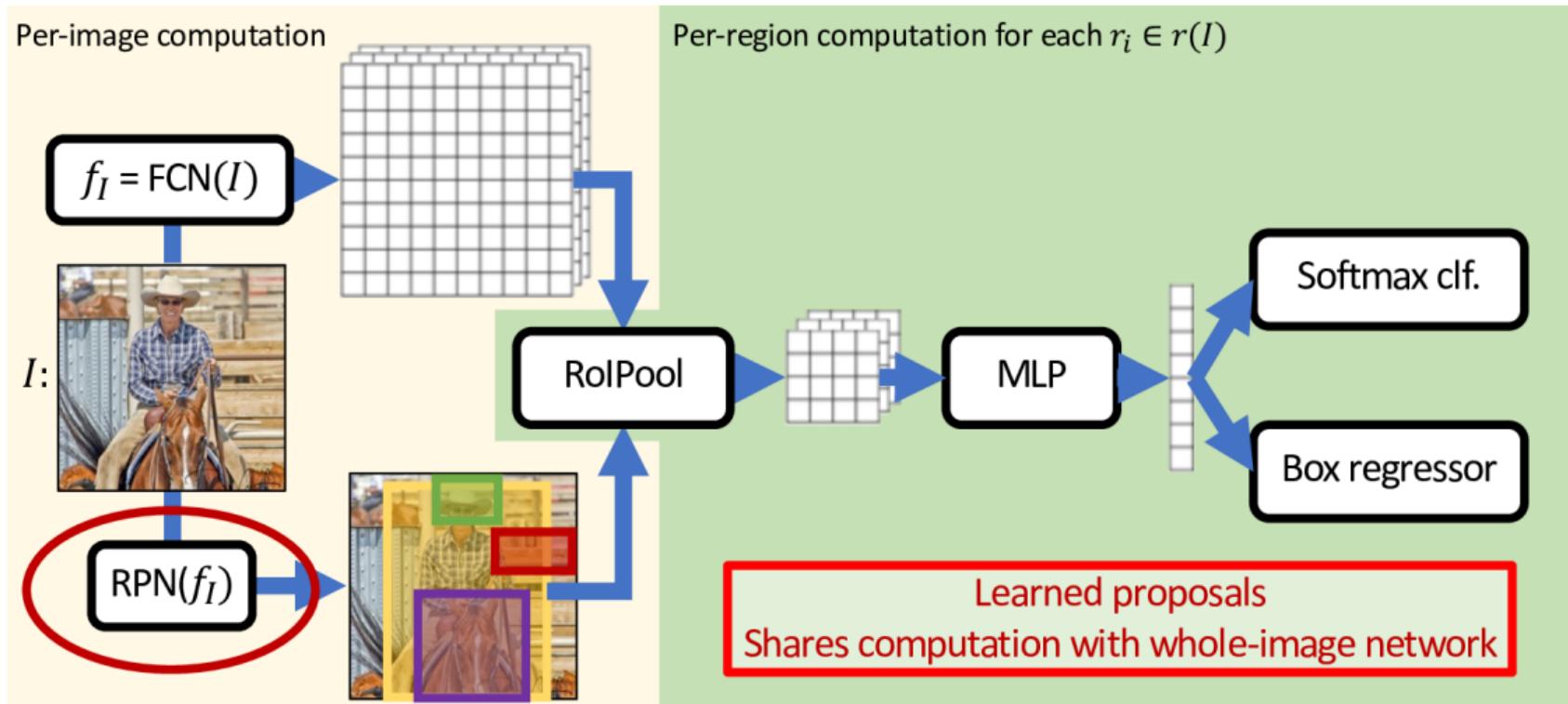
MLP

Fast R-CNN

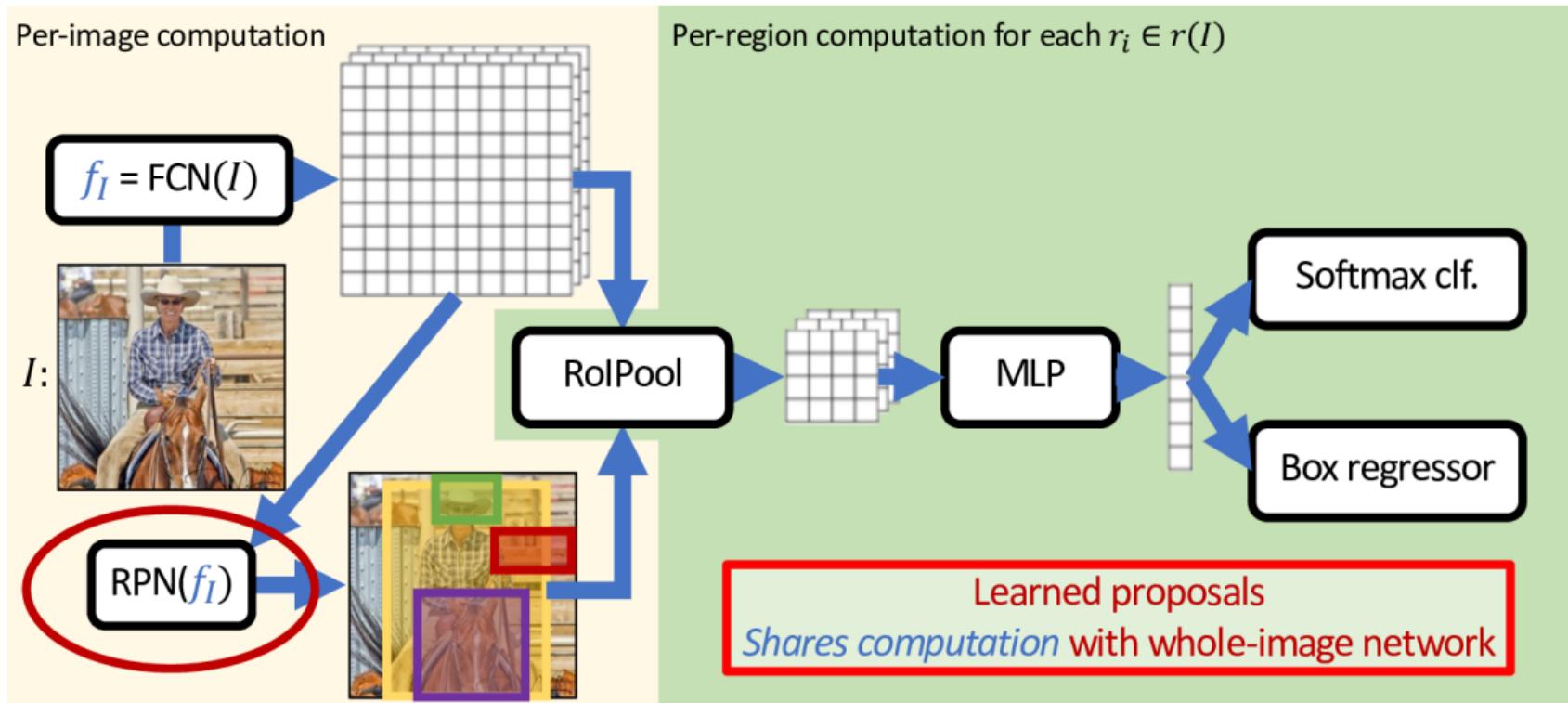
What is the problem with Fast R-CNN?

- ▶ ~~Heavy per-region computation (e.g., 2000 full network evaluations)~~
- ▶ ~~No computation/feature sharing~~
- ▶ Slow region proposal method adds to runtime
- ▶ Generic region proposal techniques have low recall

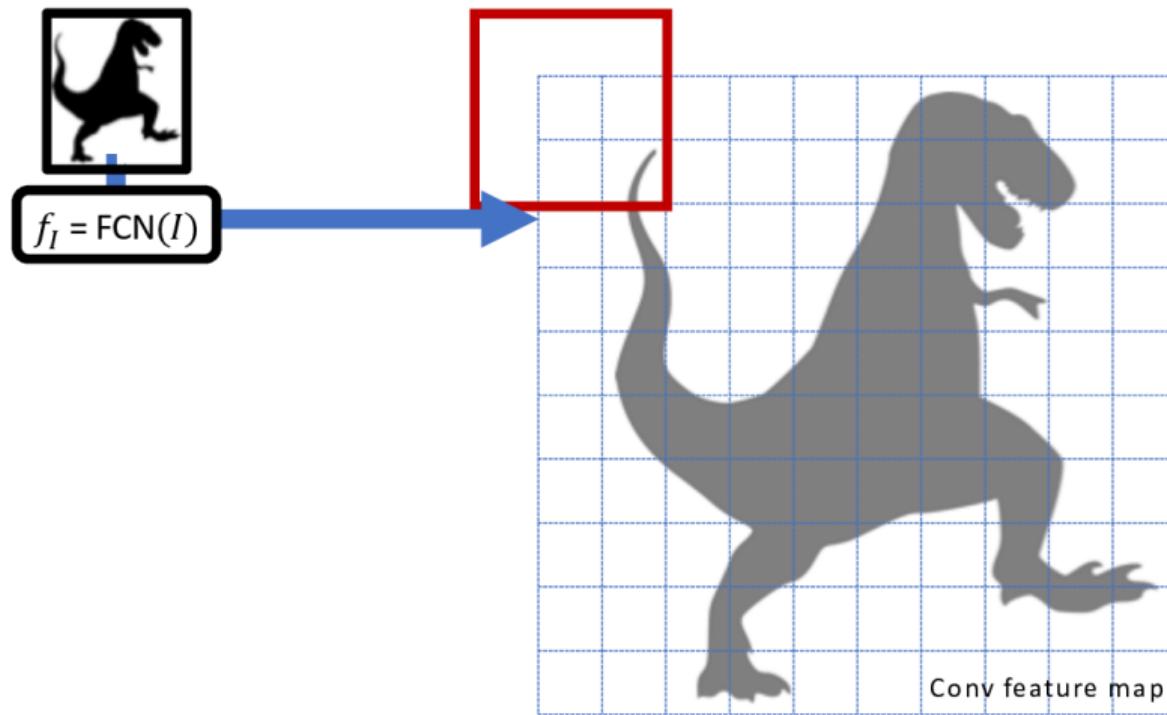
Faster R-CNN



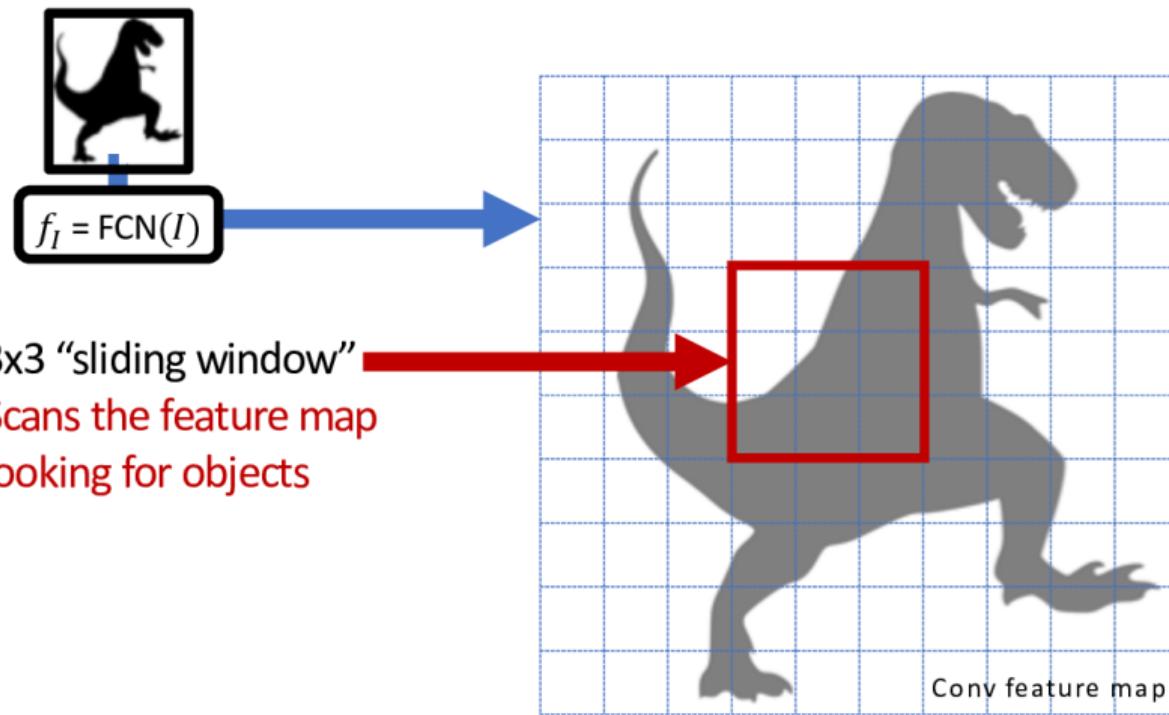
Faster R-CNN



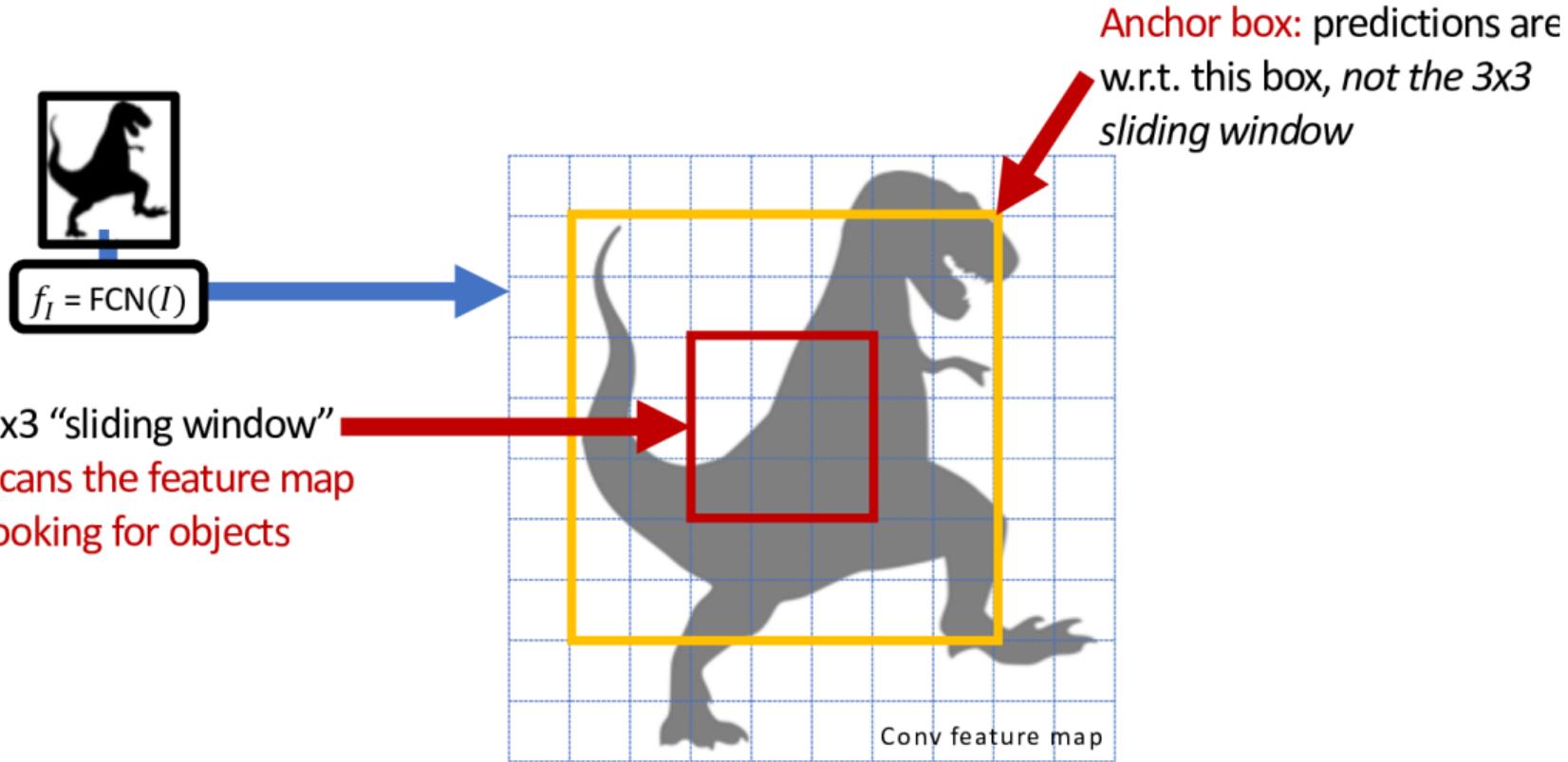
Faster R-CNN: Region Proposal Network (RPN)



Faster R-CNN: Region Proposal Network (RPN)

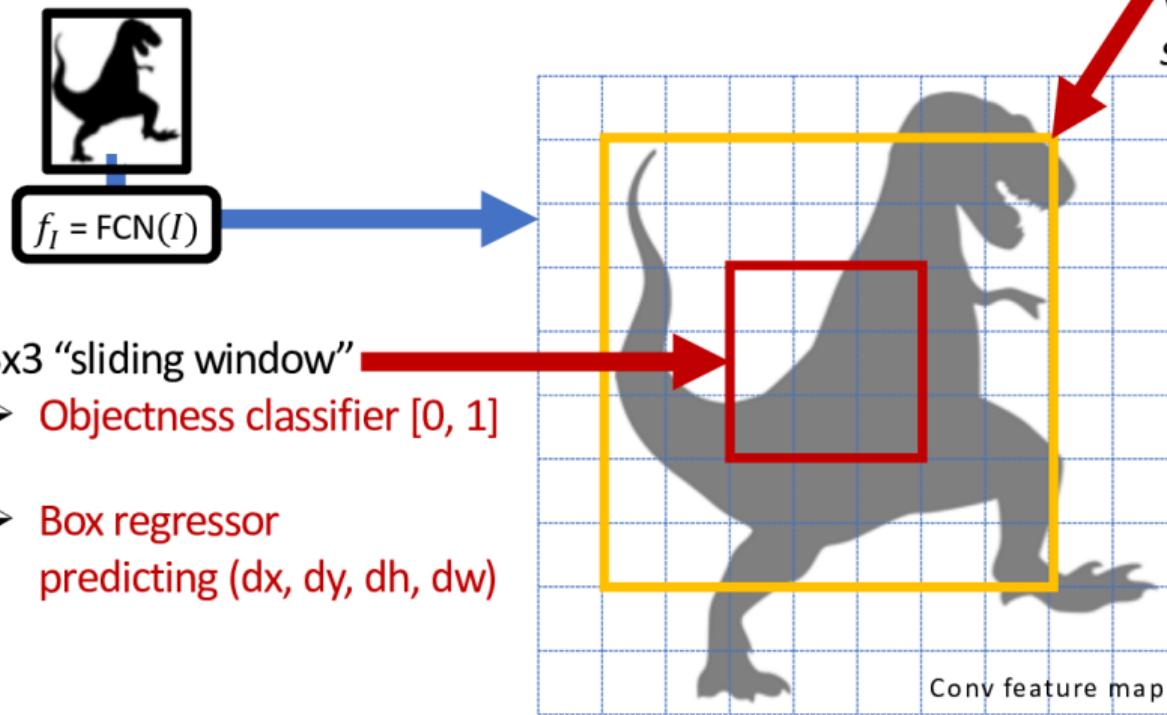


Faster R-CNN: Region Proposal Network (RPN)



Faster R-CNN: Region Proposal Network (RPN)

Anchor box: predictions are w.r.t. this box, *not the 3x3 sliding window*



Faster R-CNN: Region Proposal Network (RPN)

Objectness score



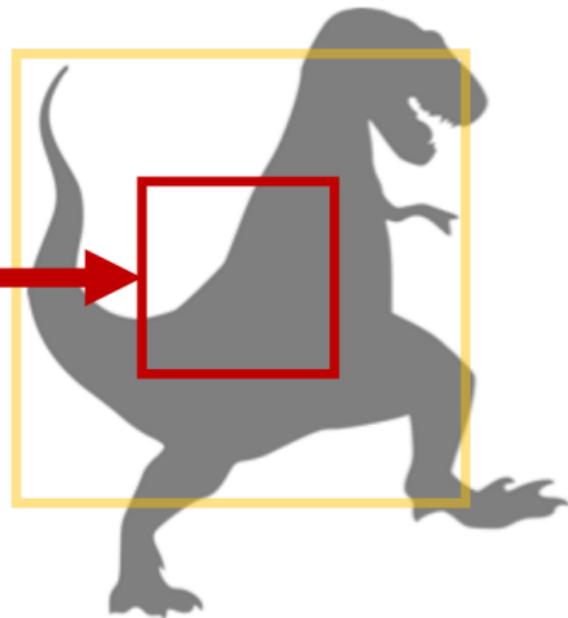
$$P(\text{object}) = 0.94$$

3x3 “sliding window”

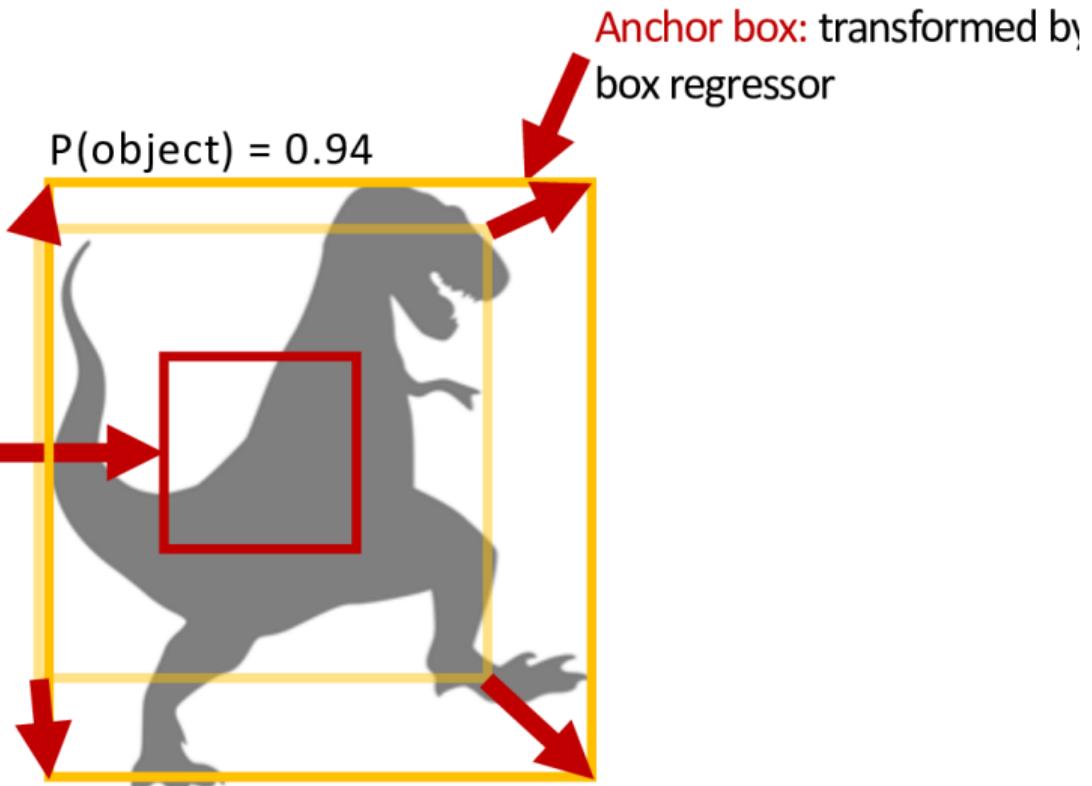
➤ Objectness classifier $[0, 1]$

➤ Box regressor

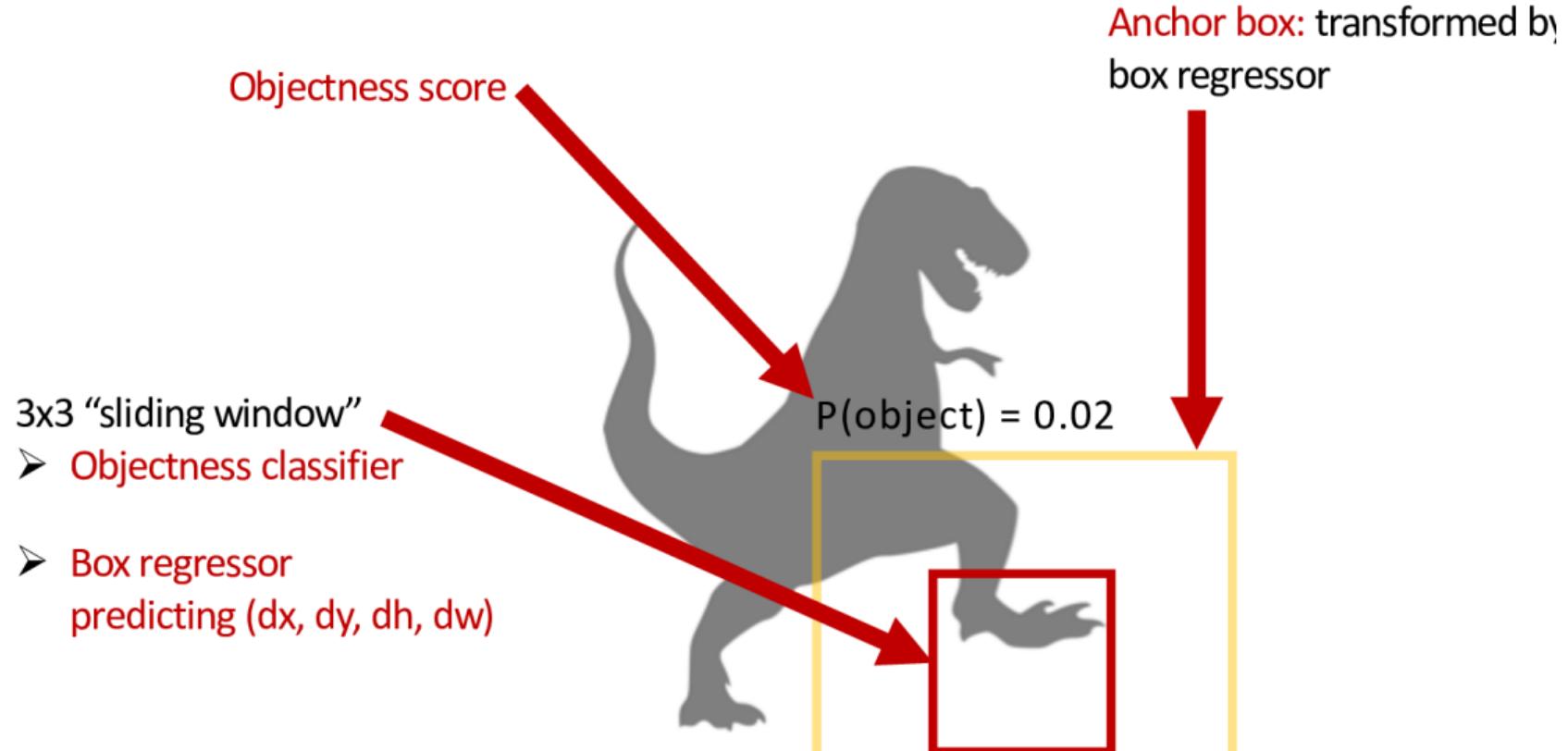
predicting (dx, dy, dh, dw)



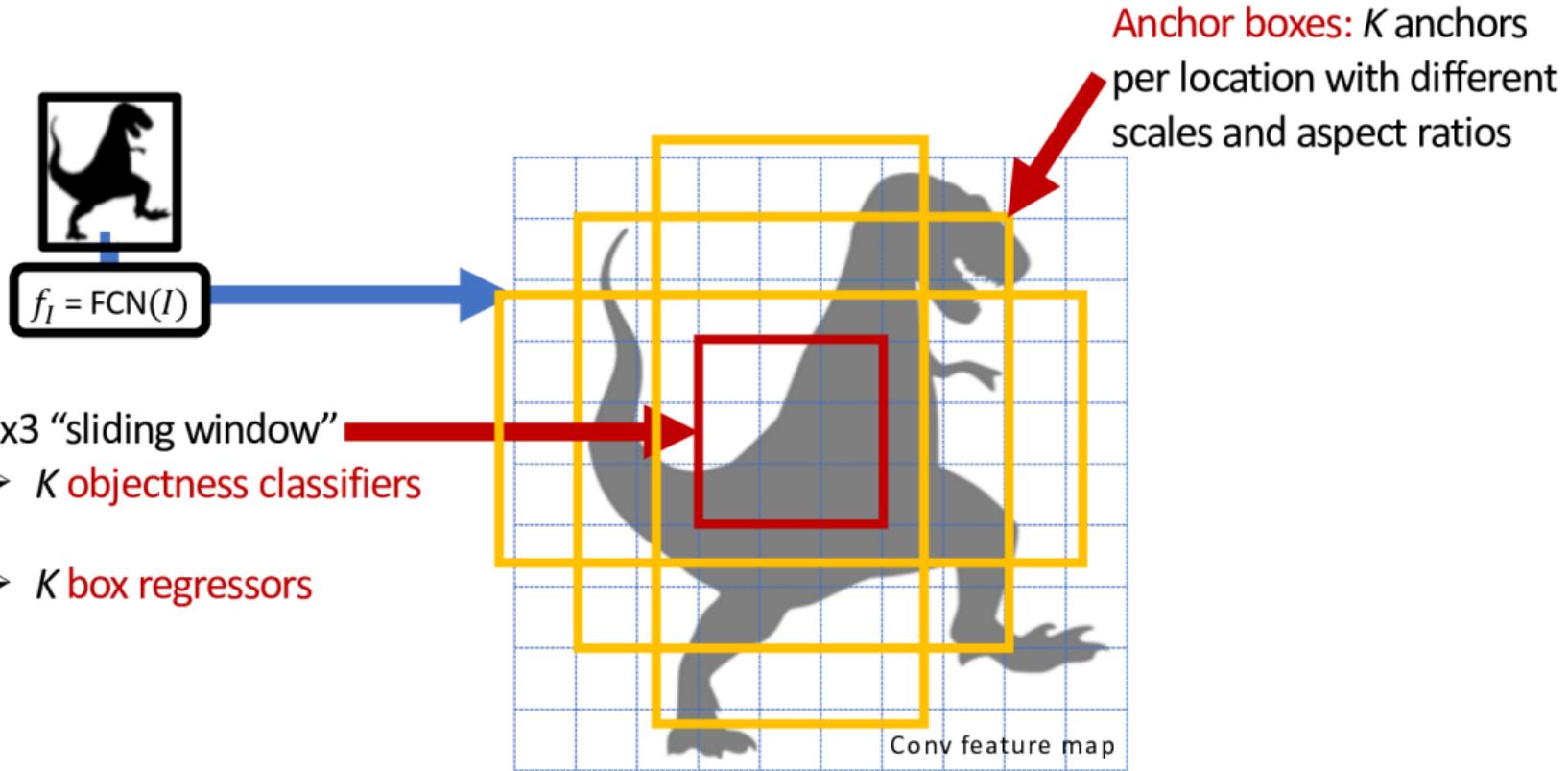
Faster R-CNN: Region Proposal Network (RPN)



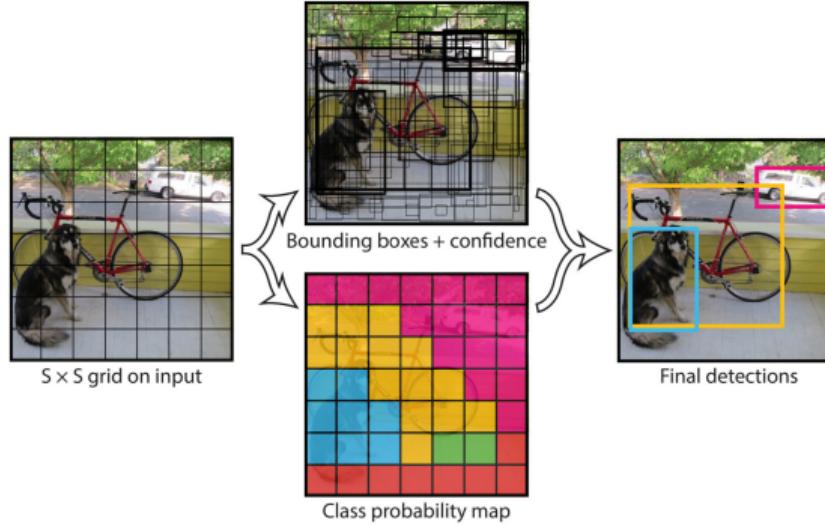
Faster R-CNN: Region Proposal Network (RPN)



Faster R-CNN: Region Proposal Network (RPN)

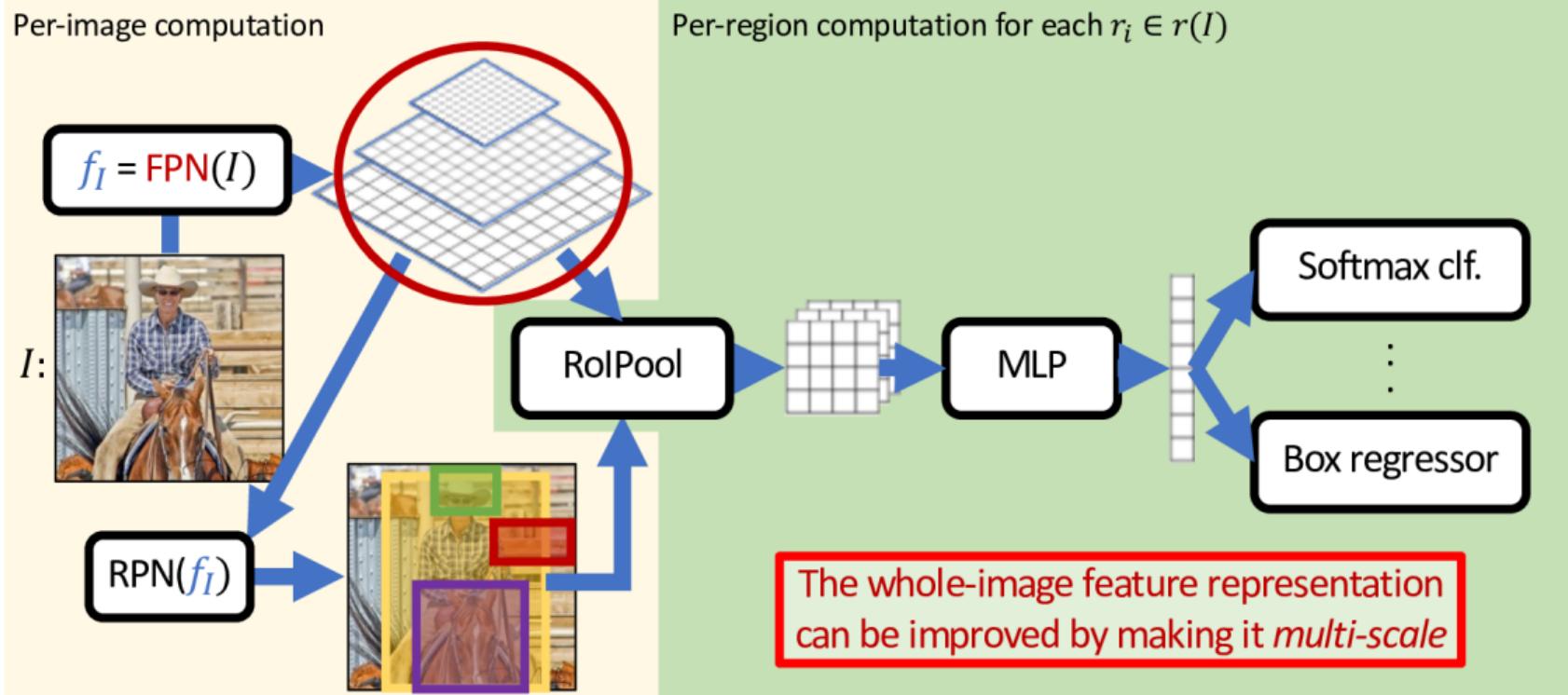


Single Stage Detection



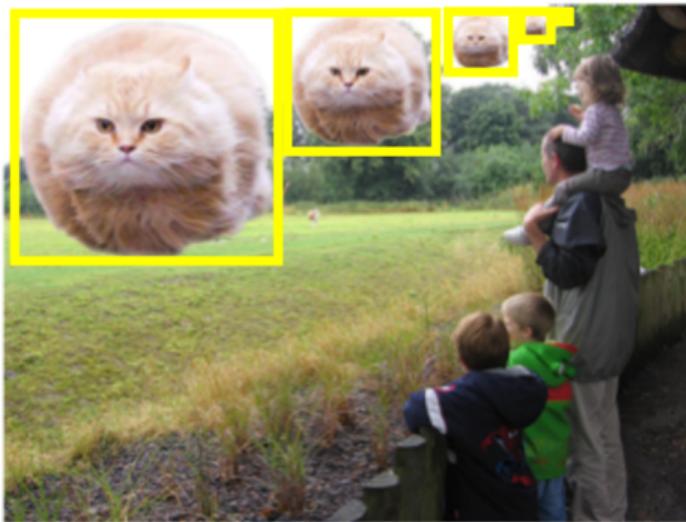
- ▶ YOLO and SSD: Alternative to proposal based (two stage) methods
- ▶ Predict 2D bounding boxes in a single stage
- ▶ Faster, but often worse performance in practice

Feature Pyramid Network



Feature Pyramid Network

Goal of Feature Pyramid Network: Improve Scale Equivariance

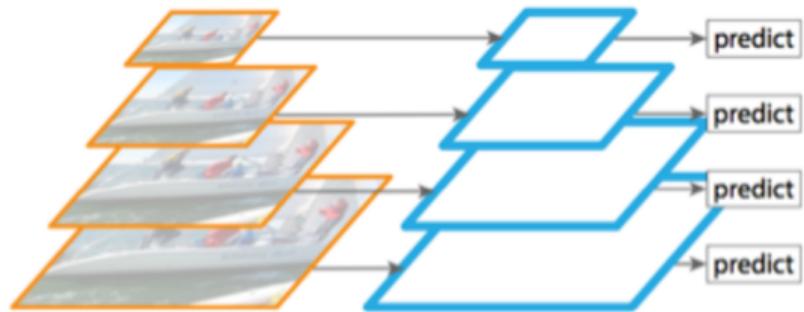
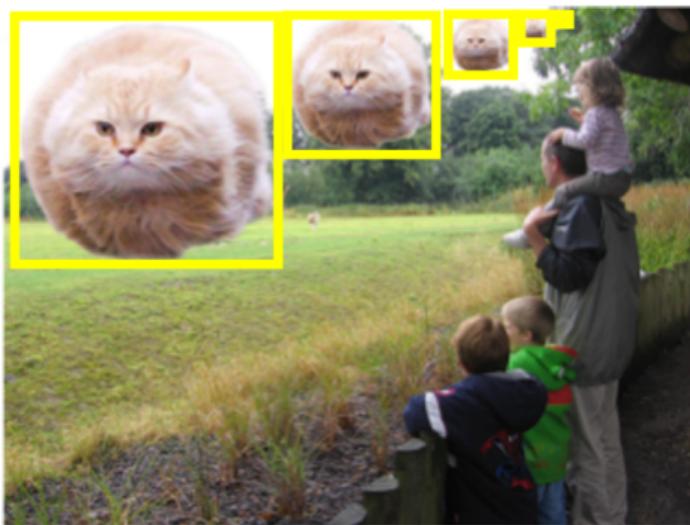


Detectors need to
1. classify and
2. localize
objects over a **wide range of scales**

FPN improves this ability

Feature Pyramid Network

Strategy 1: Image Pyramid



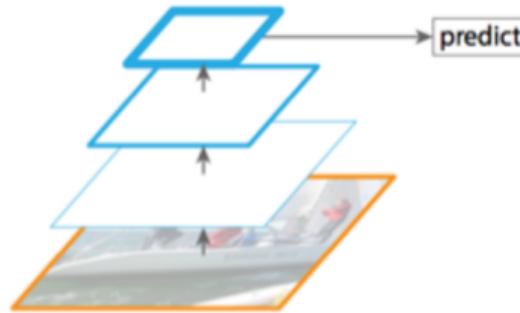
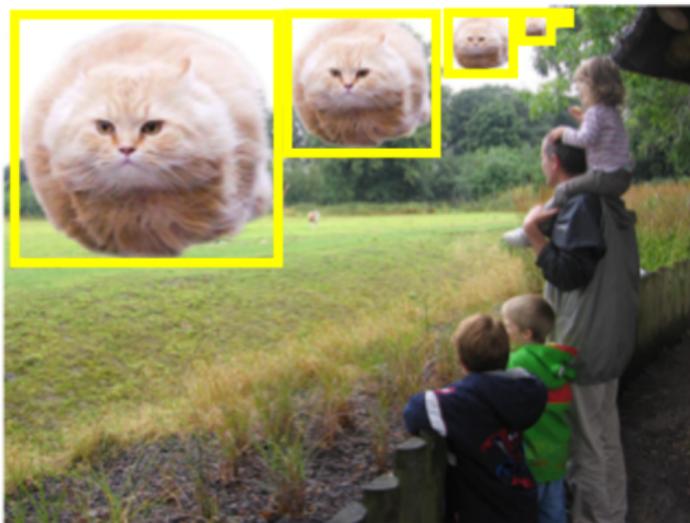
(a) Featurized image pyramid

Standard solution – *slow!*

(E.g., Viola & Jones, HOG, DPM, SPP-net,

Feature Pyramid Network

Strategy 2: Multi-scale Features (Single-scale Map)

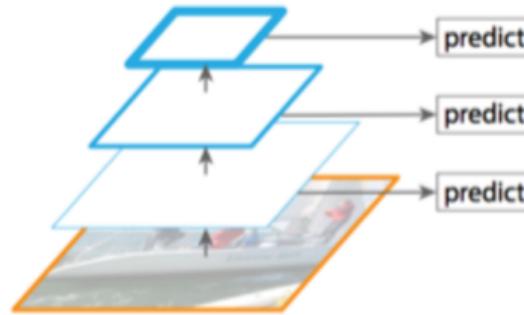
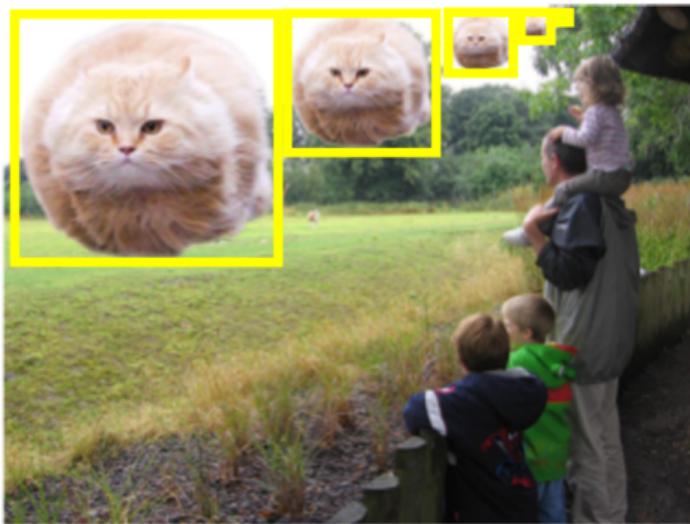


(b) Single feature map

Leave it all to the features – *fast, suboptimal*
(E.g., Fast/er R-CNN, YOLO, ...)

Feature Pyramid Network

Strategy 3: Naïve In-network Pyramid

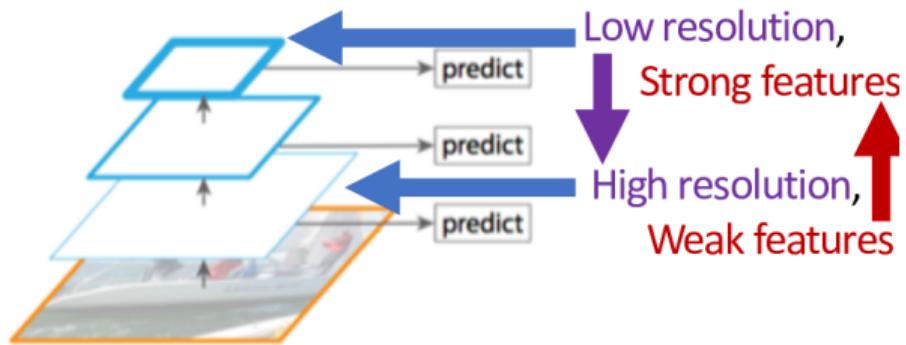
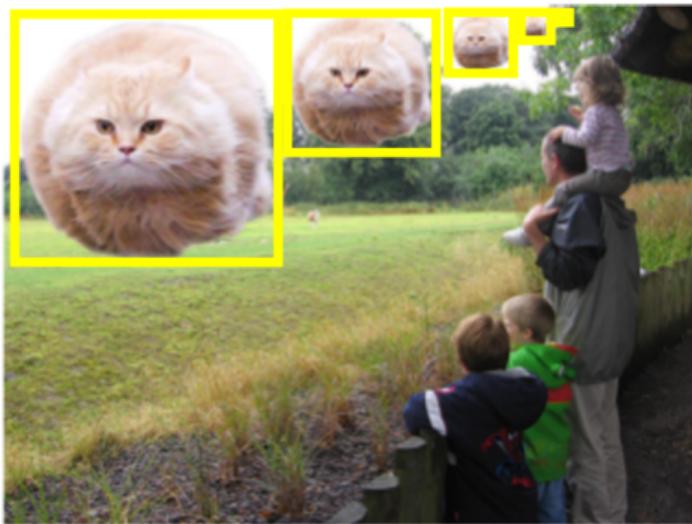


(c) Pyramidal feature hierarchy

Use the internal pyramid – *fast, suboptimal*
(E.g., \approx SSD, ...)

Feature Pyramid Network

Strategy 3: Naïve In-network Pyramid

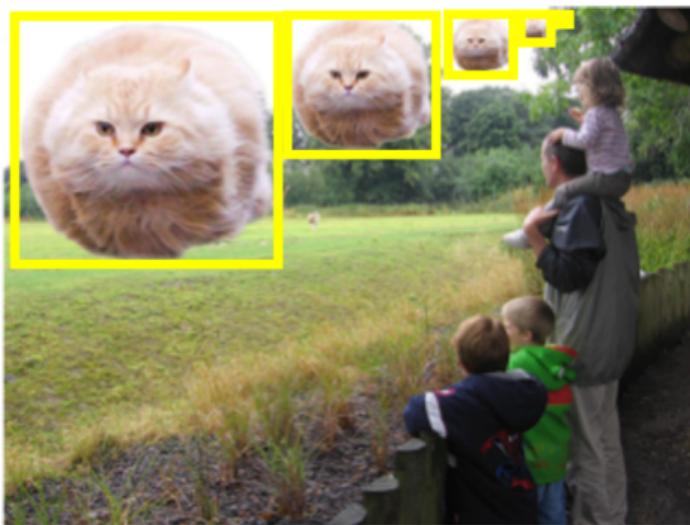


(c) Pyramidal feature hierarchy

Use the internal pyramid – *fast, suboptimal*
(E.g., \approx SSD, ...)

Feature Pyramid Network

Strategy 4: Feature Pyramid Network

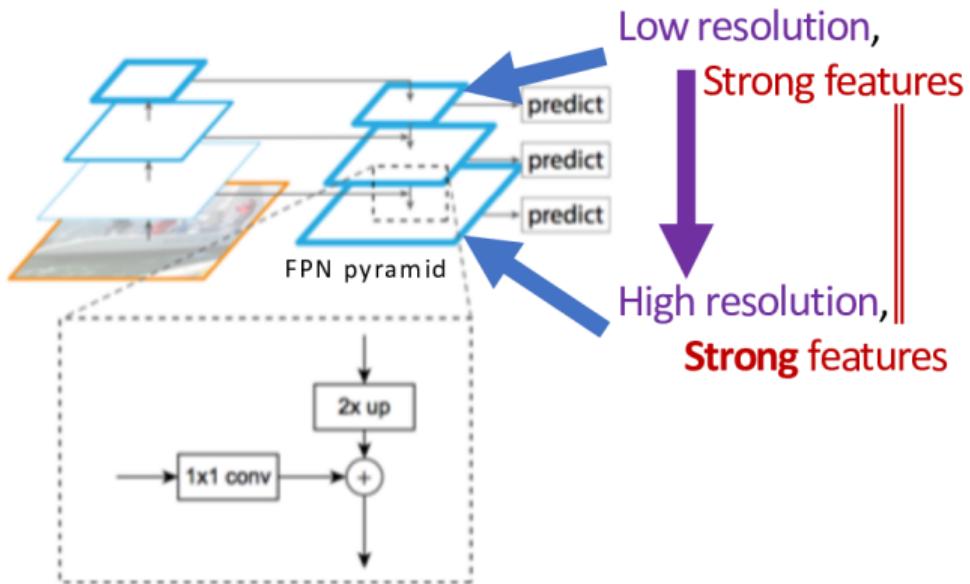
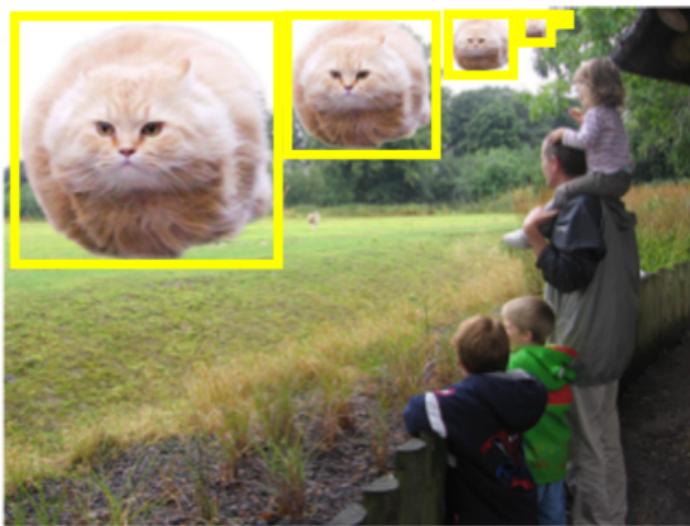


(d) Feature Pyramid Network

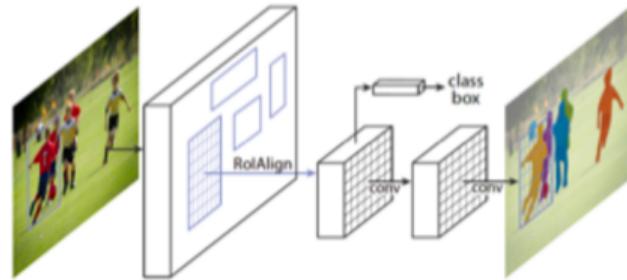
Top-down enrichment of high-res features –
fast, less suboptimal

Feature Pyramid Network

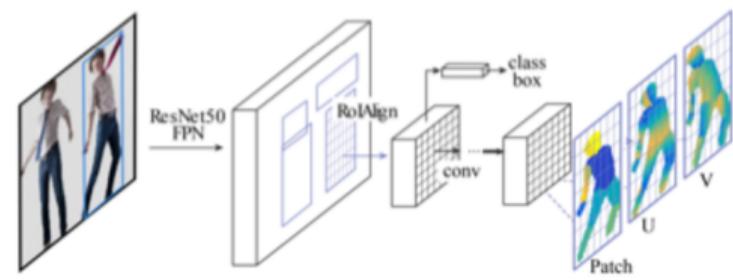
Strategy 4: Feature Pyramid Network



Generalization to other Output Modalities



Mask R-CNN
[He, Gkioxari, Dollár, Girshick]

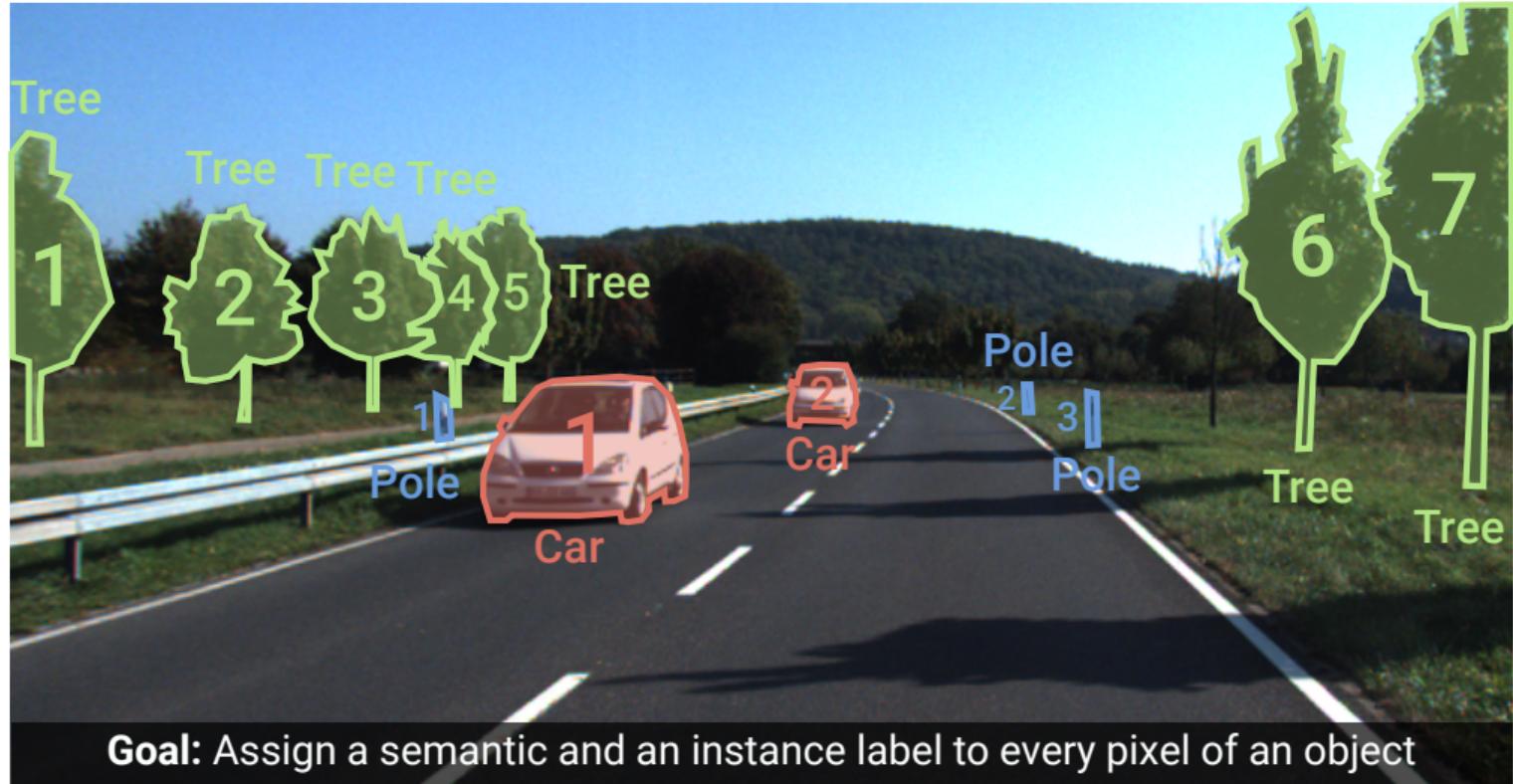


DensePose
[Güler, Neverova, Kokkinos]

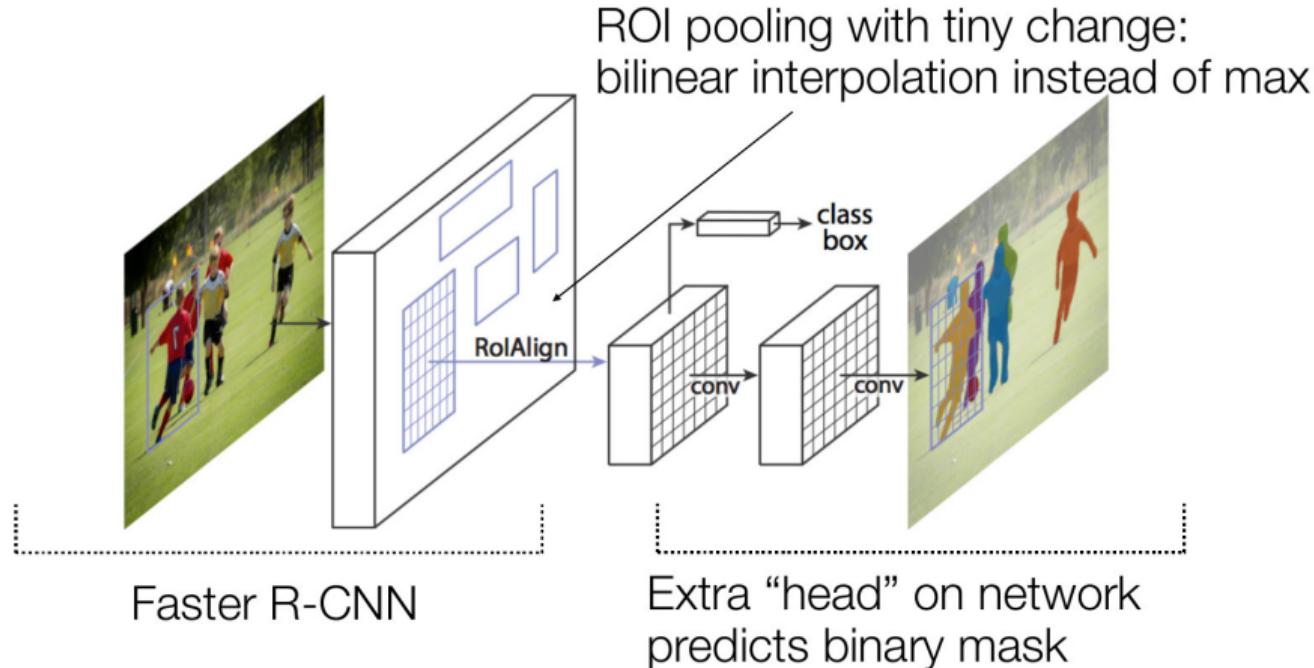
It is easy to add **additional network heads** to this framework:

- ▶ **Mask R-CNN:** Predicting an instance segmentation mask per detection
- ▶ **DensePose:** Predict object coordinates (texture map coordinates)

Instance Segmentation



Mask R-CNN: Faster R-CNN for Instance Segmentation



- ▶ Predict binary instance segmentation (foreground vs. background) per detection

Mask R-CNN: Faster R-CNN for Instance Segmentation

Image with training proposal



28x28 mask target

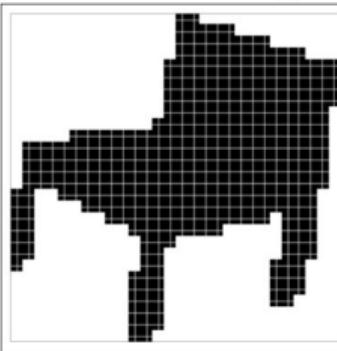
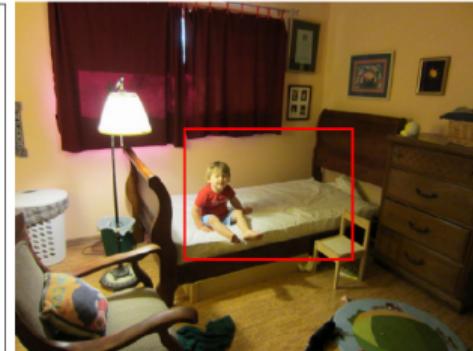
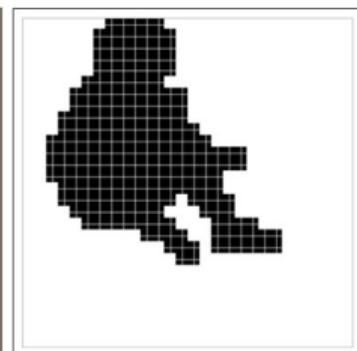
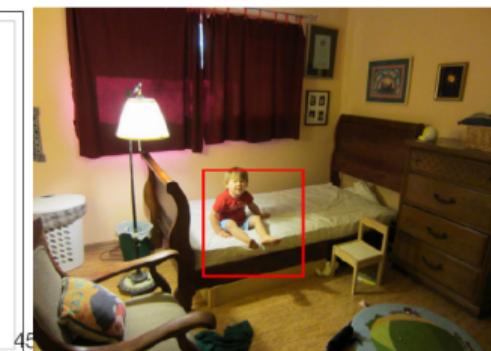
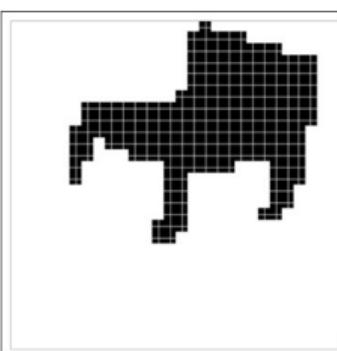


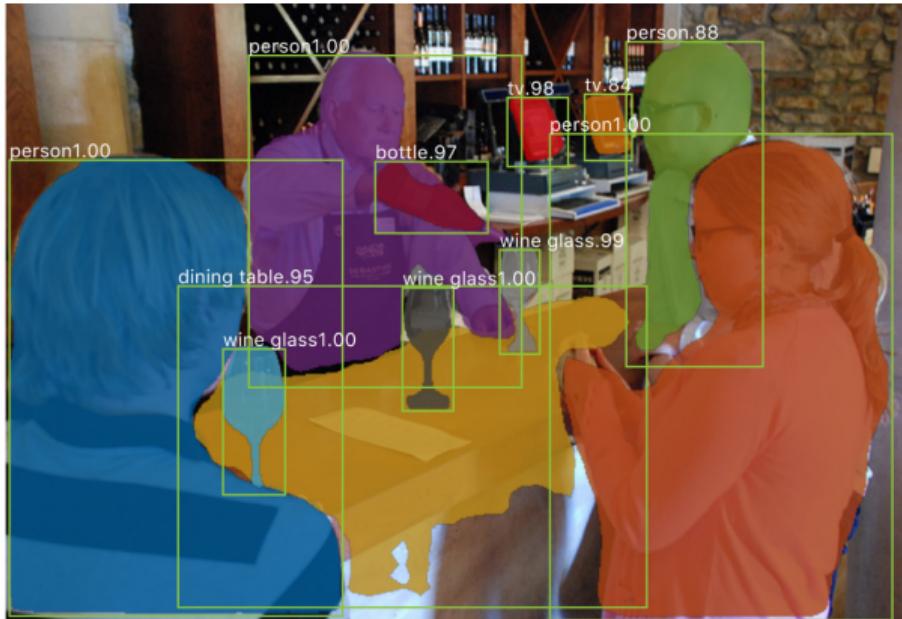
Image with training proposal



28x28 mask target



Mask R-CNN: Faster R-CNN for Instance Segmentation



- ▶ Evaluation: Intersection-over-Union at mask level (not bounding box level)

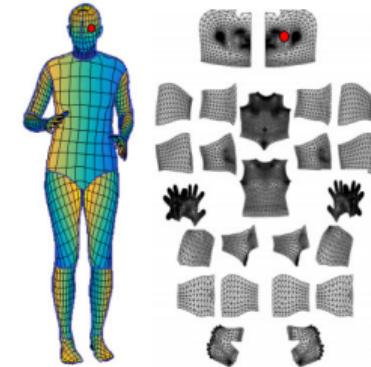
DensePose: Faster R-CNN for Dense Human Pose Estimation



DensePose-RCNN Results



DensePose COCO Dataset



- ▶ Map all human pixels of an RGB image to the 3D surface of the human body
- ▶ Densely regress part-specific UV coordinates within every human region
- ▶ Main contribution: DensePose-COCO, a large-scale dataset with manually annotated image-to-surface correspondences

Mesh R-CNN: Faster R-CNN for Meshes



Box & Mask Predictions

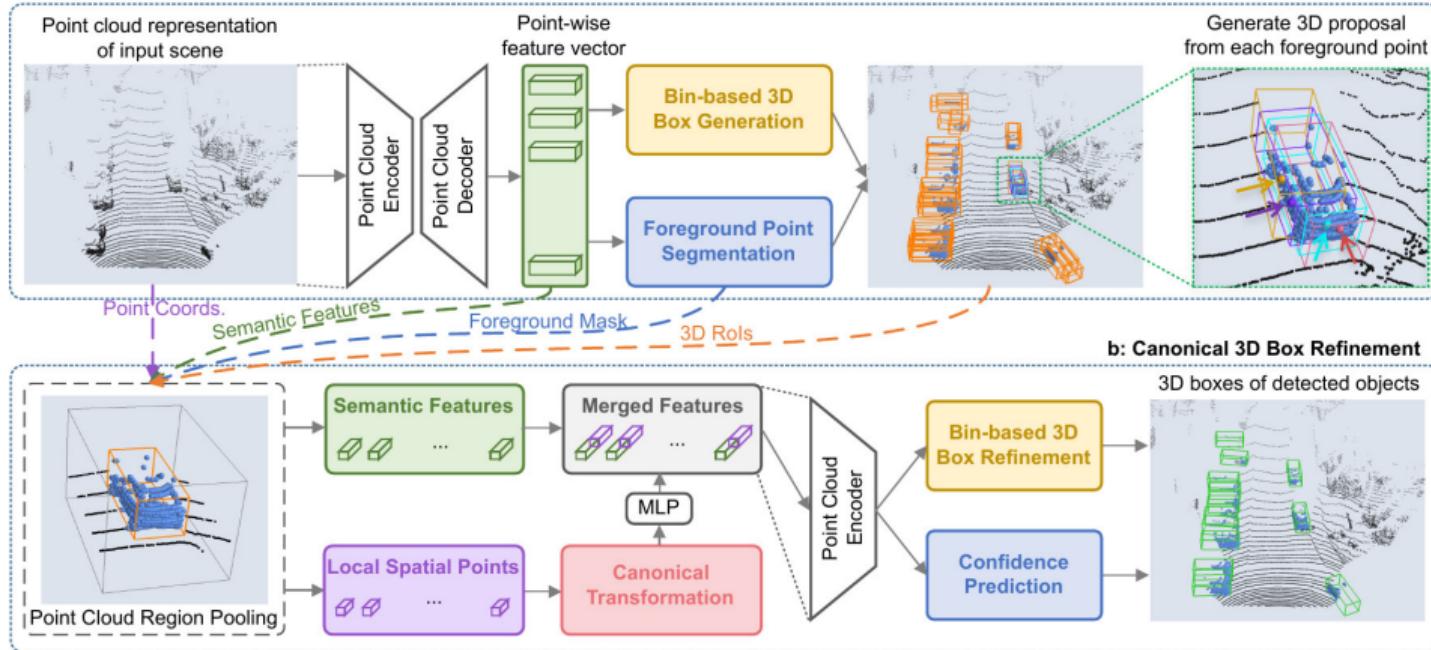


Mesh Predictions

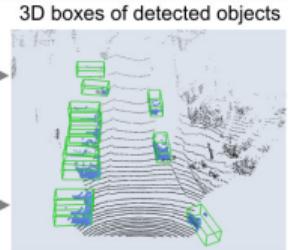
- ▶ Predict vertices and faces of a 3D mesh for each 2D bounding box

PointRCNN: Faster R-CNN for Point Clouds

a: Bottom-up 3D Proposal Generation

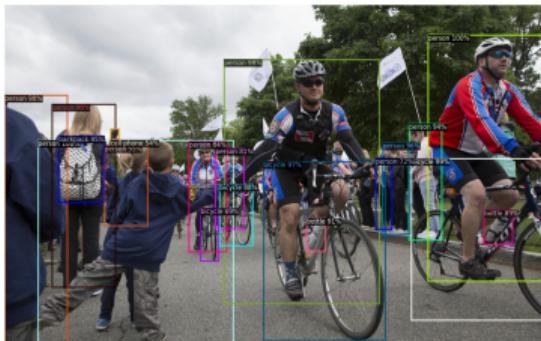


b: Canonical 3D Box Refinement



- Predict 3D bounding boxes from point cloud (e.g., Velodyne HDL-64 laser scan)

Detectron2 for PyTorch



<https://github.com/facebookresearch/detectron2>