greatlearning
Learning for Life

Advanced Neural Networks for Computer Vision

greatlearning
Learning for Life

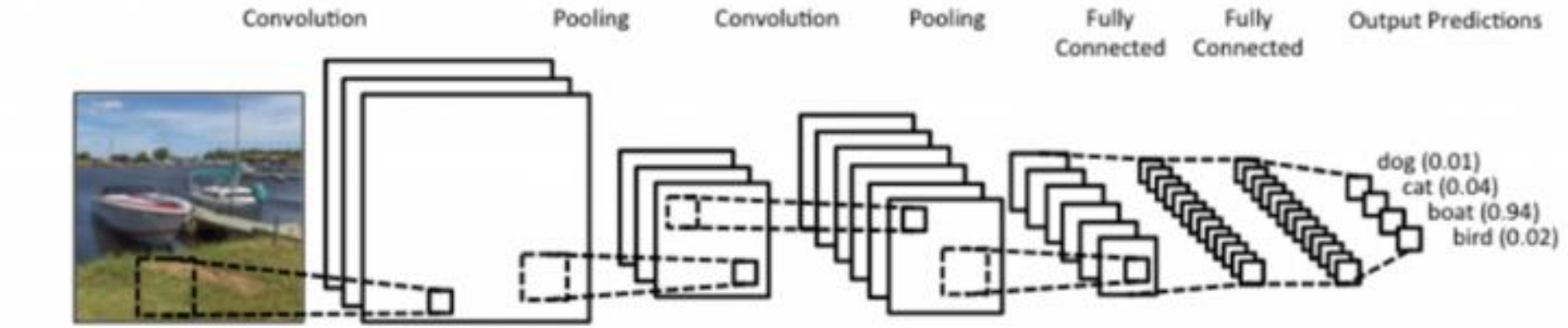
Module objectives

- Identify problems other than image classification
- Match advanced NN architectures suitable for these problems
- Approaches to solve the problems
- Metrics to measure

Revising CNN's for Object Classification

Typical

ConvNet



$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: number of input features

n_{out}: number of output features

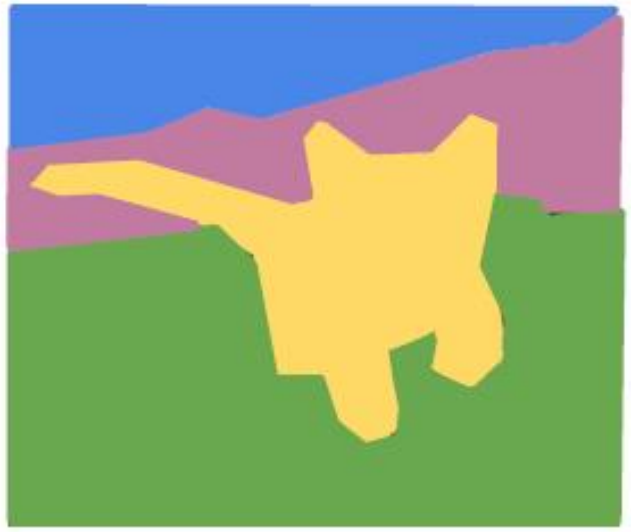
k: convolution kernel size

p: convolution padding size

s: convolution stride size

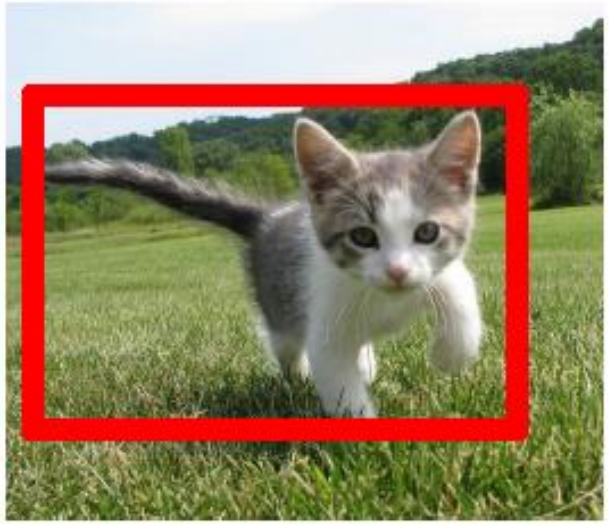
Classification + Localization

Classification + detection



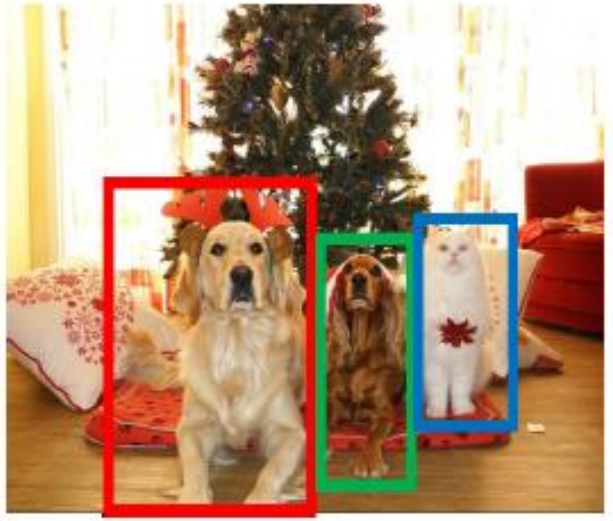
GRASS, CAT, TREE, SKY

No objects, just pixels



CAT

Single Object



DOG, DOG, CAT

Multiple Object

In this first session, we will talk about 2 problems

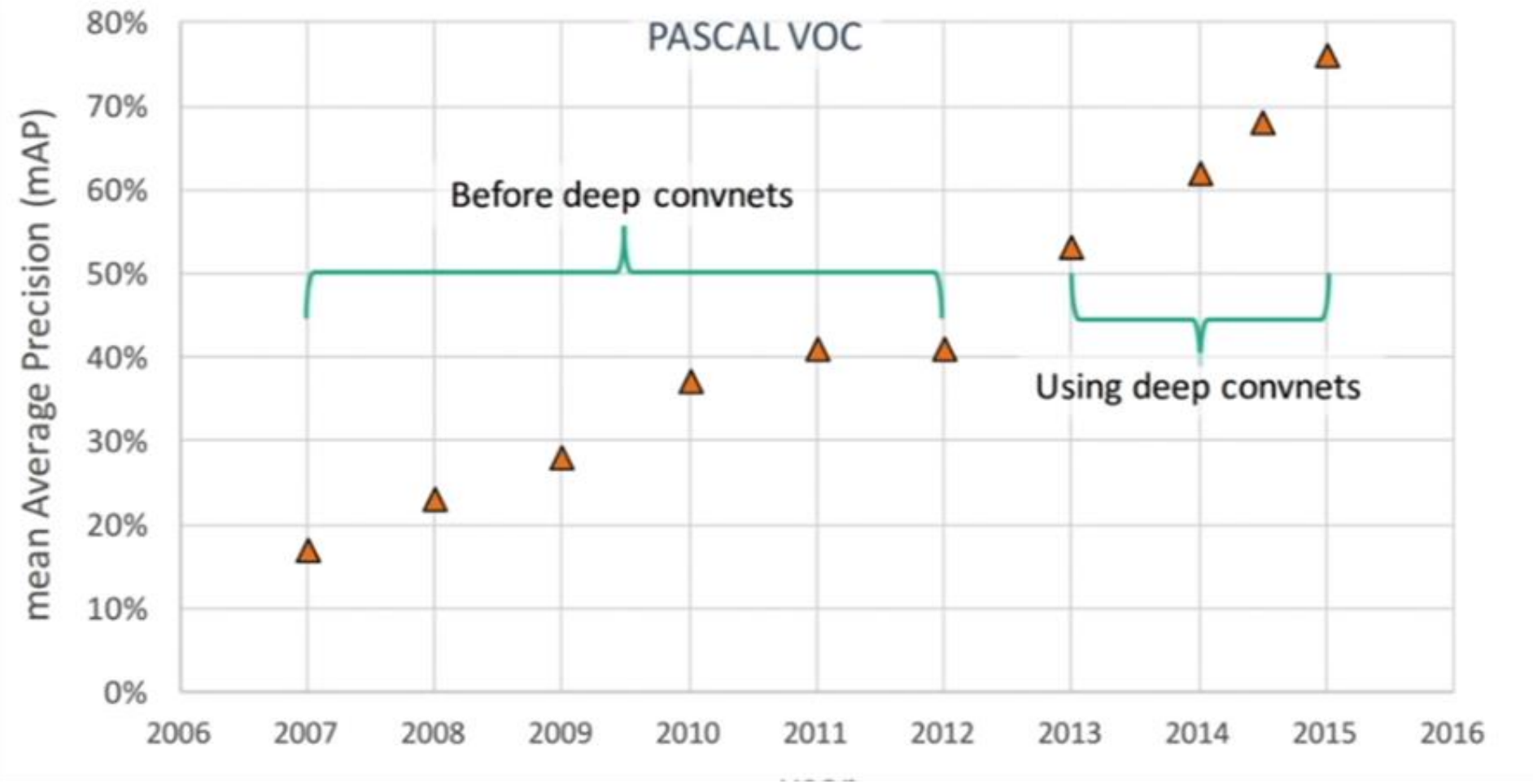
1.Object Detection

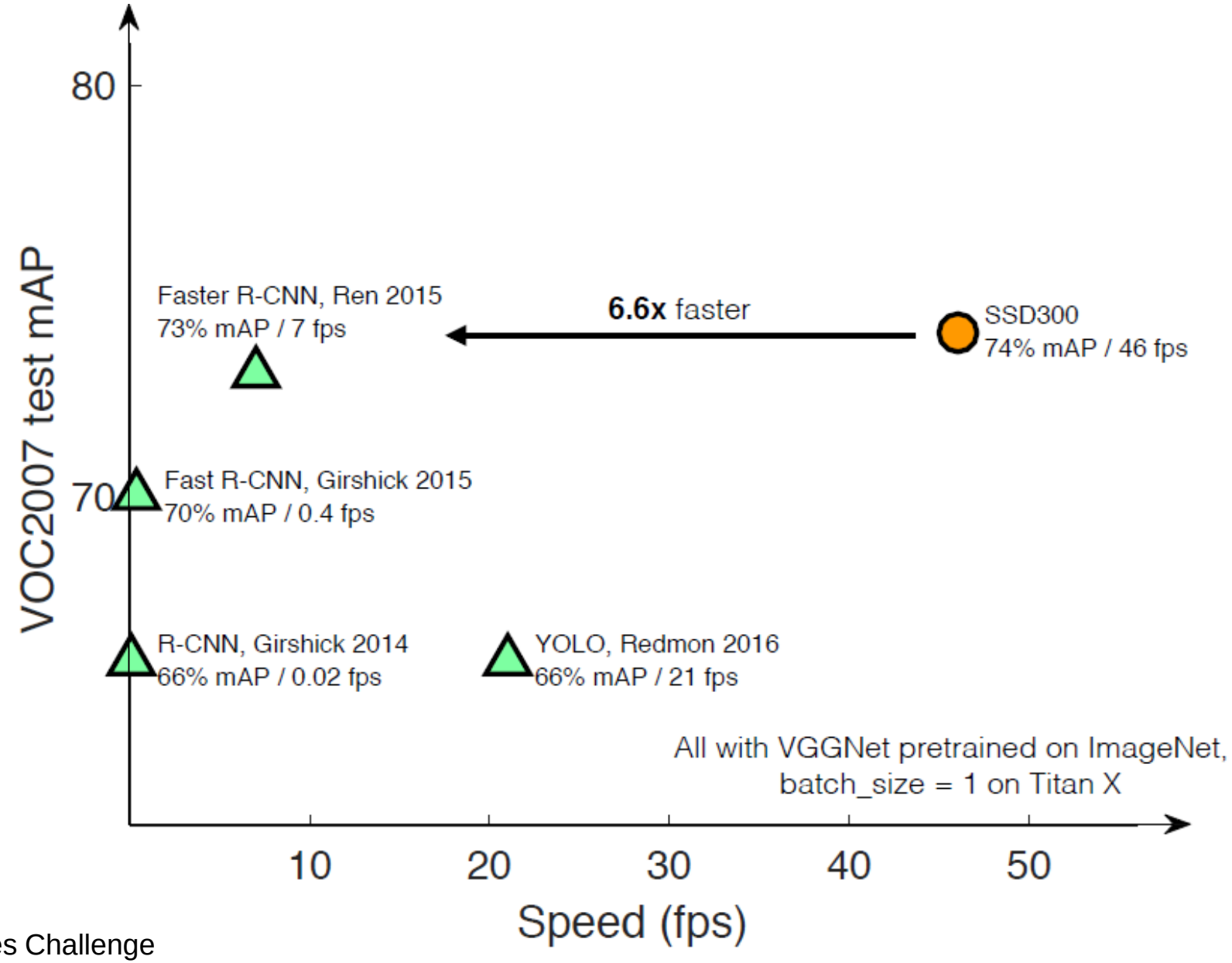
- Different approches
- Region proposals and Selective search
- R-CNN, Fast R-CNN, Faster R-CNN
- YoLO, SSD
- Metric - IoU
- Python notebook

2.Semantic Segmentation

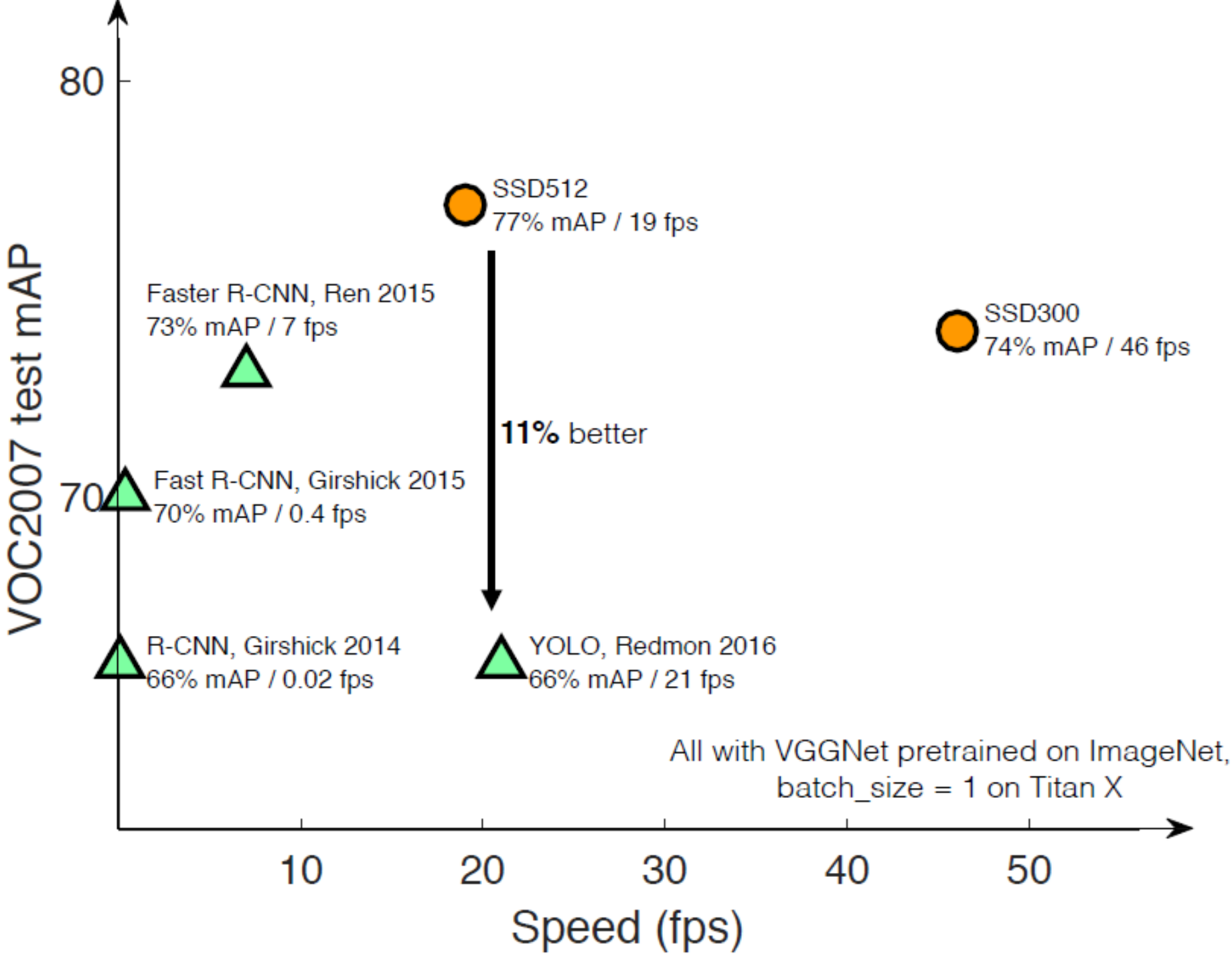
- Approches
- Sliding window, fully convolutional, upSampling
- UNet
- Python Notebook

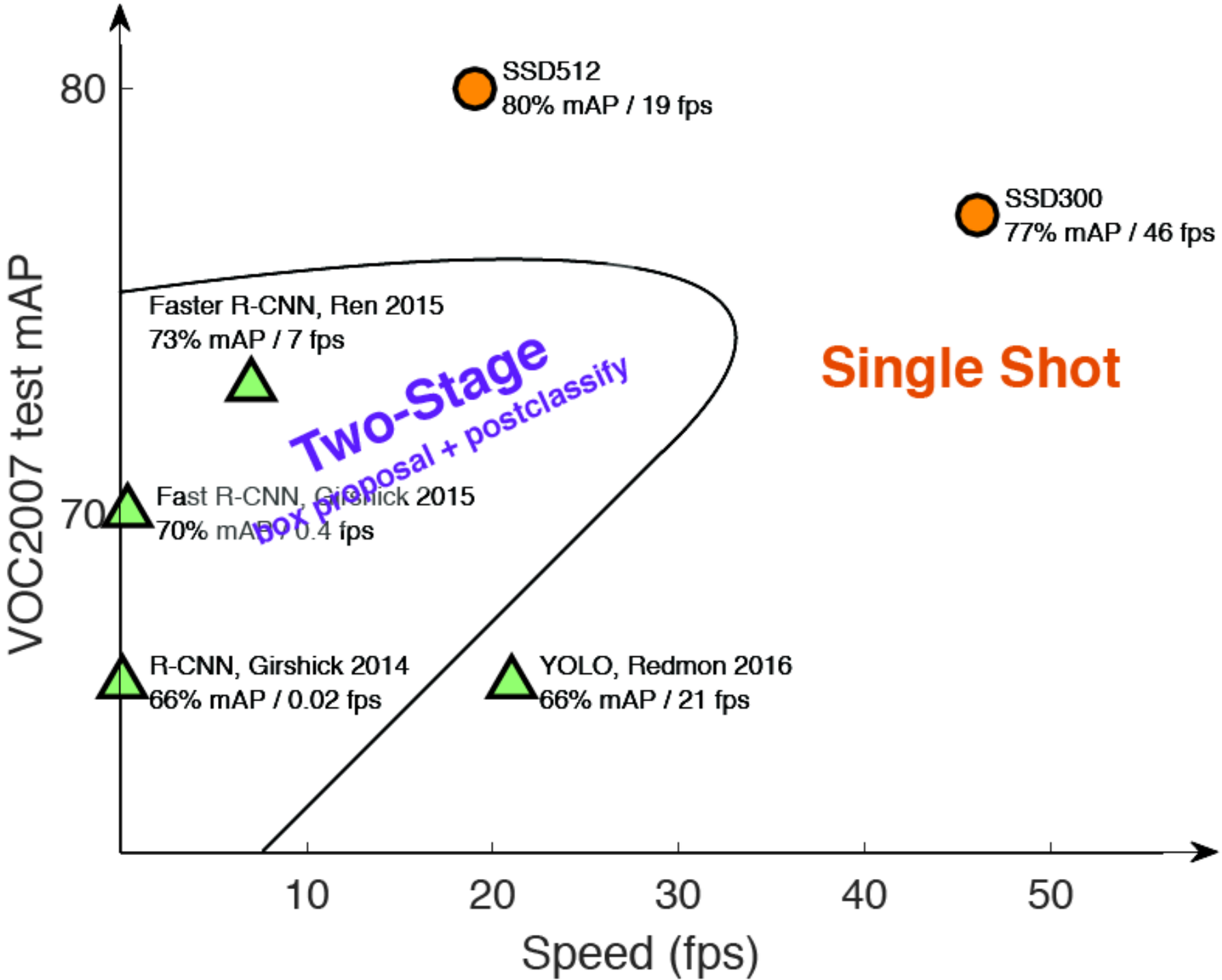
Object Detection: Impact of Deep Learning





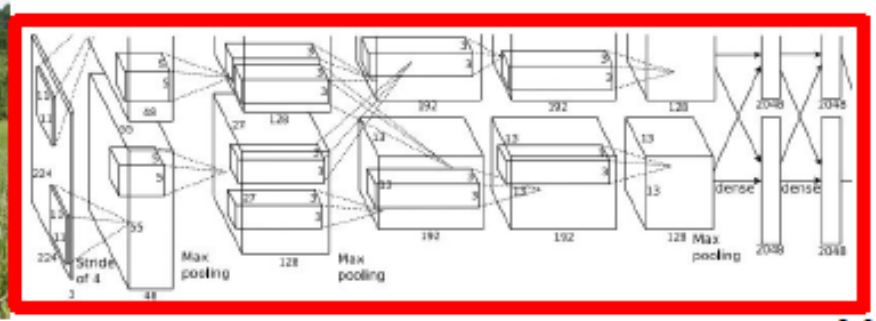
Pascal Visual Object Classes Challenge
:
http://host.robots.ox.ac.uk/pascal/VOC/voc2007/




80
70
VOC2007 test mAP
SSD512
80% mAP / 19 fps
SSD300
77% mAP / 46 fps
Faster R-CNN, Ren 2015
73% mAP / 7 fps
Two-Stage
box proposal + postclassify
Single Shot
Fast R-CNN, Girshick 2015
70% mAP / 0.4 fps
R-CNN, Girshick 2014
66% mAP / 0.02 fps
YOLO, Redmon 2016
66% mAP / 21 fps
10
20
30
40
50
Speed (fps)

Classification + Localization

This image is CC0 public domain



Often pretrained on ImageNet
(Transfer learning)

Vector: 4096

Fully Connected: 4096 to 1000

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

Softmax Loss

Correct label: Cat

Multitask Loss

Loss

Fully Connected: 4096 to 4

Box Coordinates (x, y, w, h)

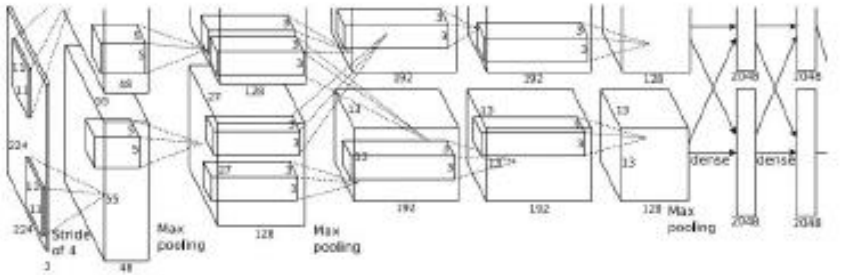
L2 Loss

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

Treat localization as a regression problem!

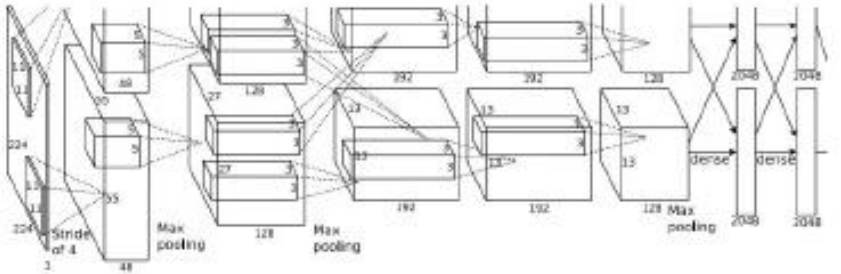
Object Detection as Regression?

Each image needs a different number of outputs!



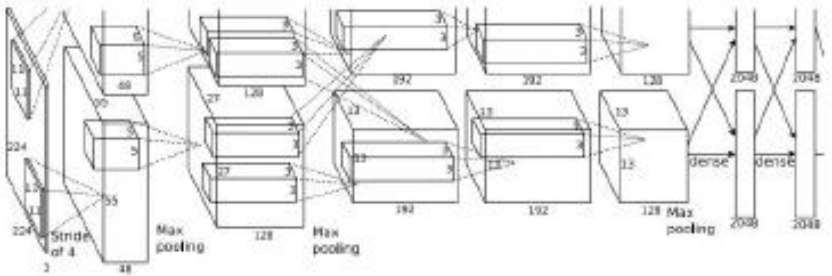
CAT: (x, y, w, h)

4 numbers



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

16 numbers

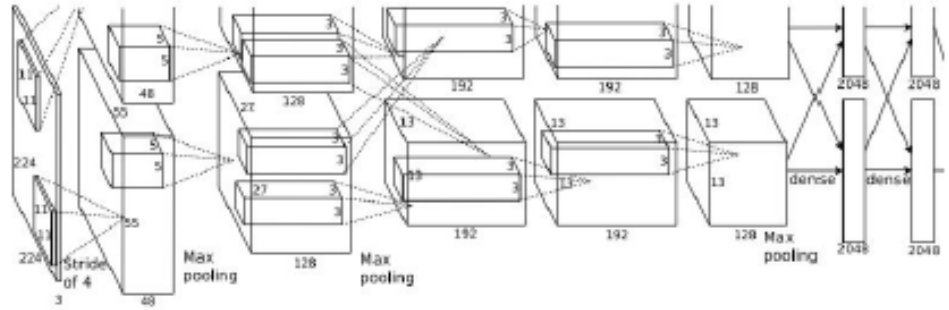


DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
....

Many numbers!

Object Detection as Classification: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

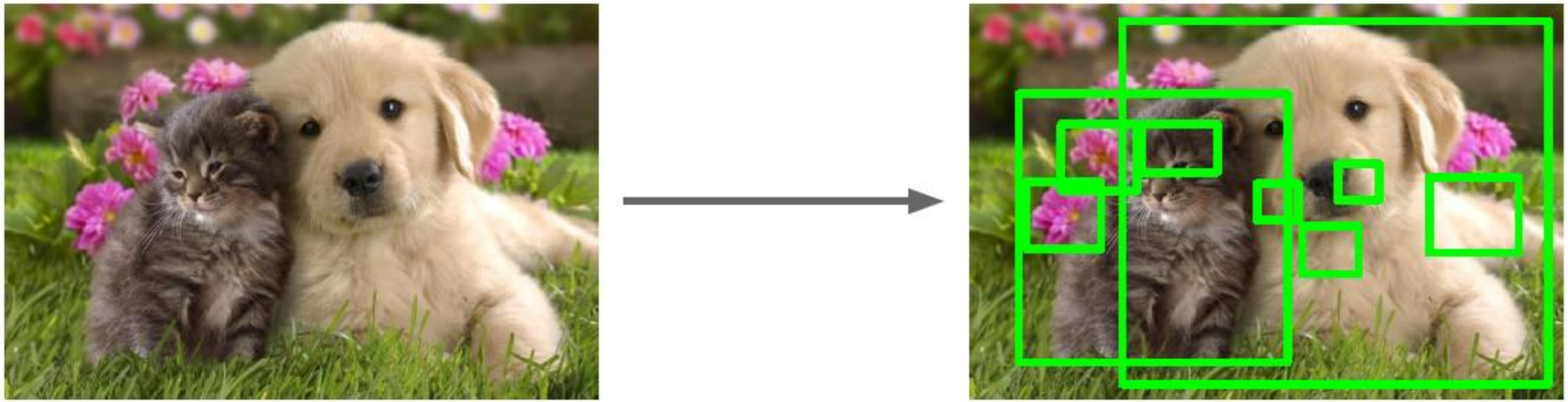


Dog? NO
Cat? YES
Background? NO

Problem: Need to apply CNN to huge number of locations and scales, very computationally expensive!

Region Proposals

- Find “blobby” image regions that are likely to contain objects
- Relatively fast to run; e.g. Selective Search gives 1000 region proposals in a few seconds on CPU

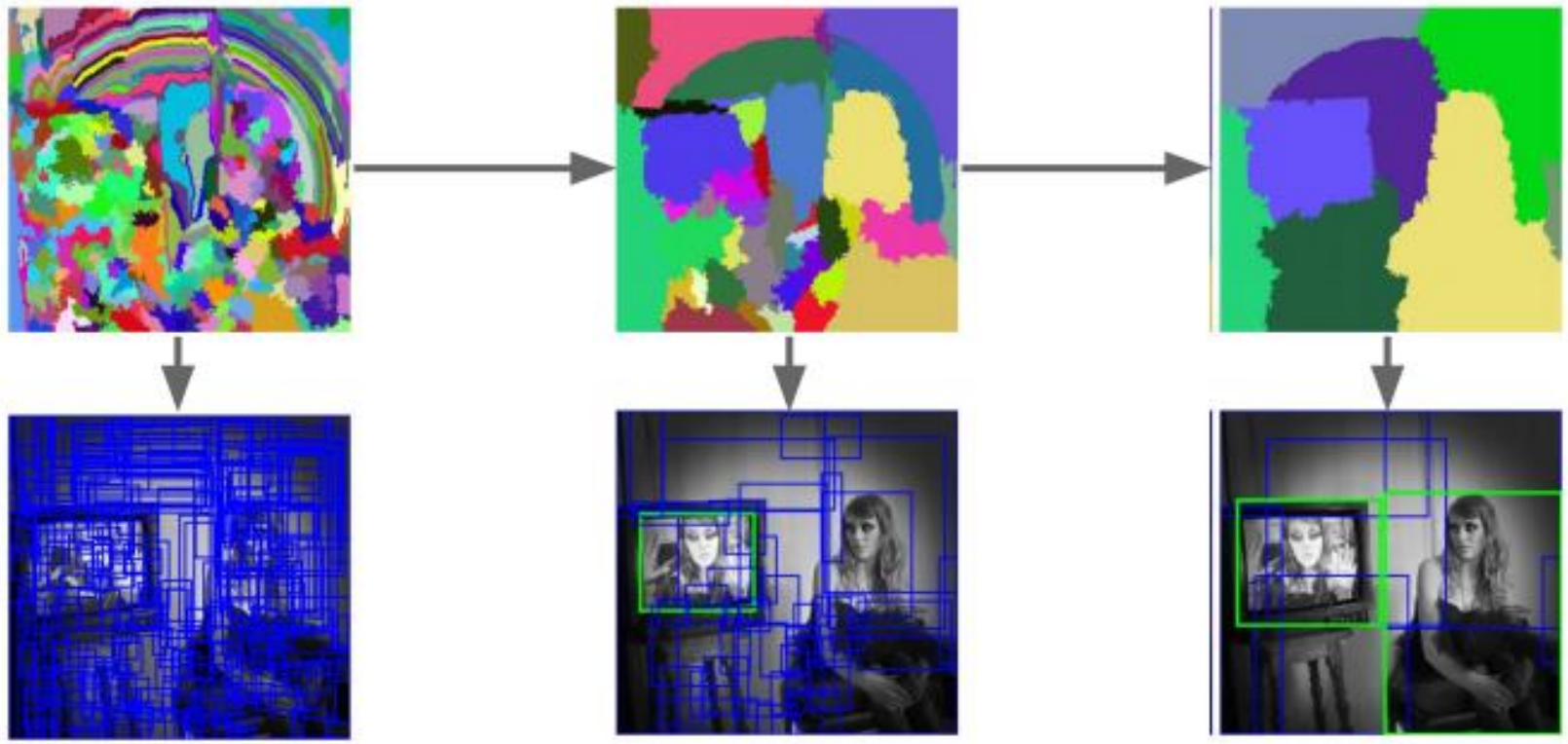


Alexe et al, "Measuring the objectness of image windows", TPAMI 2012
Uijlings et al, "Selective Search for Object Recognition", IJCV 2013
Cheng et al, "BING: Binarized normed gradients for objectness estimation at 300fps", CVPR 2014
Zitnick and Dollar, "Edge boxes: Locating object proposals from edges", ECCV 2014

Region Proposals: Selective Search

Bottom-up segmentation, merging regions at multiple scales

Convert regions to boxes



Uijlings et al, “Selective Search for Object Recognition”, IJCV 2013

Selective Search for object recognition

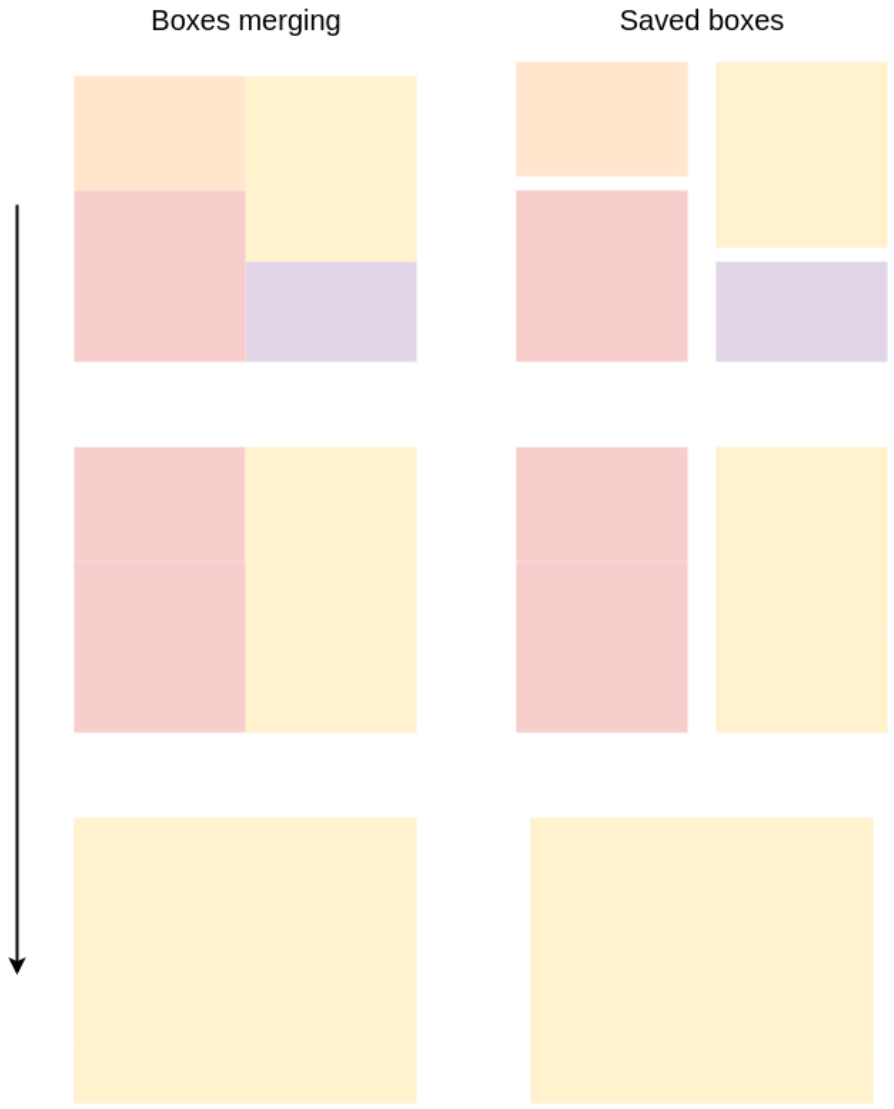
- **Selective Search** uses the best of both worlds: Exhaustive search and segmentation.
- Segmentation improve the sampling process of different boxes - **reduces considerably the search space.**
- To improve the algorithm's robustness a variety of strategies are used during the bottom-up boxes' merging. (That implies multiple passes)
- Selective Search produces boxes that are good proposals for objects, it handles well different image conditions, but more important it is fast enough to be used in a prediction pipeline (like Fast-RCNN) to do real-time object detection.

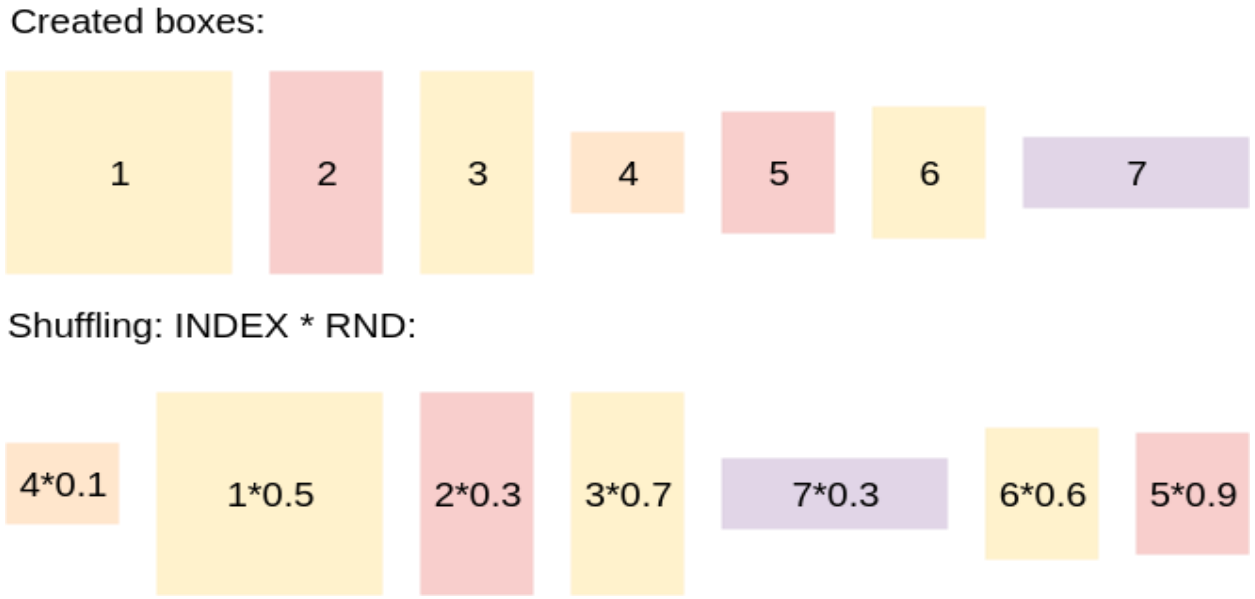
The selective search Algorithm

At first the authors produce a sampling a bounding boxes based on regions' segmentation produced by the efficient graph based segmentation.

Starting from these initial boxes, the authors use a **bottom-up merging based on similarity**: Boxes, small at first, are merged with their most similar neighbour box.

The history of all seen boxes at the different steps of the algorithm is kept.





By keeping all existing boxes, the search can capture **all scales which is important in hierarchical image**: *Imagine a pilot in a plane: the pilot's box in comprised in the bigger plane's box.*

Plenty of boxes are created, the last box is the entire image!

The authors sort the boxes by creation time, the most recent first. To avoid privileging too much large boxes, the box's index (*1 being the most recent box*) is multiplied by a **random number between 0 and 1.**

The model using the selective search can now make a trade-off between having all location proposals and getting only the *k*-first most probable.

Diversification Strategies

3 strategies to improve the search's robustness:

1. **Different color spaces**

In order to **handle different lightning,** the authors apply their algorithm to the same image transposed in different color spaces.
The most known color space is RGB
Other used color spaces:

- Grayscale: Where a pixel has for single value its intensity
- Lab: Luminocity L*a*b
- HSV: With hue, saturation, and a value

2. **Different Starting Regions**

After generating initial boxes for regions, parameter k is chosen that affects the size of the regions.

Different starting regions from the segmentation affect deeply the selective search.

3. Different Similarity Measures

These similarities are added together producing a **final similarity measure.**

3.1. Color Similarity

Each box has a **color histogram of 25 bins.** The similarity of two boxes is the histogram intersection (common pixel values)

3.2. Texture Similarity

Textures are important and texture historm is created with SIFT.

3.3. Size Similarity

The size similarity has been created in order to avoid an imbalance between the boxes' size. Where one growing big box would forbid intermediary boxes to form.
The propagation of this feature is **simply the sum of the two sizes**.

3.4. Fitness Feature

The initial boxes created from the segmentation **may overlap.** Two overlapping boxes

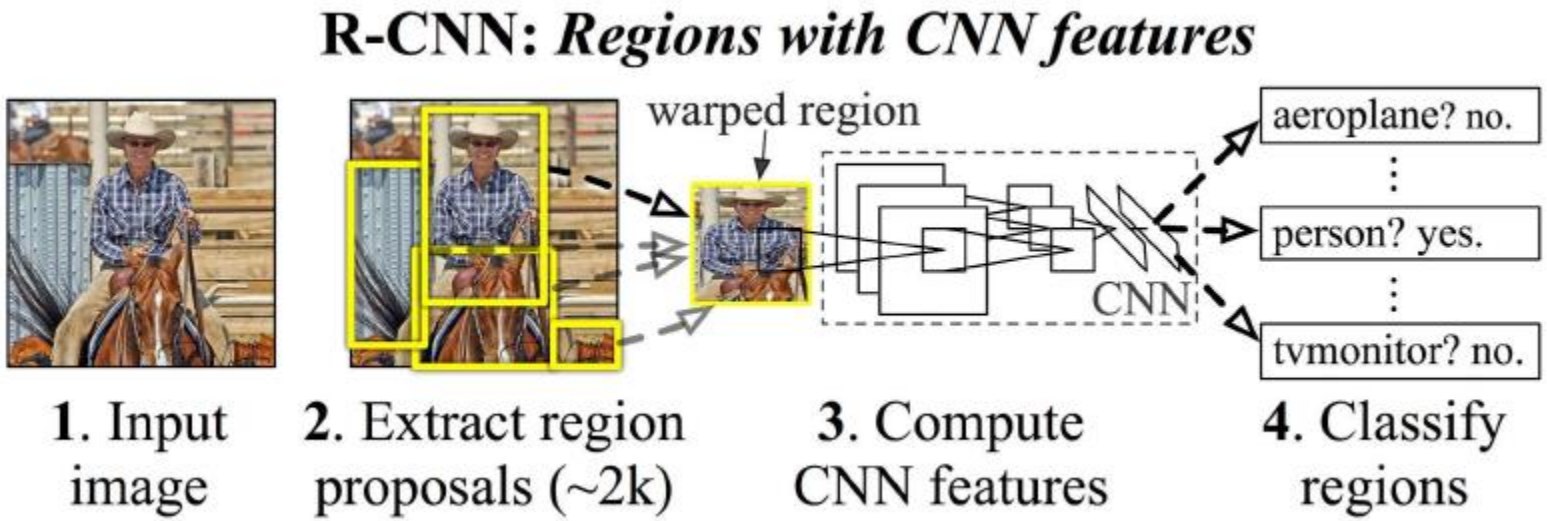
Region Proposals: Many other choices

Method	Approach	Outputs Segments	Outputs Score	Control #proposals	Time (sec.)	Repea-tability	Recall Results	Detection Results
Bing [18]	Window scoring		✓	✓	0.2	★★★	★	·
CPMC [19]	Grouping	✓	✓	✓	250	-	★★	★
EdgeBoxes [20]	Window scoring		✓	✓	0.3	★★	★★★	★★★
Endres [21]	Grouping	✓	✓	✓	100	-	★★★	★★
Geodesic [22]	Grouping	✓		✓	1	★	★★★	★★
MCG [23]	Grouping	✓	✓	✓	30	★	★★★	★★★
Objectness [24]	Window scoring		✓	✓	3	·	★	·
Rahtu [25]	Window scoring		✓	✓	3	·	·	★
RandomizedPrim's [26]	Grouping	✓		✓	1	★	★	★★
Rantalankila [27]	Grouping	✓		✓	10	★★	·	★★
Rigor [28]	Grouping	✓		✓	10	★	★★	★★
SelectiveSearch [29]	Grouping	✓	✓	✓	10	★★	★★★	★★★
Gaussian				✓	0	·	·	★
SlidingWindow				✓	0	★★★	·	·
Superpixels		✓			1	★	·	·
Uniform				✓	0	·	·	·

Hosang et al, "What makes for effective detection proposals?", PAMI 2015

Region-CNN

- R-CNN, first generates 2K region proposals (bounding box candidates), then detect object within the each region proposal as below:



How it works?

- Multi task loss : Softmax for Classification , L2 for Bounding Box Regression
- Gradient descent works only on 1 scaler so we use a weight function to the loss function
- This weights are applied using a hyperparameter (this is slightly difficult to choose as it affects the value of loss function directly)
- Weighted sum of two losses is the final loss which is used to adjust gradients during back propogation

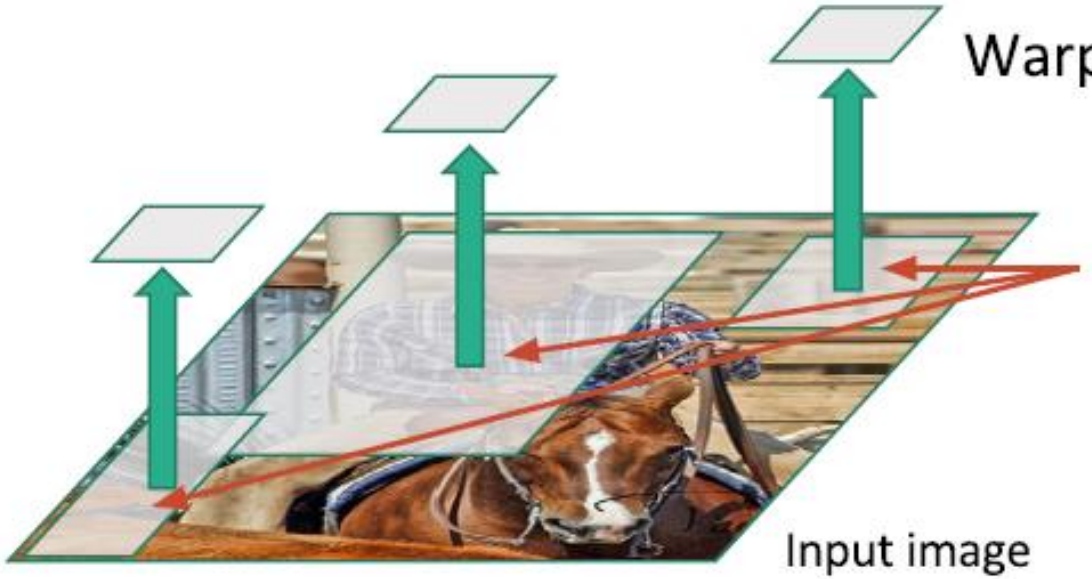
R-CNN

Input image

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

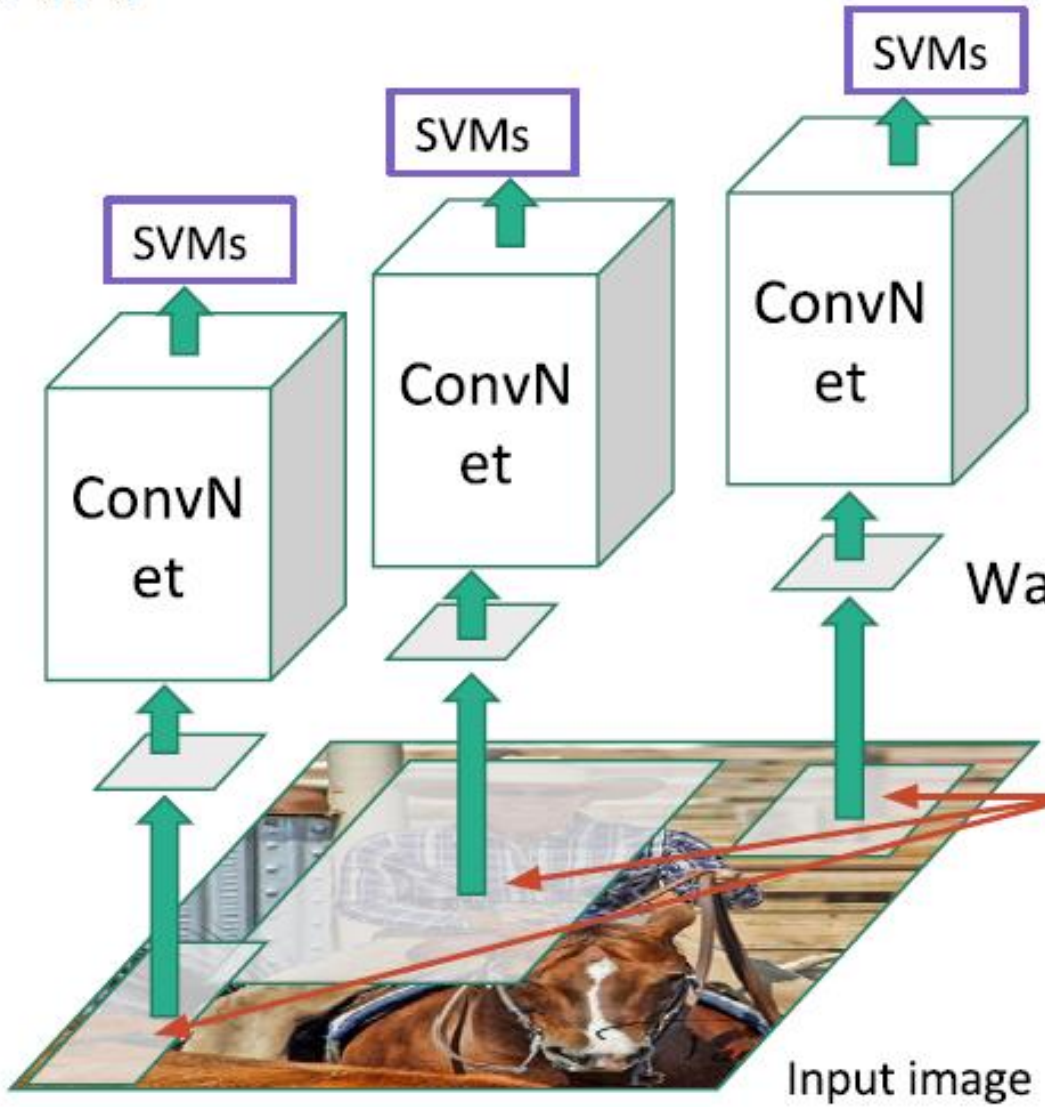
Refer to understand more https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

R-CNN



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN



Classify regions with SVMs

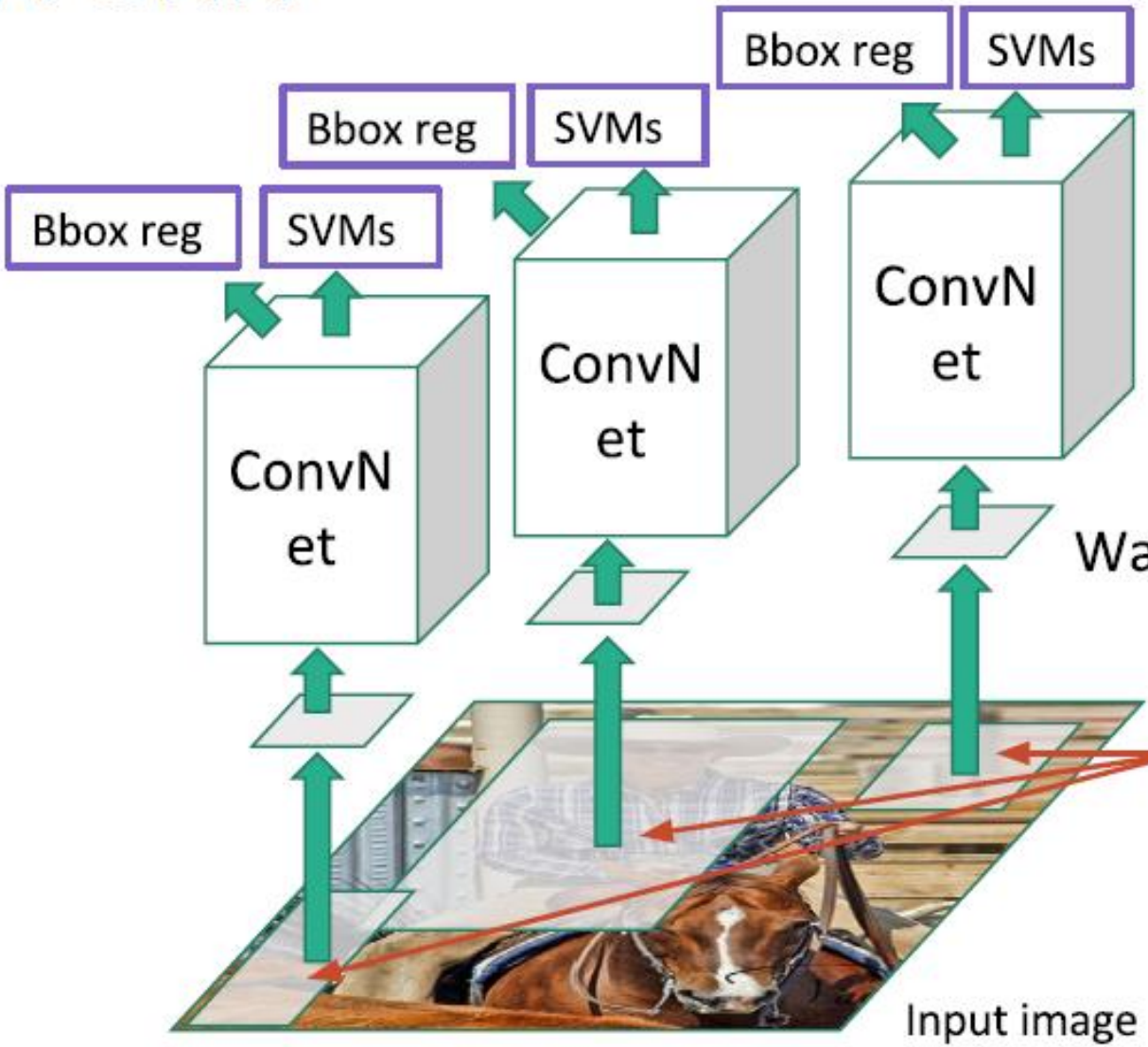
Forward each region through ConvNet

Warped image regions

Regions of Interest (RoI) from a proposal method (~2k)

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN



Linear Regression for bounding box offsets

Classify regions with SVMs

Forward each region through ConvNet

Warped image regions

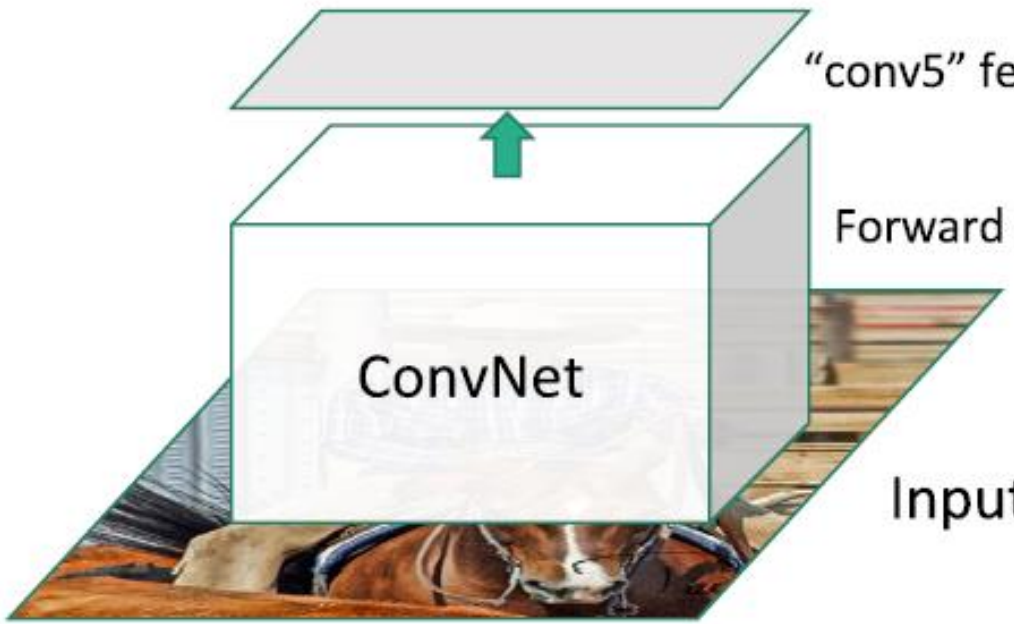
Regions of Interest (RoI) from a proposal method (~2k)

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Disadvantage of R-CNN:

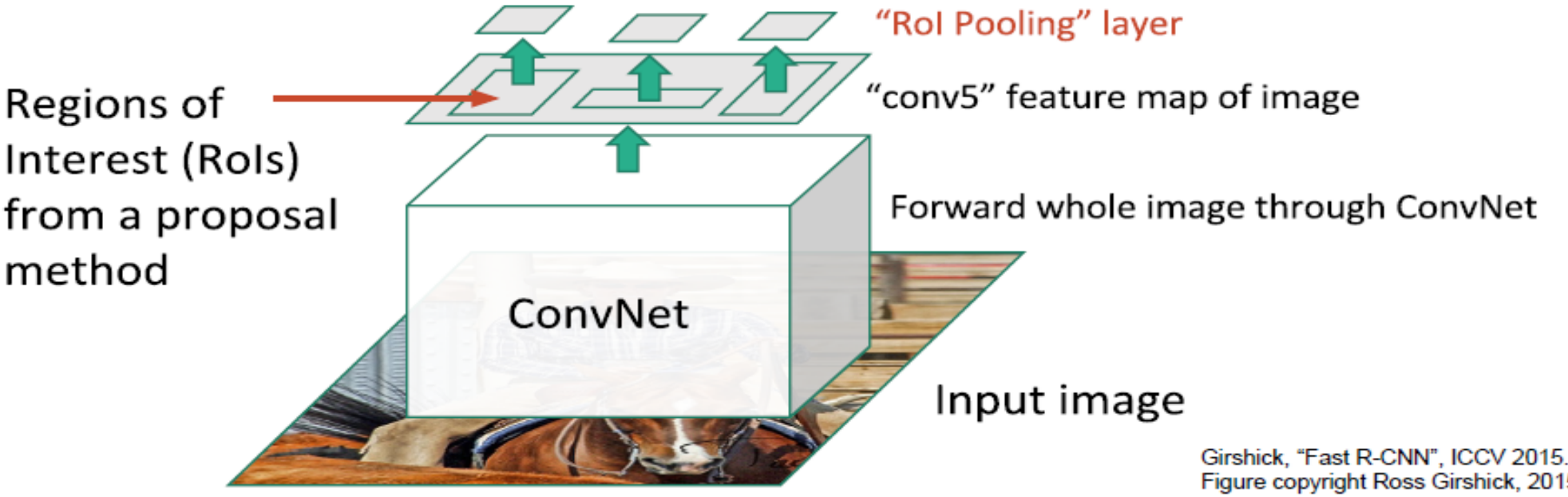
Separate convnet for each box (slow)

Fast R-CNN



Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

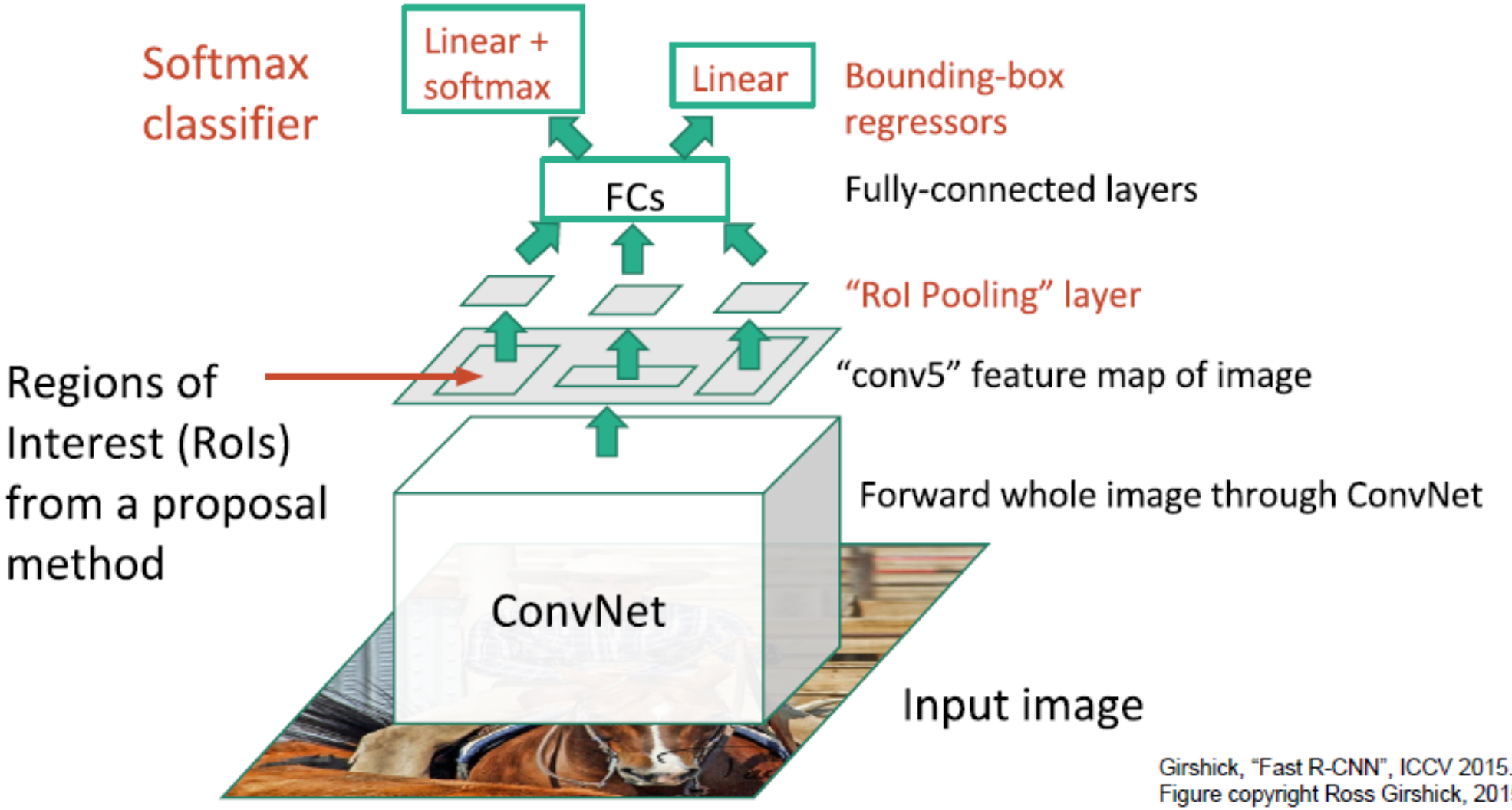
Fast R-CNN



Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Refer to this to know more - https://medium.com/@tanaykarmarkar/region-proposal-network-rpn-backbone-of-faster-r-cnn-4a744a38d7f9

Fast R-CNN



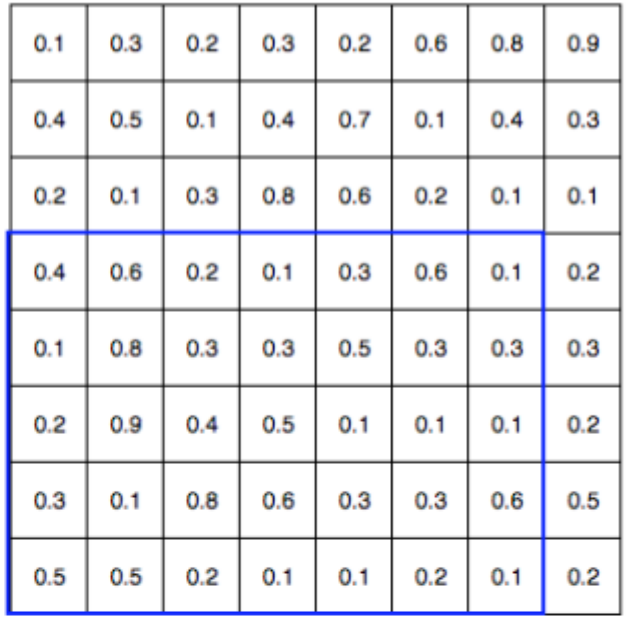
Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

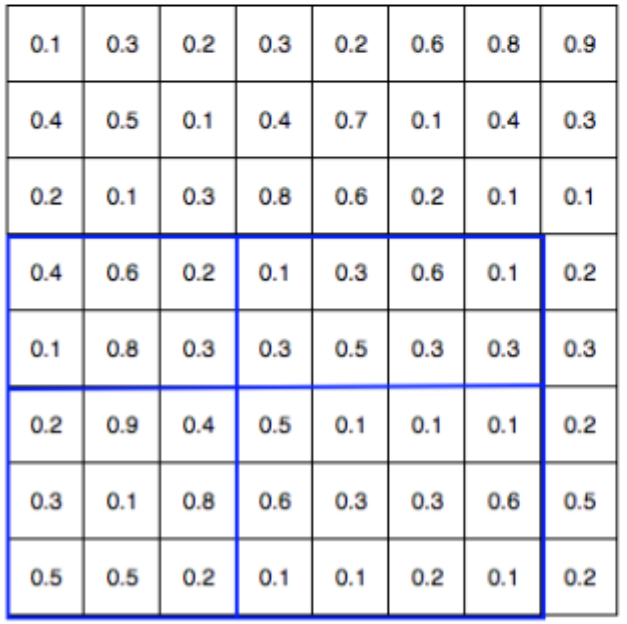
ROI Pooling

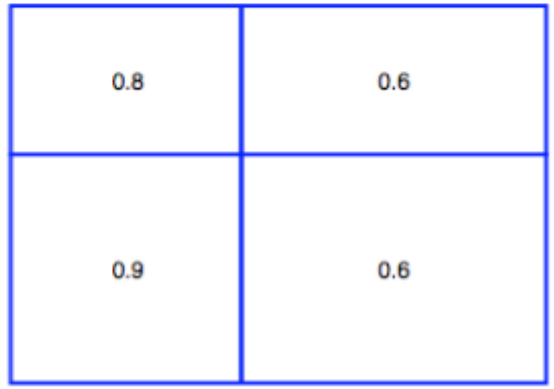
Transforming 8 × 8 feature maps into a predefined 2 × 2 shape

Analogus to max pooling with slight adjustments

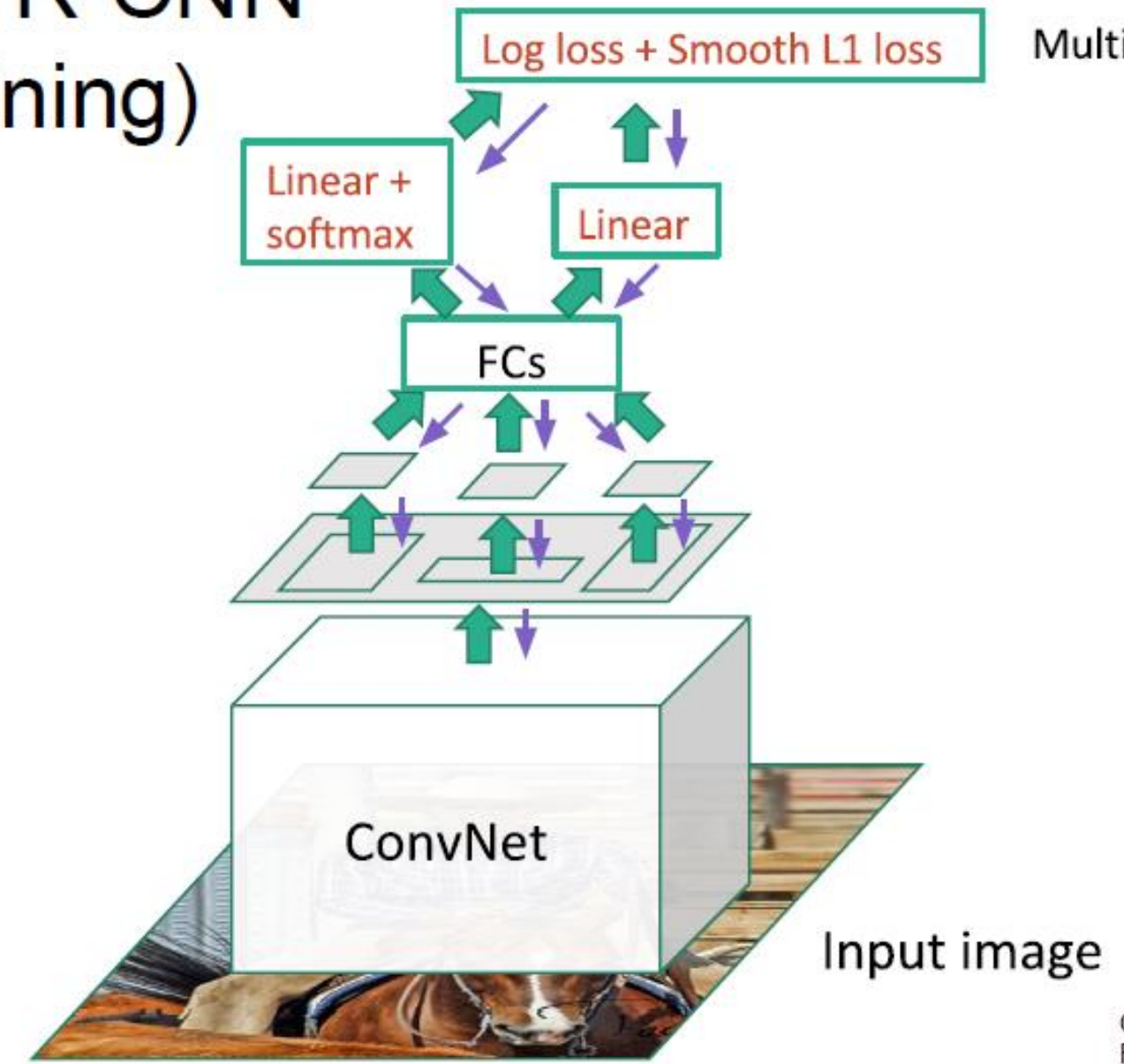
0.1	0.3	0.2	0.3	0.2	0.6	0.8	0.9
0.4	0.5	0.1	0.4	0.7	0.1	0.4	0.3
0.2	0.1	0.3	0.8	0.6	0.2	0.1	0.1
0.4	0.6	0.2	0.1	0.3	0.6	0.1	0.2
0.1	0.8	0.3	0.3	0.5	0.3	0.3	0.3
0.2	0.9	0.4	0.5	0.1	0.1	0.1	0.2
0.3	0.1	0.8	0.6	0.3	0.3	0.6	0.5
0.5	0.5	0.2	0.1	0.1	0.2	0.1	0.2







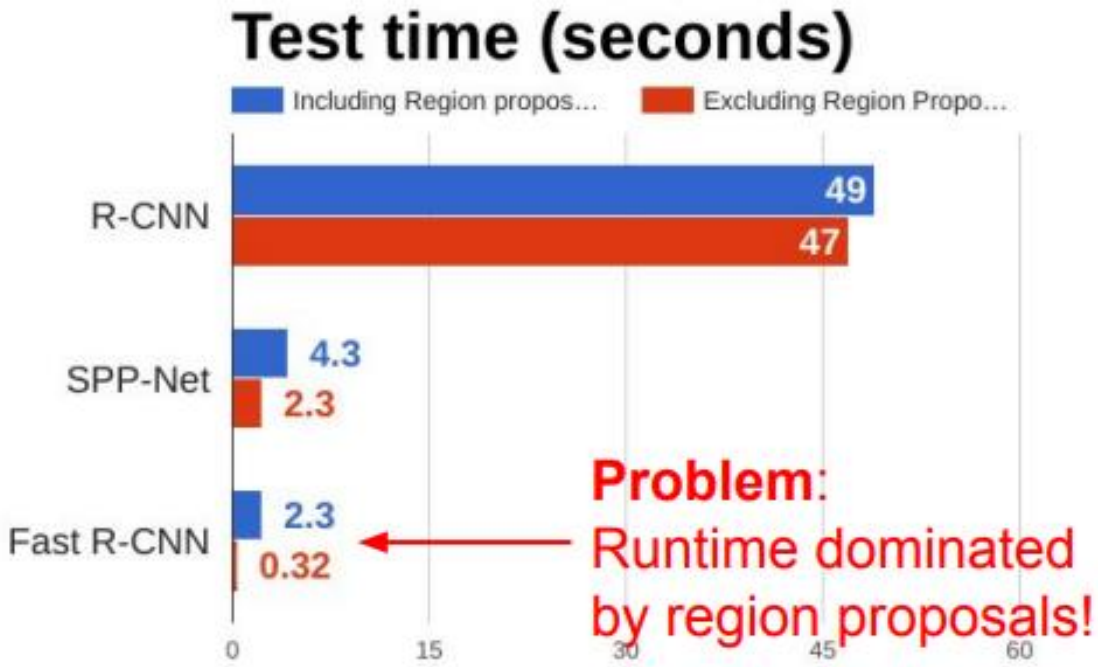
Fast R-CNN (Training)



Multi-task loss

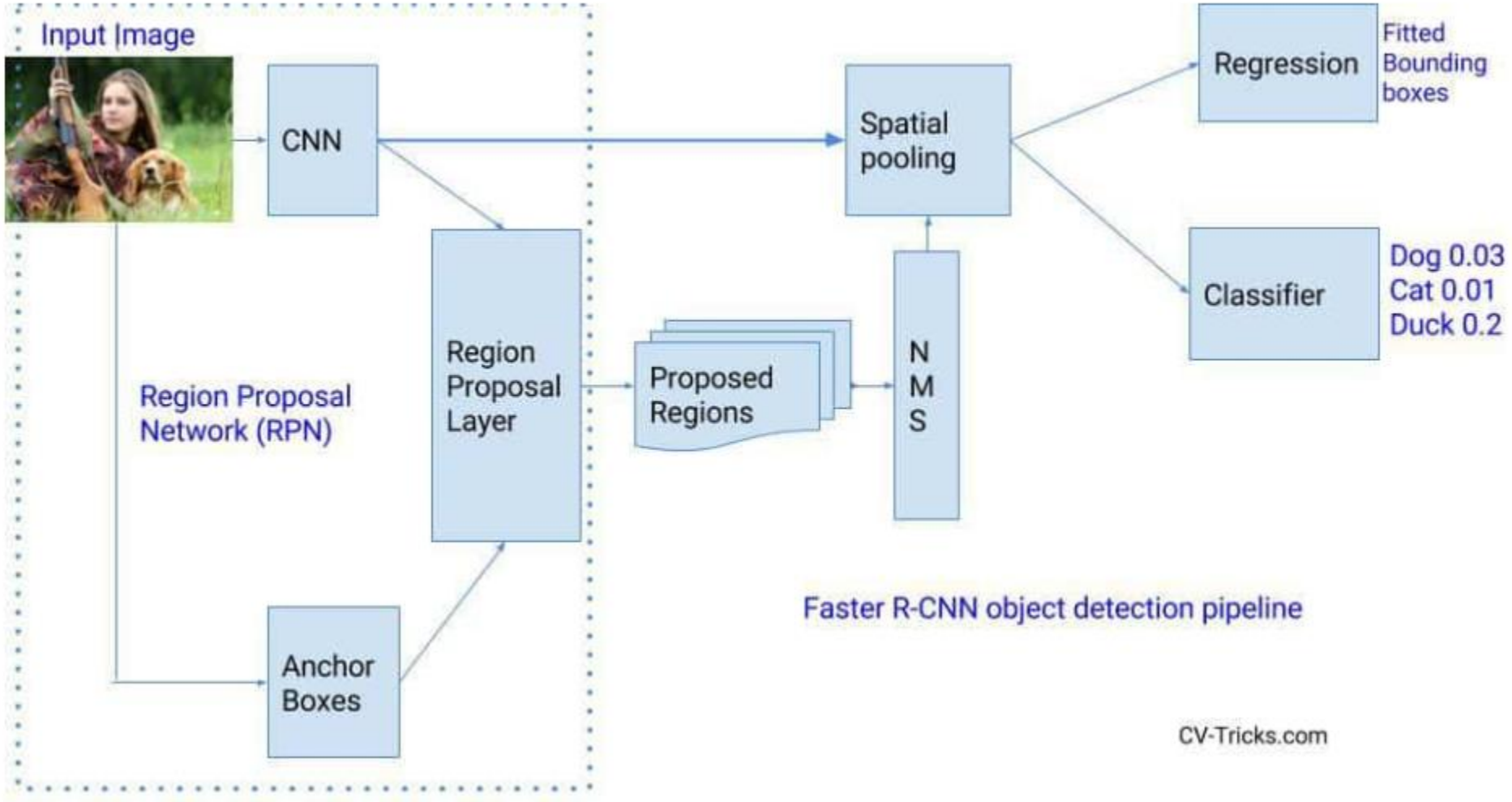
Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN vs SPP vs Fast R-CNN



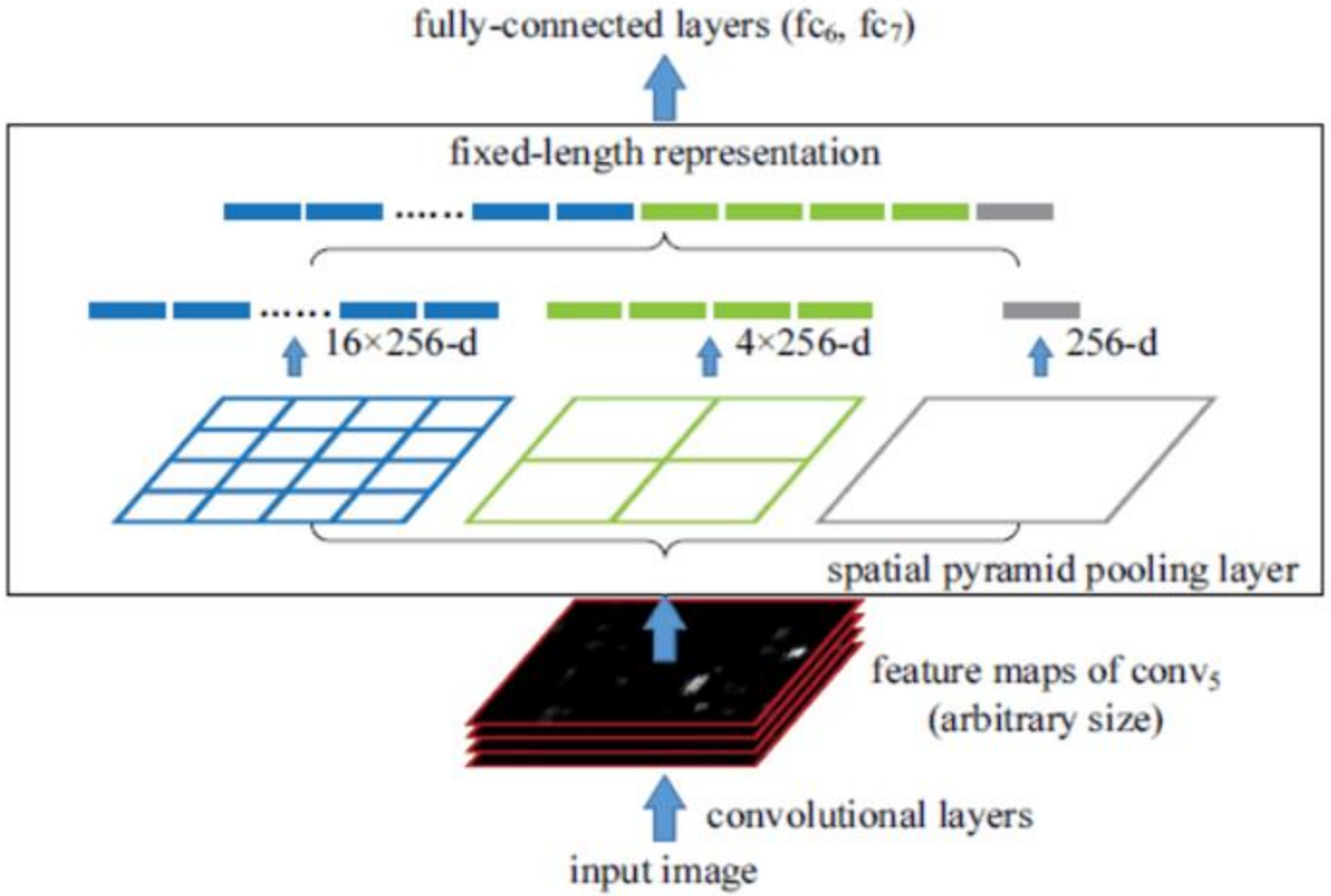
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
He et al, "Spatial pyramid pooling in deep convolutional networks for visual recognition", ECCV 2014
Girshick, "Fast R-CNN", ICCV 2015

Faster R-CNN



https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/

Spatial Pyramid Pooling



- Variable input size/scale
- Spatial pooling after last convolutional layer instead of max-pooling
- SPP layer divides a region into constant number of bins
- Constant vector size is produced
- Fixed length representation
- To generate fixed size of input for the FC layers

https://arxiv.org/pdf/1406.4729.pdf

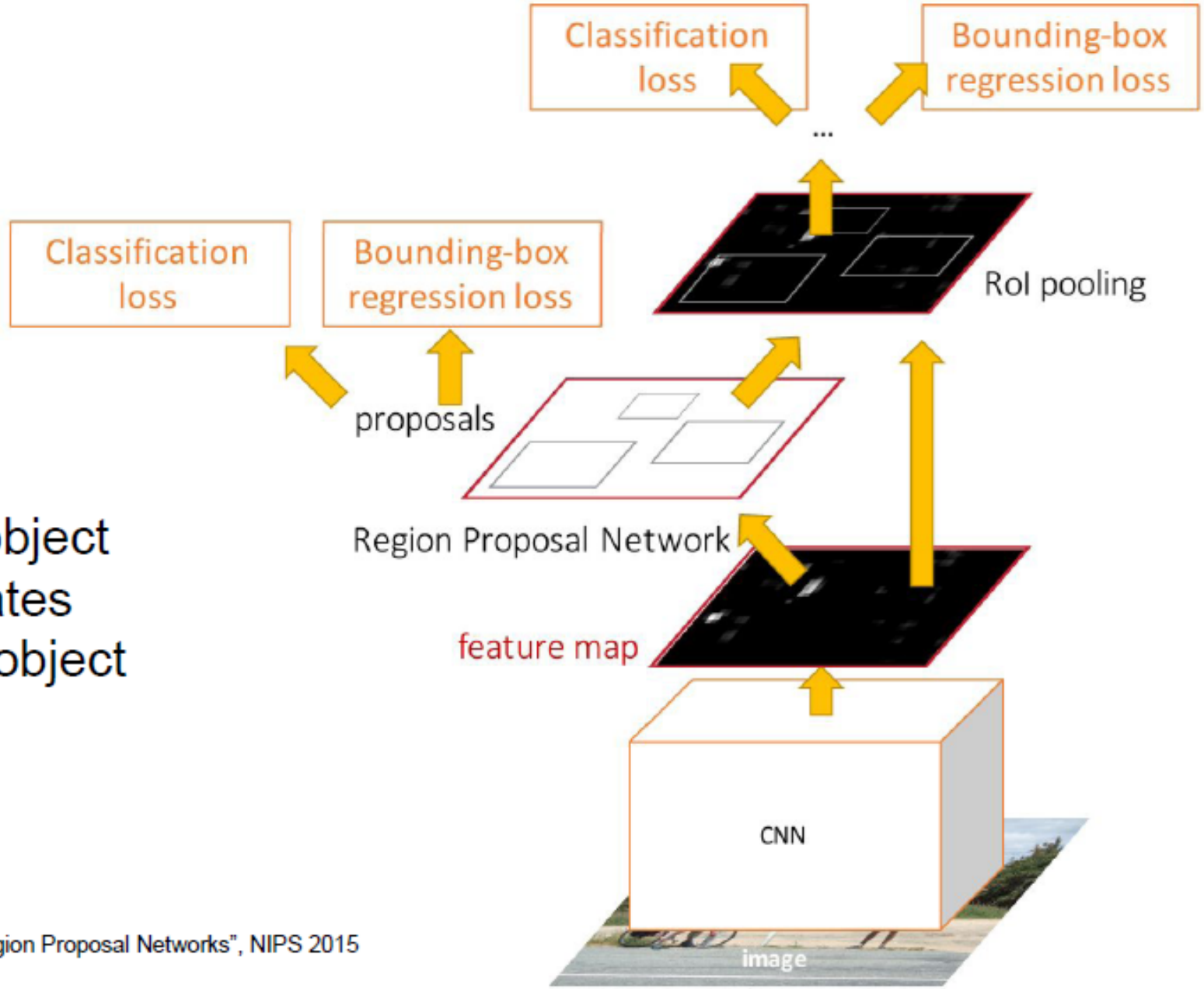
Faster R-CNN:

Make CNN do proposals!

Insert **Region Proposal Network (RPN)** to predict proposals from features

Jointly train with 4 losses:

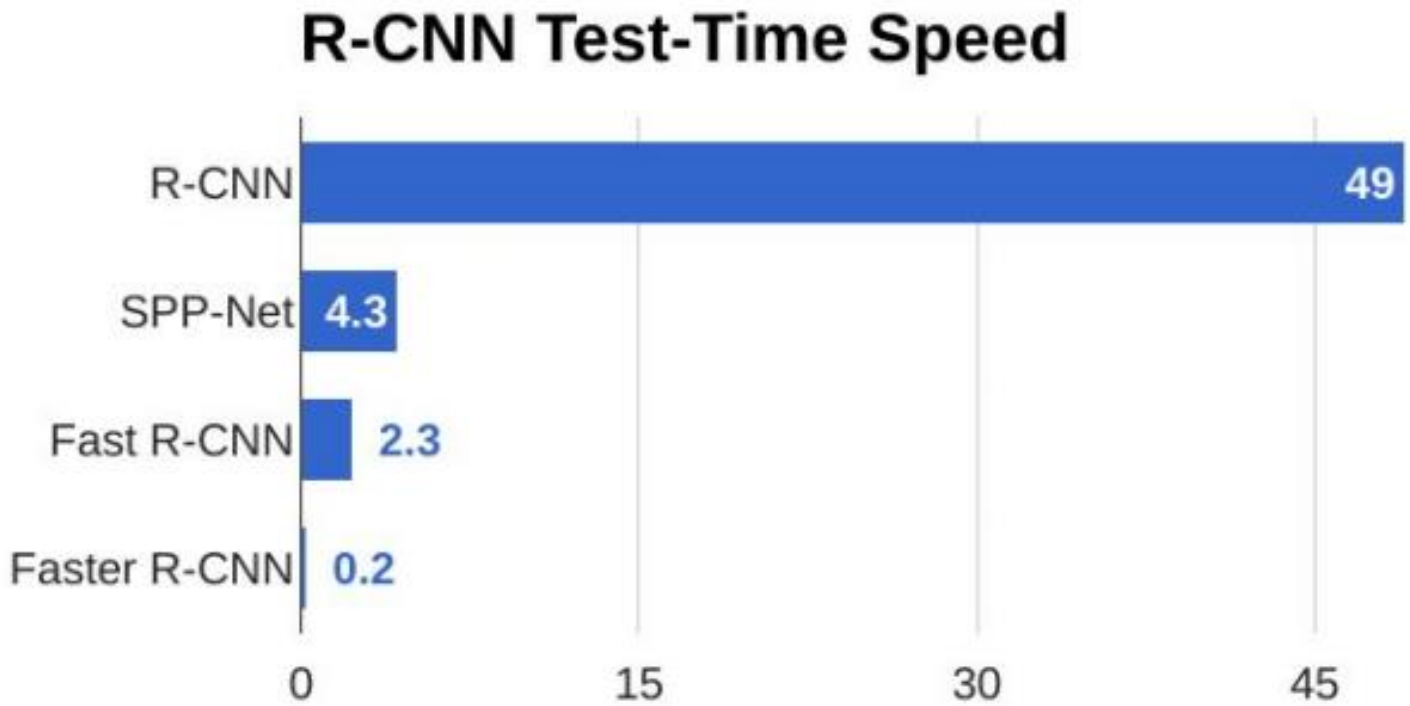
1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates



Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015
Figure copyright 2015, Ross Girshick; reproduced with permission

Faster R-CNN:

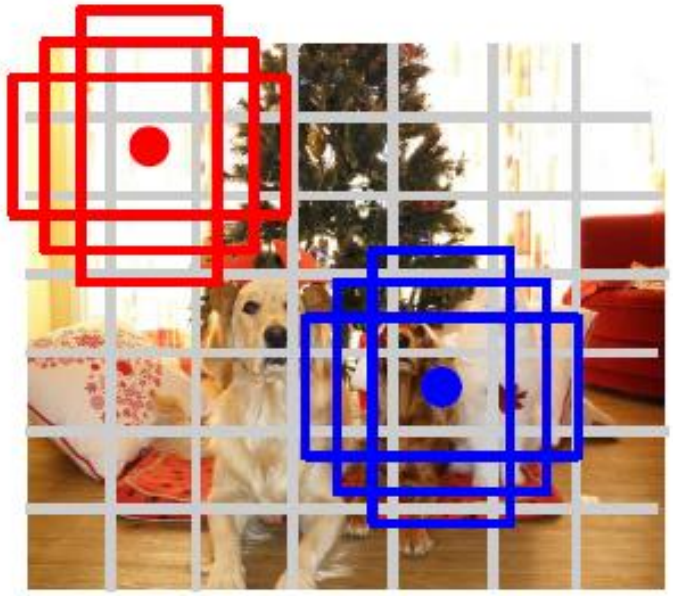
Make CNN do proposals!



Detection without Proposals: YOLO / SSD

Go from input image to tensor of scores with one big convolutional network!

Input image
3 x H x W



Divide image into grid
7 x 7

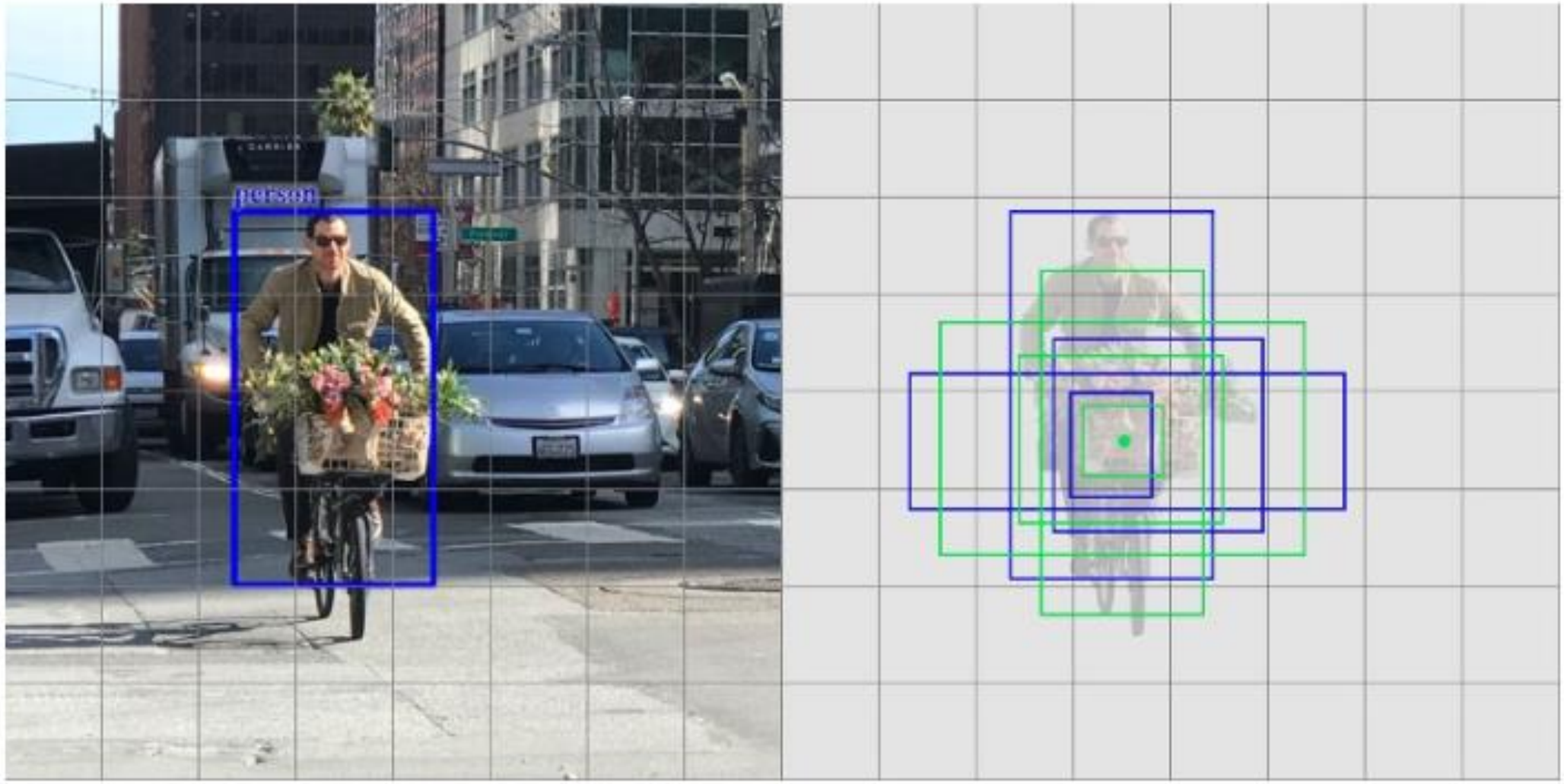
Image a set of **base boxes** centered at each grid cell
Here B = 3

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)

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



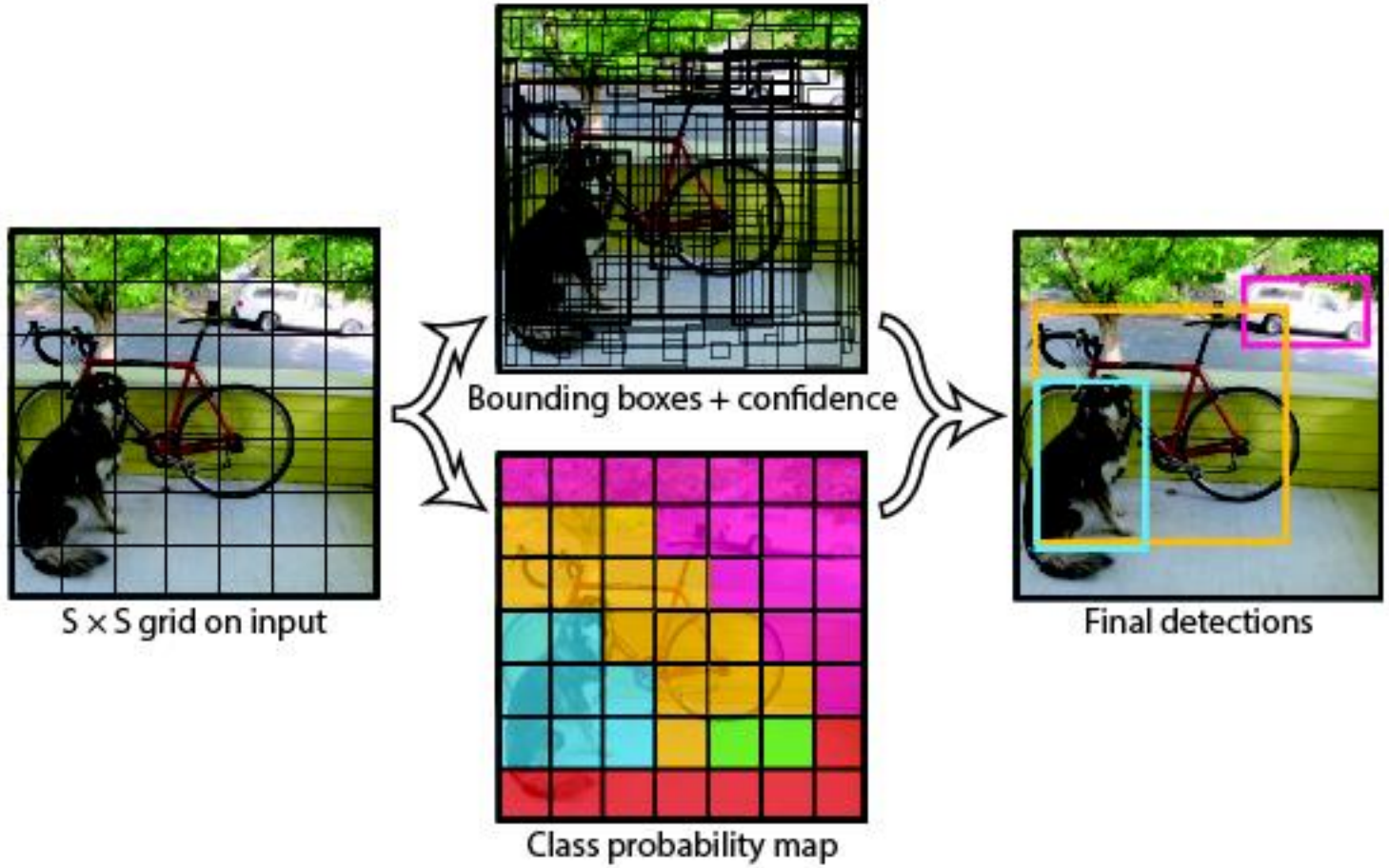
4 predictions each relative to an anchor

YOLO

- YOLO – You Only Look Once ; looks at the image just once but in a clever way

"We reframe the object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities"

- Algorithmic flow :
  - Actually Divides the image into a **grid** of say, 13*13 cells (S=13)
  - Each of these cells is responsible for predicting 5 bounding boxes (B=5) (A **bounding box** describes the rectangle that encloses an object)
  - YOLO for each bounding box
    - outputs a **confidence score** that tells us how good the shape of the box is
    - the **cell also predicts a class**
  - The **confidence score of bounding box and class prediction** are combined into final score -> probability that this bounding box contains a



system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

YOLO details

- 13*13 = 169 grid cells
- Each cell predicts 5 bounding boxes
- 169*5 = 845 bounding boxes
- Most of the boxes have low confidence score.
- Threshold of 30% or more -> 3
- Input image: 416*416 pixels resized
- 13*13*125 tensor describing the bounding boxes for grid cells
- YoLO v2 vs v1 – faster version

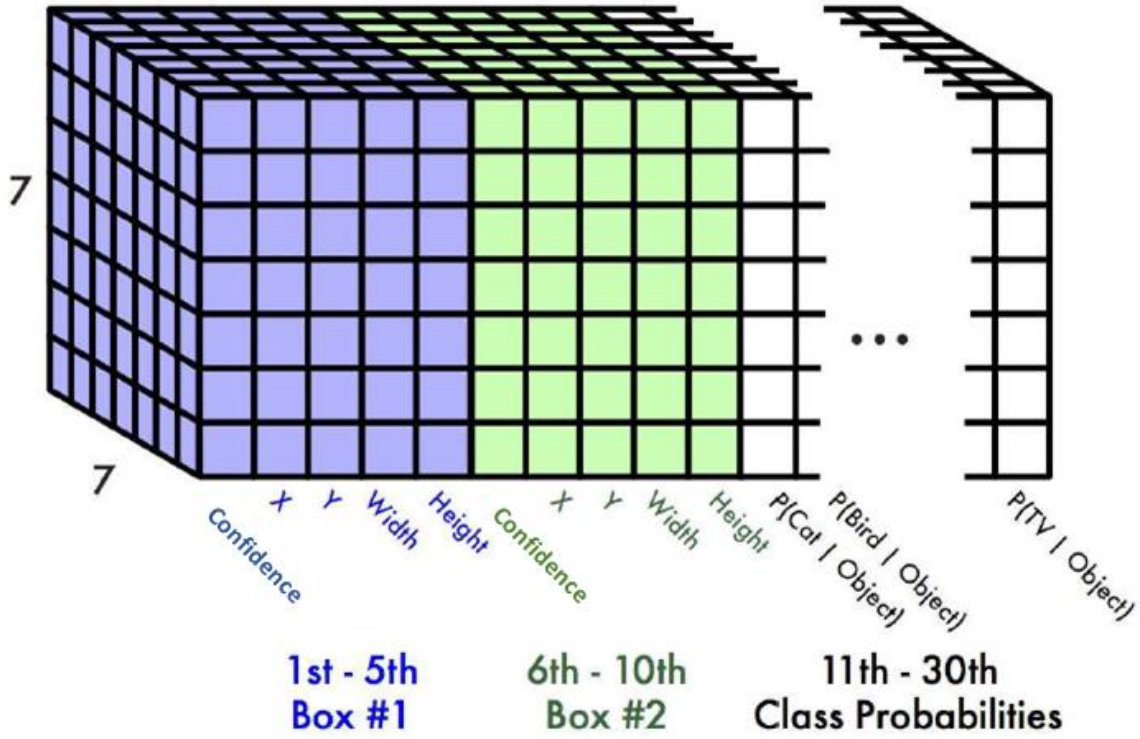
YOLO Prediction Vector Details

- **The input image is divided into an S×S grid (S=7).** If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object.
- **Each grid cell predicts B bounding boxes (B=2) and confidence scores for those boxes.** These confidence scores reflect how confident the model is that the box contains an object, **i.e. any objects in the box, P(Objects).**
- **Each bounding box consists of 5 predictions: x, y, w, h, and confidence.**
- The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell.
- The width w and height h are predicted relative to the whole image.
- The confidence represents the Intersection Over Union (IOU) between the predicted box and any ground truth box.

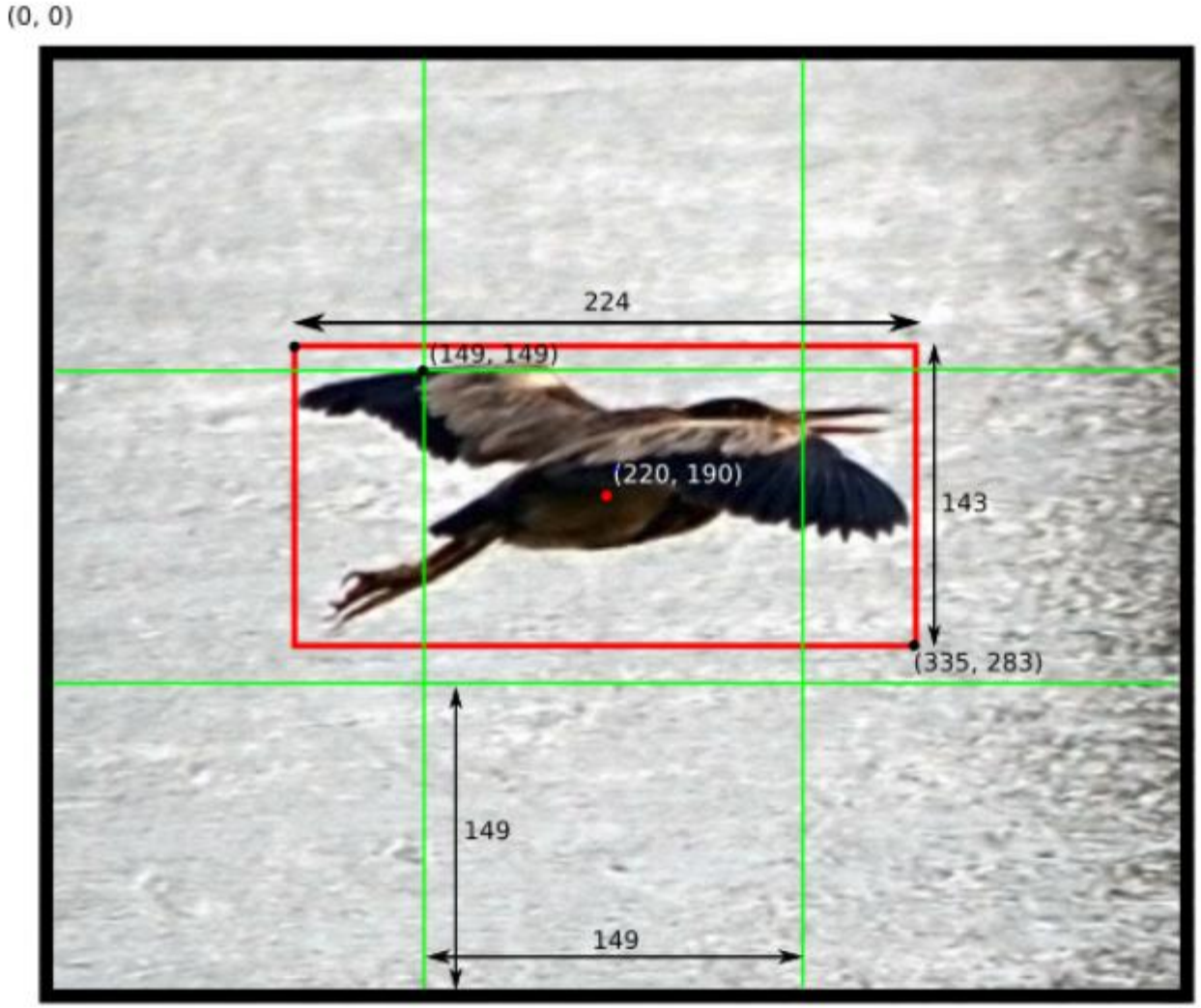
YOLO Bounding boxes

- Each grid cell also predicts conditional class probabilities, P(Classi|Object). (Total number of classes=20)

- **The output size becomes: 7×7×(2×5+20)=1470**



Encode Bounding Box



x & y bounded between 0 & 1

x = (220-149) / 149 = 0.48

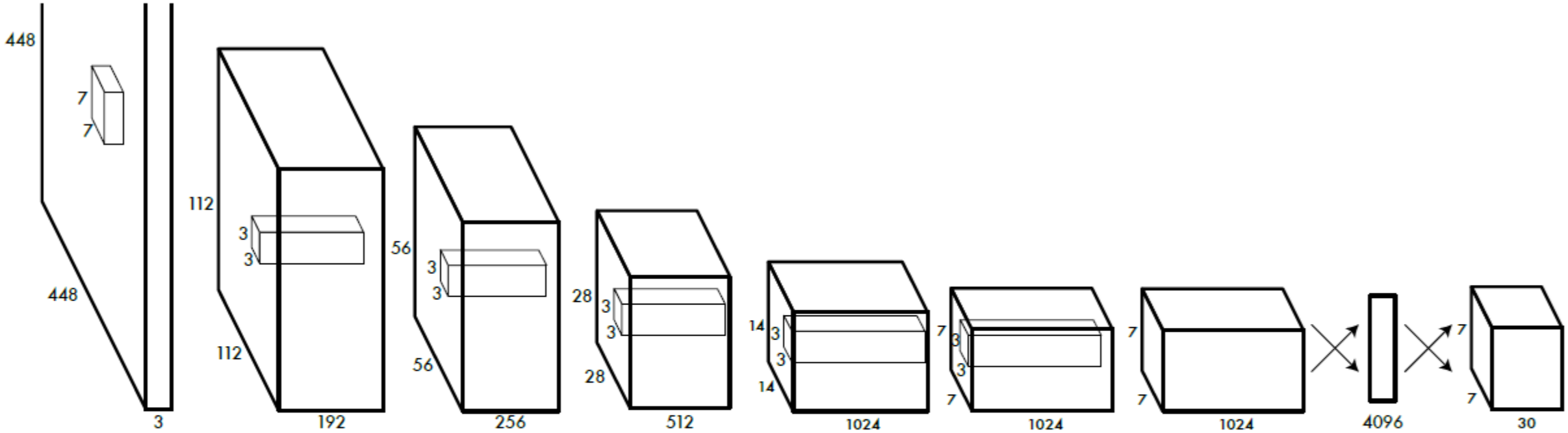
y = (190-149) / 149 = 0.28

w = 224 / 448 = 0.50

h = 143 / 448 = 0.32

https://hackernoon.com/understanding-yolo-f5a74bbc7967

YOLO Network Architecture



Conv. Layer
7x7x64-s-2
Maxpool Layer
2x2-s-2

Conv. Layer
3x3x192
Maxpool Layer
2x2-s-2

Conv. Layers
1x1x128
3x3x256
1x1x256
3x3x512
Maxpool Layer
2x2-s-2

Conv. Layers
1x1x256
3x3x512
} ×4
1x1x512
3x3x1024
Maxpool Layer
2x2-s-2

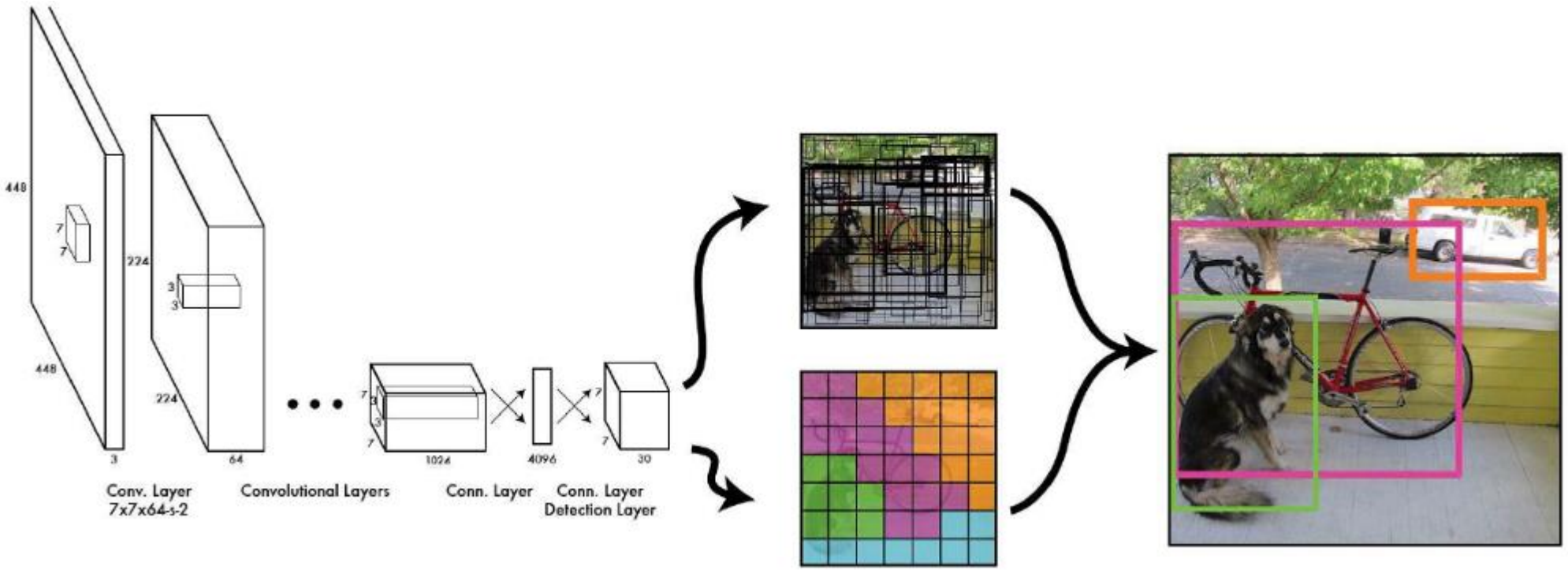
Conv. Layers
1x1x512
3x3x1024
} ×2
3x3x1024
3x3x1024-s-2

Conv. Layers
3x3x1024
3x3x1024

Conn. Layer

Conn. Layer

Whole YOLO Network Pipeline

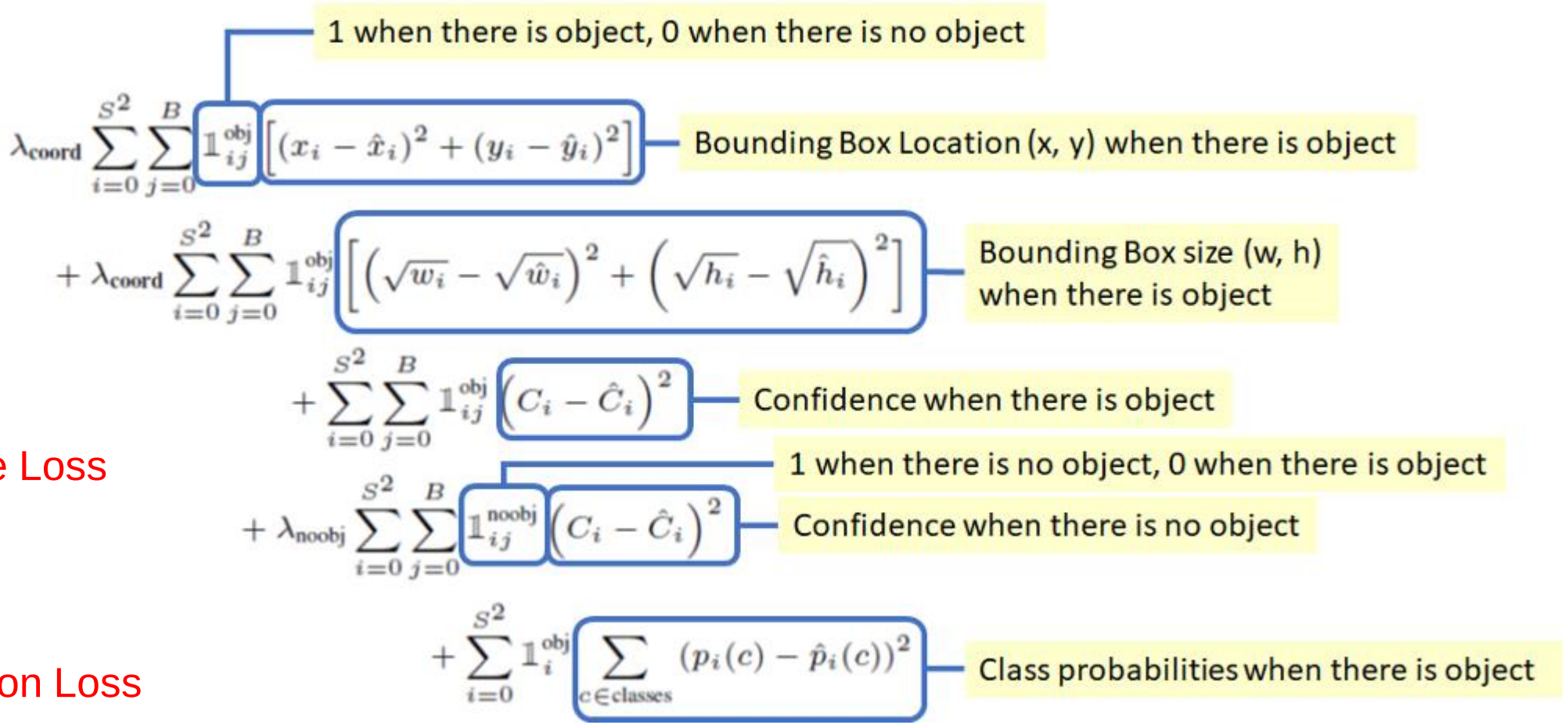


Loss function in YOLO

Localization Loss

Confidence Loss

Classification Loss



Loss Function in YOLO

- **1st term (x, y)**: The **bounding box x and y coordinates** is parametrized to be offsets of a particular grid cell location so they are also bounded between 0 and 1.
- Sum of square error (SSE) is estimated only when there is object.

- **2nd term (w, h)**: The **bounding box width and height** are normalized by the image width and height so that they fall between 0 and 1.
- SSE is estimated only when there is object.

Since small deviations in large boxes matter less than in small boxes. **square root of the bounding box width w and height h** instead of the width and height directly to partially address this problem.

Loss function in YOLO

- **3rd term and 4th term (The confidence)**
- The IOU between the predicted box and any ground truth box
- In every image many grid cells do not contain any object. This pushes the "confidence" scores of those cells towards zero, often overpowering the gradient from cells that do contain objects, and makes the model unstable.
- The loss from confidence predictions for boxes that don't contain objects, is decreased, i.e. **λ_{noobj}**=0.5.

- **5th term (Class Probabilities)**: SSE of class probabilities when there is objects.

https://www.jeremyjordan.me/object-detection-one-stage/

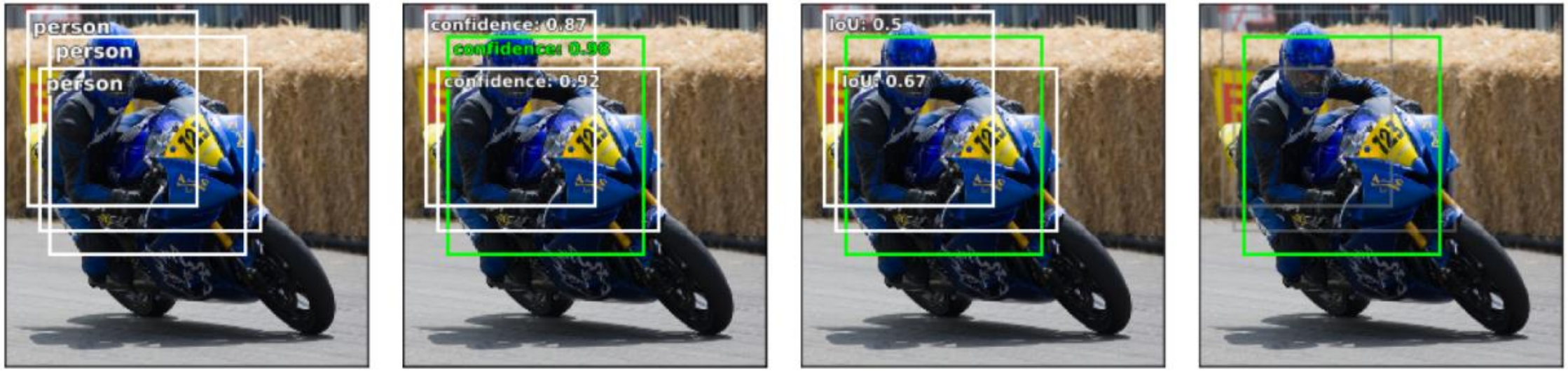
greatlearning
Learning for Life

Remove Duplicate Detections in Yolo : Non Maximal Suppression

Improves score by 2-3 mAP

Repeat with next highest confidence prediction until no more boxes are being suppressed

For each class...



After filtering out low confidence predictions, we may still be left with ***redundant*** **detections**

Select the bounding box prediction with the **highest confidence**

Calculate the IoU between the **selected box** and all remaining predictions

Remove any boxes which have an IoU score above some defined threshold

YoLO implementation

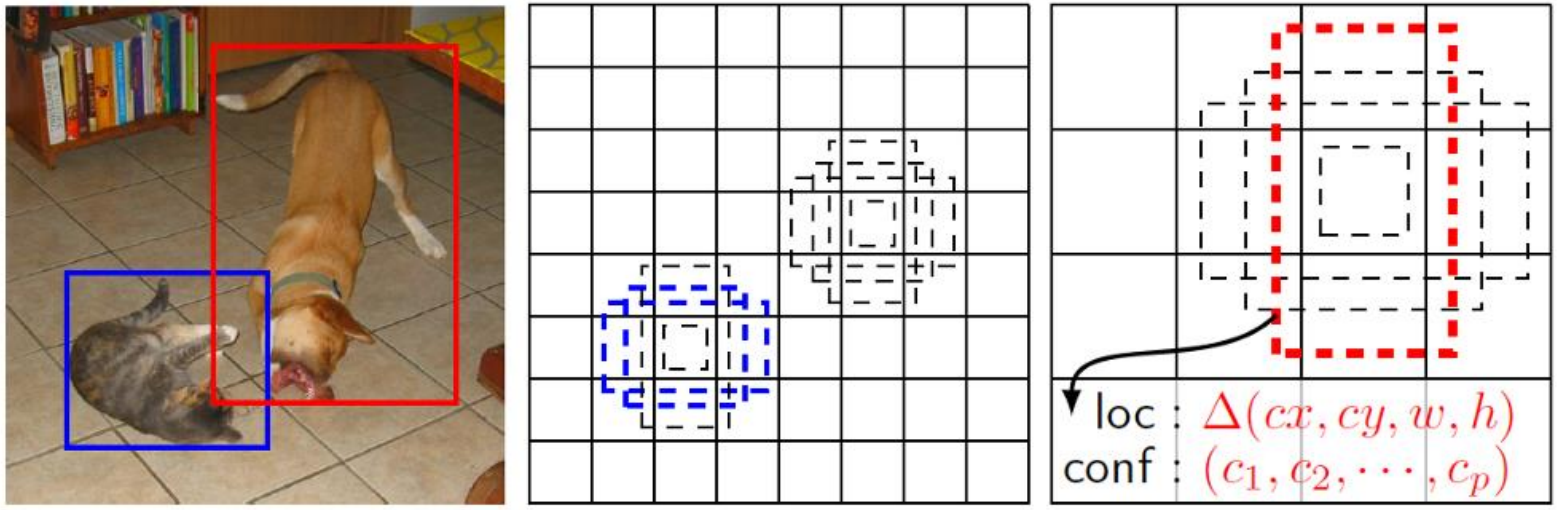
- Written in darknet in C
- Darkflow – tensorflow version of darknet
- Cython (like Jython)
- Use pretrained version of Darkflow

Single Shot Detection (SSD)

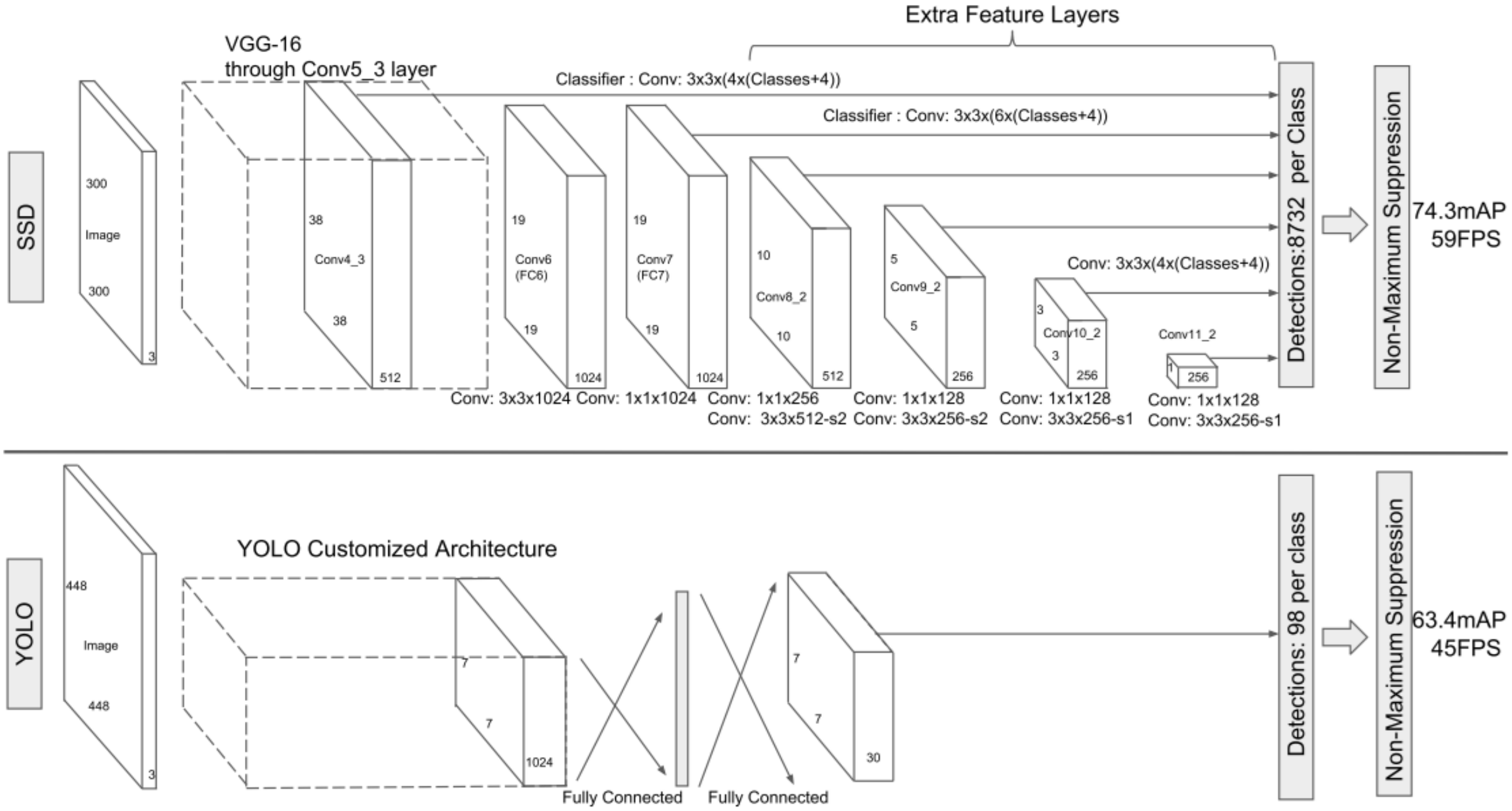
- By using SSD, we only need to **take one single shot to detect multiple objects within the image**
- Regional proposal network (RPN) based approaches such as R-CNN, Fast R-CNN series need two shots, one for generating region proposals, one for detecting the object of each proposal.
- SSD is much faster compared with two-shot RPN-based approaches.
- The loss function consists of two terms: Lconf (confidence loss) and Lloc (localization loss)

SSD

- **A feature layer of size *m*×*n* (number of locations) with *p* channels**
- **For each location, we got *k* bounding boxes**
- **For each of the bounding box, we will compute *c* class scores and 4 offsets relative to the original default bounding box shape.**
- Thus, we got **(*c*+4) *kmn* outputs.**



(a) Image with GT boxes (b) 8 × 8 feature map (c) 4 × 4 feature map


SSD
300
Image
300
3
VGG-16
through Conv5_3 layer
38
Conv4_3
38
512
19
Conv6
(FC6)
19
1024
19
Conv7
(FC7)
19
1024
10
Conv8_2
10
512
5
Conv9_2
5
256
3
Conv10_2
3
256
Conv11_2
1
256
Extra Feature Layers
Classifier : Conv: 3x3x(4x(Classes+4))
Classifier : Conv: 3x3x(6x(Classes+4))
Conv: 3x3x(4x(Classes+4))
Conv: 3x3x1024
Conv: 1x1x1024
Conv: 1x1x256
Conv: 3x3x512-s2
Conv: 1x1x128
Conv: 3x3x256-s2
Conv: 1x1x128
Conv: 3x3x256-s1
Conv: 1x1x128
Conv: 3x3x256-s1
Detections:8732 per Class
Non-Maximum Suppression
74.3mAP
59FPS
YOLO
448
Image
448
3
YOLO Customized Architecture
7
7
1024
Fully Connected
Fully Connected
7
7
30
Detections: 98 per class
Non-Maximum Suppression
63.4mAP
45FPS

SSD Layers and Boxes

- SSD predicts bounding boxes after multiple convolutional layers. Since each convolutional layer operates at a different scale, it is able to **detect objects of various scales.**
- **SSD has bounding boxes which is more than that of YOLO**

SSD Layers and Boxes

- To have more accurate detection, different layers of feature maps are also going through a small 3×3 convolution for object detection.
- For example, **at Conv4_3, it is of size 38×38×512. 3×3 conv is applied.**
- And there are **4 bounding boxes** and **each bounding box will have (classes + 4) outputs**.
- Thus, at Conv4_3, the output is 38×38×4×(*c*+4). Suppose there are 20 object classes plus one background class, the output is 38×38×4×(21+4) = 144,400.
- **In terms of number of bounding boxes, there are 38×38×4 = 5776 bounding boxes.**

Object Detection: Lots of variables

Base Network

VGG16

ResNet-101

Inception V2

Inception V3

Inception

ResNet

MobileNet

Object Detection architecture

Faster R-CNN

R-FCN

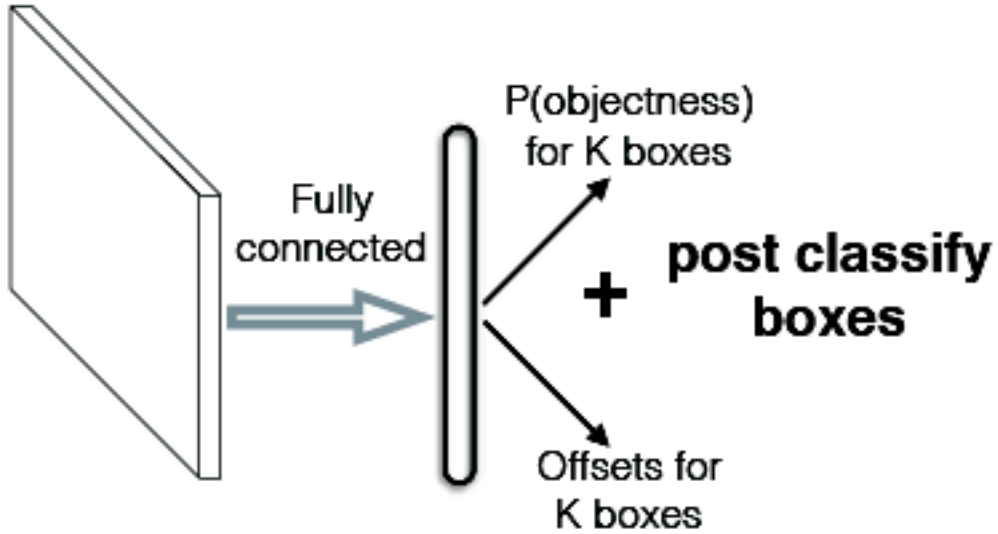
SSD

Image Size

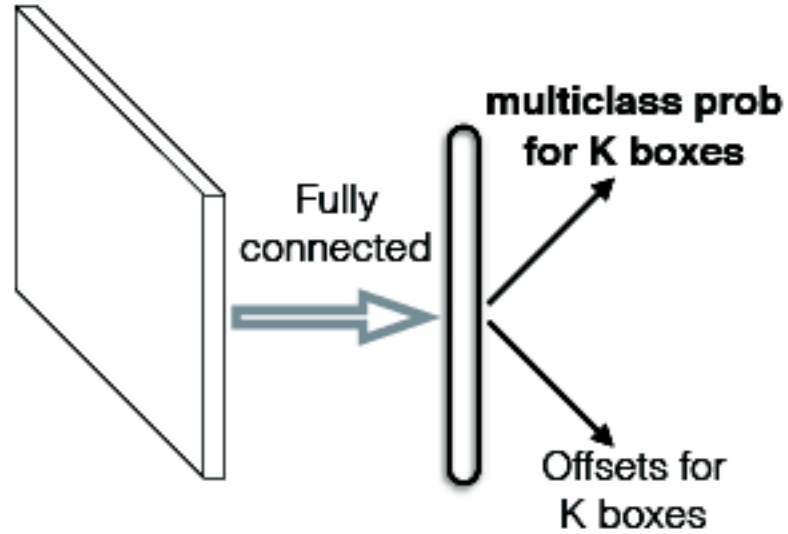
 # Region Proposals

...

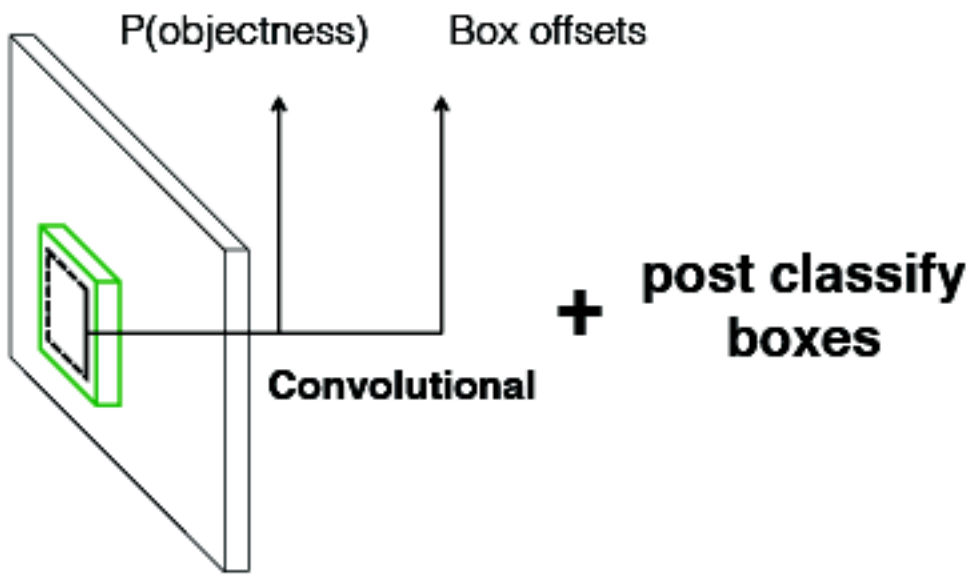
MultiBox [Erhan et al. CVPR14]



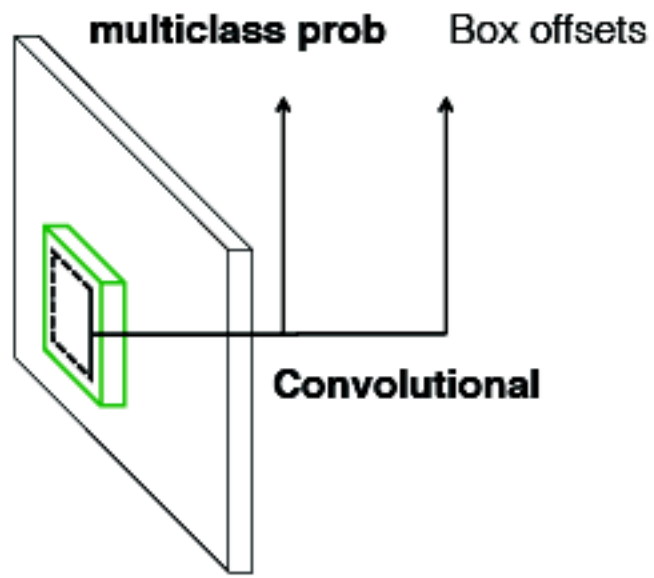
YOLO [Redmon et al. CVPR16]



Faster R-CNN [Ren et al. NIPS15]

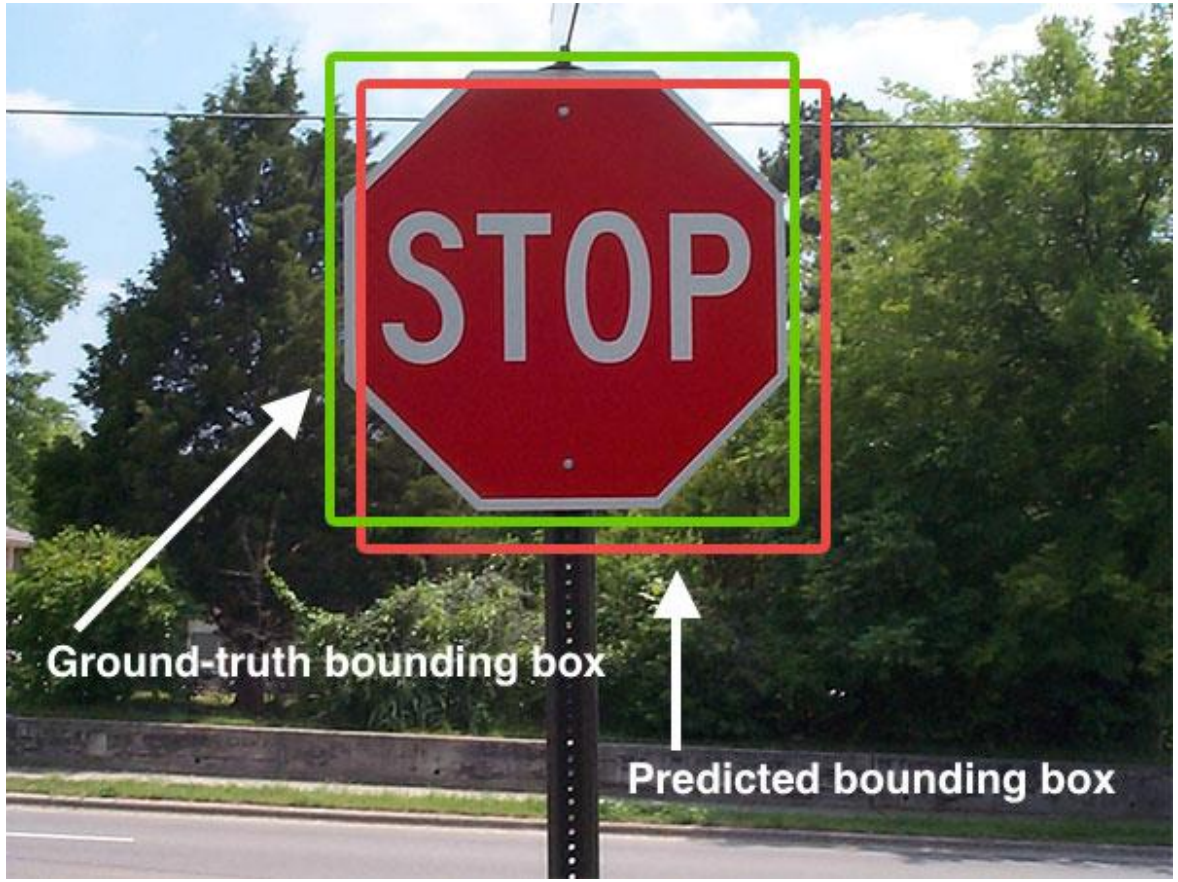


SSD



Loss function: Intersection over Union (IoU)

An example of detecting a stop sign in an image. The predicted bounding box is drawn in red while the ground-truth bounding box is drawn in green. Our goal is to compute the Intersection of Union between these bounding box.



Souce: PyImageSearch.com

What is Intersection over Union?

- Intersection over Union is an evaluation metric **used to measure the accuracy of an object detector on a particular dataset**.
- We often see this evaluation metric used in object detection challenges such as the popular PASCAL VOC challenge.

- Intersection over Union is simply an ***evaluation metric***. Any algorithm that provides predicted bounding boxes as output can be evaluated using IoU.

- More formally, in order to apply Intersection over Union to evaluate an (arbitrary) object detector we need:
  - The ***ground-truth bounding boxes*** (i.e., the hand labeled bounding boxes from the testing set that specify *where* in the image our object is).
  - The ***predicted bounding boxes*** from our model.

- As long as we have these two sets of bounding boxes we can apply Intersection over Union.

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

Computing the Intersection of Union is as simple as dividing the area of overlap between the bounding boxes by the area of union.

In the numerator we compute the ***area of overlap*** between the *predicted* bounding box and the *ground-truth* bounding box.

The denominator is the ***area of union***, or more simply, the area encompassed by *both* the predicted bounding box and the ground-truth bounding box.

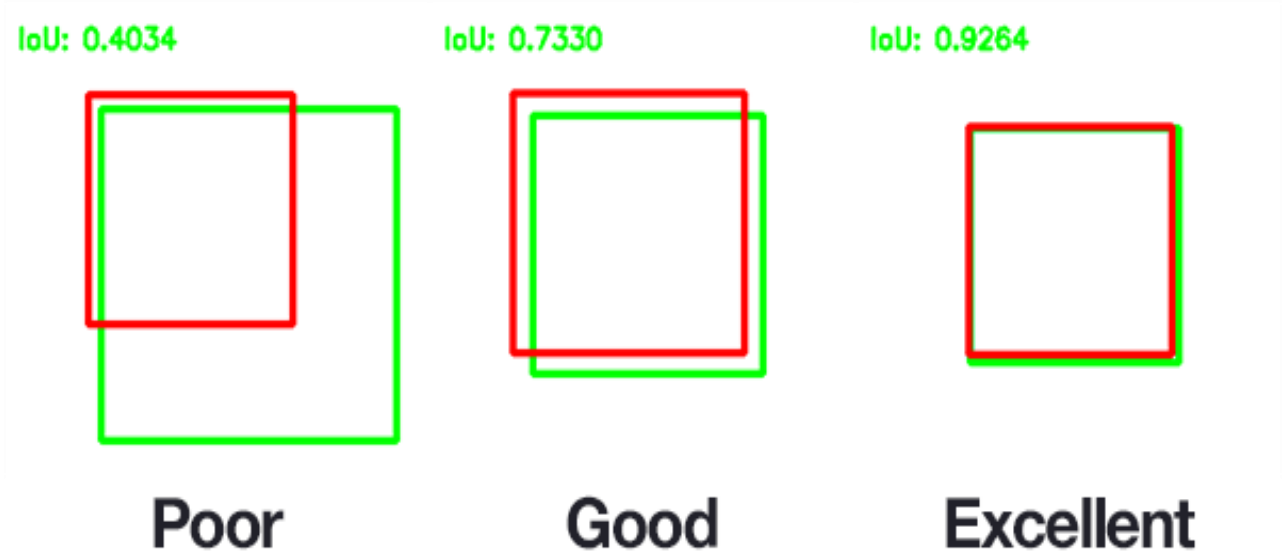
Dividing the area of overlap by the area of union yields our final score — *the Intersection over Union.*

An Intersection over Union score > *0.5* is normally considered a "good" prediction.

Why do we use Intersection over Union?

Binary classification makes computing accuracy straightforward; however, for object detection it's not so simple.

In all reality, it's *extremely unlikely* that the *(x, y)*-coordinates of our predicted bounding box are going to ***exactly match*** the *(x, y)*-coordinates of the ground-truth bounding box.



Predicted bounding boxes that heavily overlap with the ground-truth bounding boxes have higher scores than those with less overlap.

This makes Intersection over Union an excellent **metric for evaluating custom object detectors.**

References

- https://www.mpi-inf.mpg.de/fileadmin/inf/d2/HLCV/cv-ss13-0605-sliding-window.pdf
- https://www.youtube.com/watch?v=nDPWywWRIRo
- https://arthurdouillard.com/post/selective-search/
- https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4
- https://medium.freecodecamp.org/mask-r-cnn-explained-7f82bec890e3
- https://towardsdatascience.com/review-ssd-single-shot-detector-object-detection-851a94607d11
- https://towardsdatascience.com/yolov1-you-only-look-once-object-detection-e1f3ffec8a89
- https://medium.com/mlreview/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-e0f514068807
- https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/
- https://arxiv.org/pdf/1502.05082.pdf
- https://arxiv.org/pdf/1807.05511.pdf (16-Apr-2019)
- https://towardsdatascience.com/review-mobilenetv1-depthwise-separable-convolution-light-weight-model-a382df364b69

Let’s move on to the python notebooks !!

Semantic Segmentation

What is Semantic Segmentation?

Semantic segmentation is a natural step in the progression from coarse to fine inference:

- The origin could be located at **classification**, which consists of making a prediction for a whole input.
- The next step is **localization / detection**, which provide not only the classes but also additional information regarding the spatial location of those classes.
- Finally, **semantic segmentation** achieves fine-grained inference by making dense predictions inferring labels for every pixel, so that each pixel is labeled with the class of its enclosing object ore region.

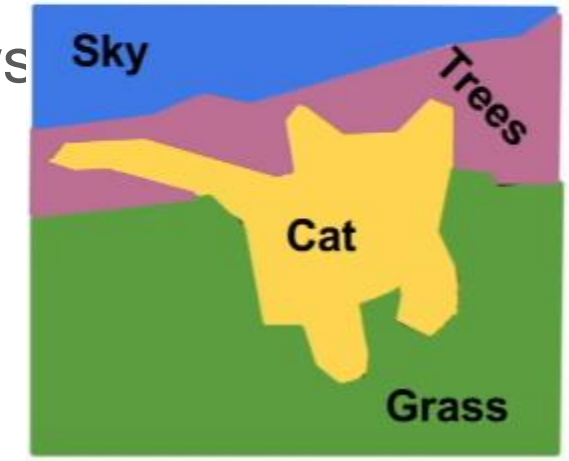
Semantic Segmentation

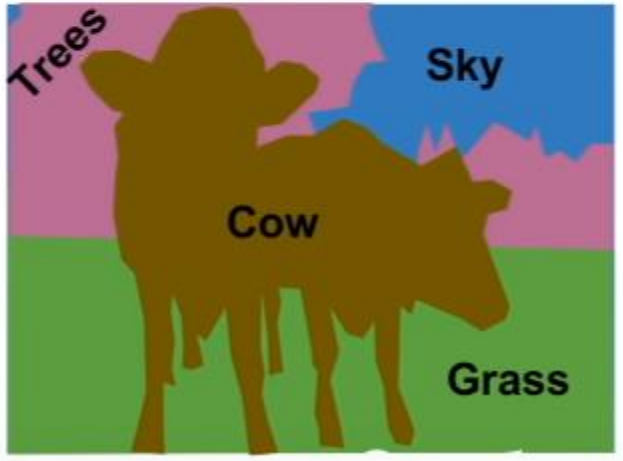
- Spatial information should be preserved
- Classifier information should be preserved

Semantic vs Instance Segmentation

- In Semantic segmentation we only care about individual pixels and not about instances as such.

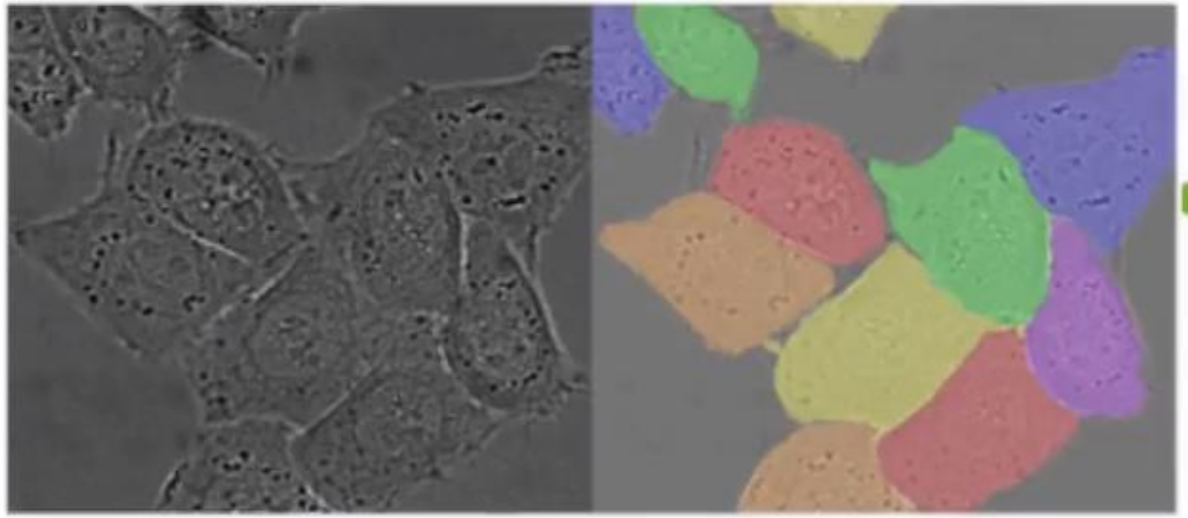
- As in the figure all the areas with cows are marked but 2 cows need not be individually identified in semantic segmentation





Applications

- Self driving cars



- Biomedical Aplications

- Augmented Reality

Semantic Segmentation

Label every pixel!

Don't differentiate instances (cows)

Classic computer vision problem

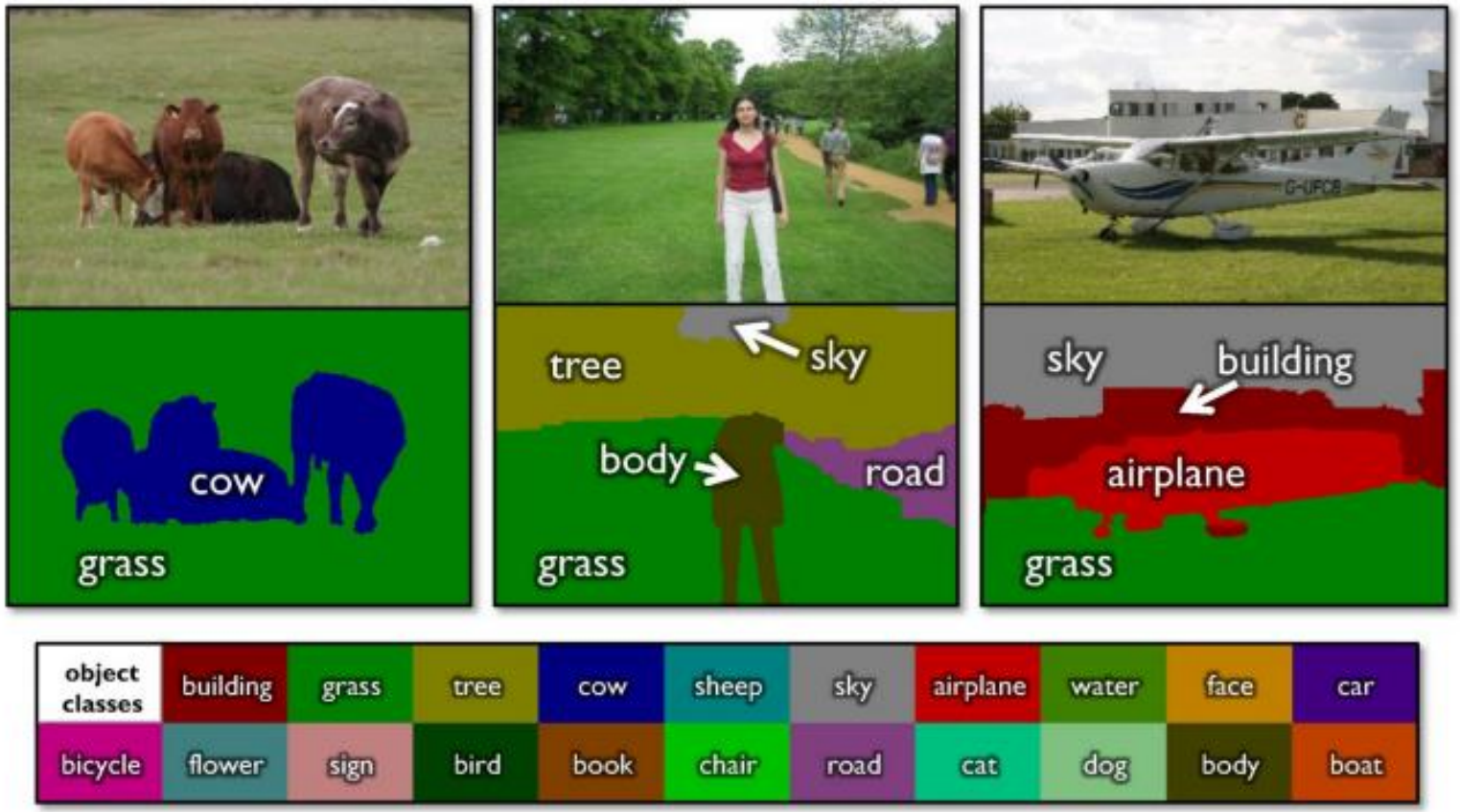


Figure credit: Shotton et al, "TextonBoost for Image Understanding: Multi-Class Object Recognition and Segmentation by Jointly Modeling Texture, Layout, and Context", IJCV 2007

Pixel labels for training images must be known to train for semantic segmentation

void	road	sidewalk	building	wall
fence	pole	traffic light	traffic sign	vegetation
terrain	sky	person	rider	car
truck	bus	train	motorcycle	bicycle

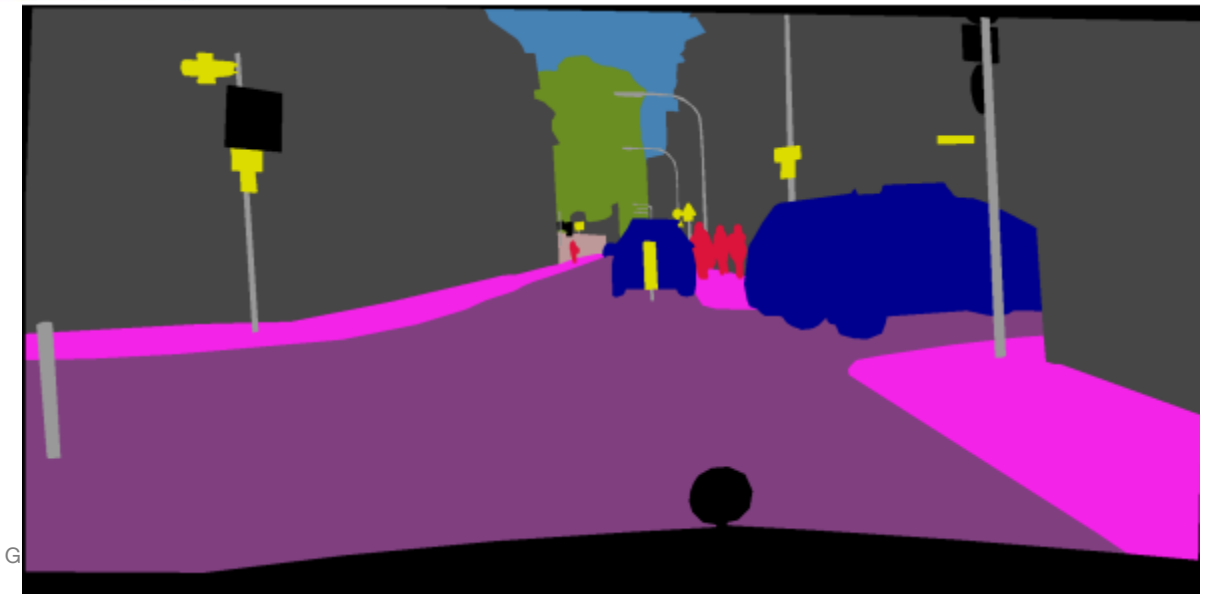
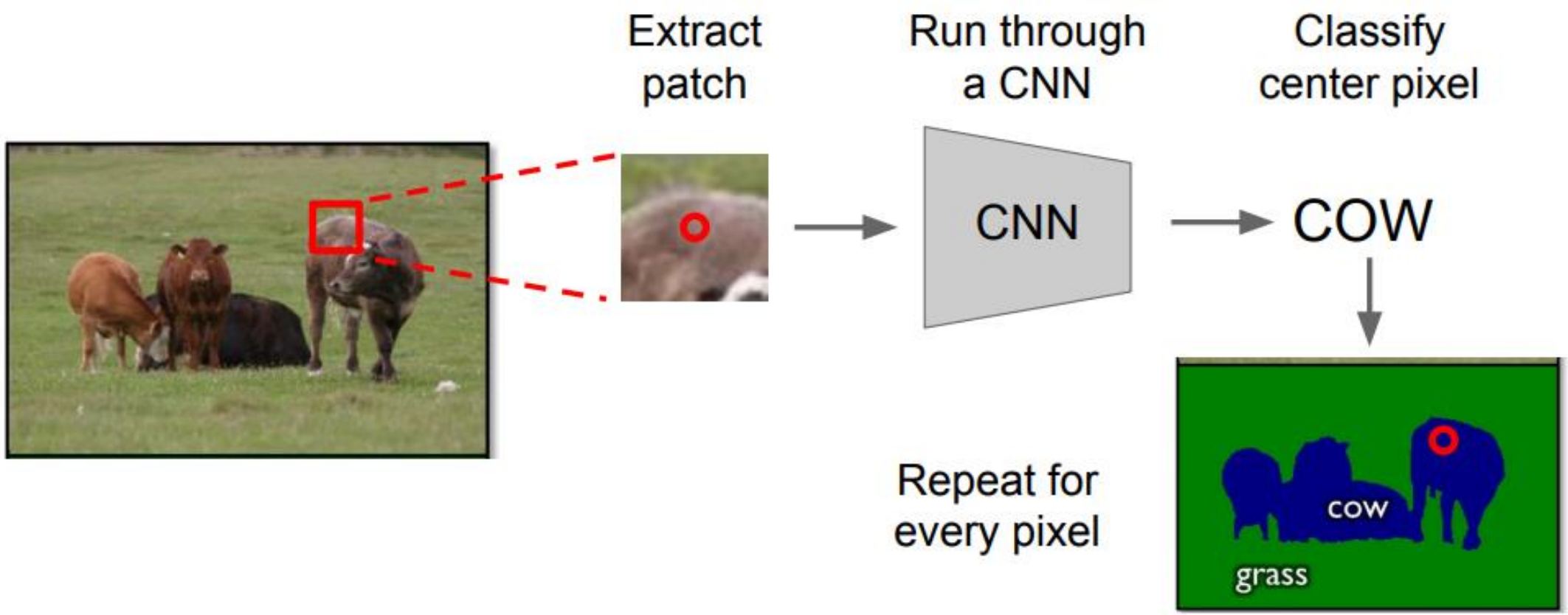
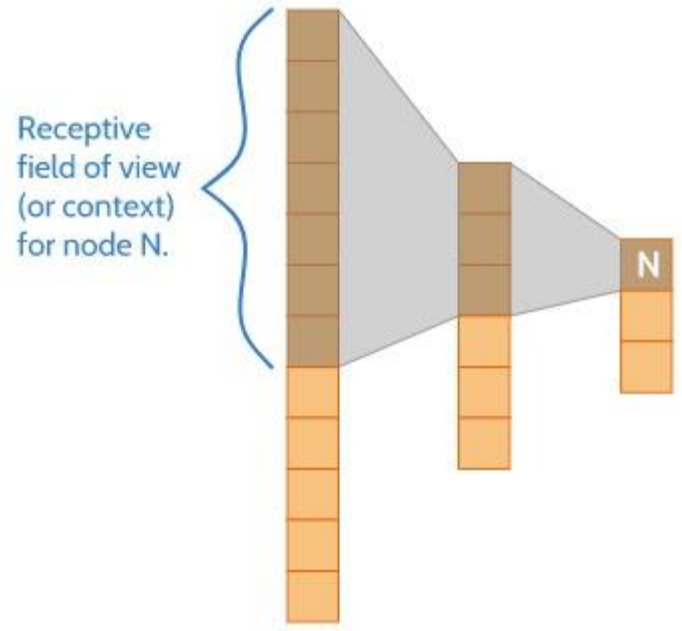


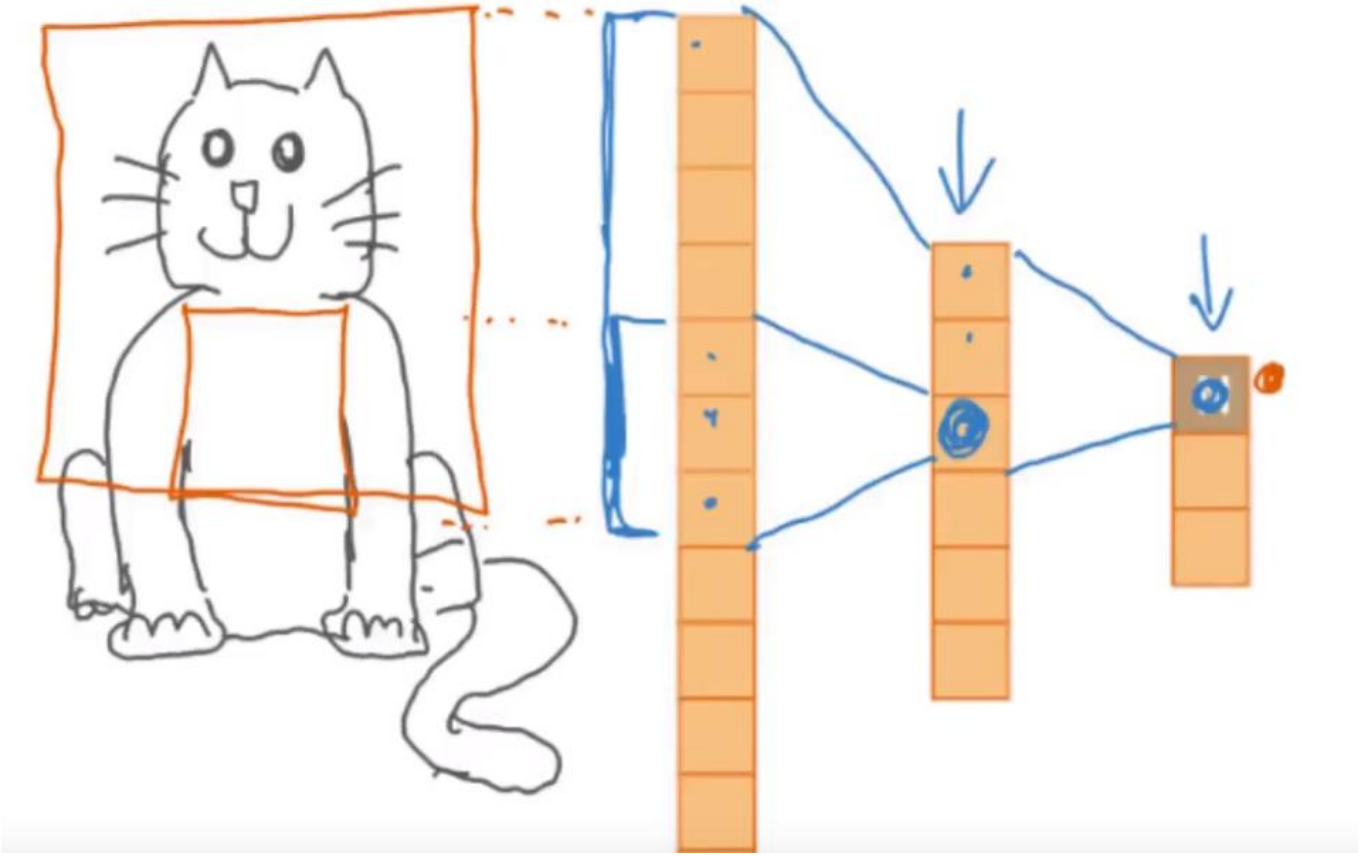
Image Source: "ICNet for Real-Time Semantic Segmentation on High-Resolution Images" Hengshuang Zhao1, Xiaojuan Qi, Xiaoyong Shen, Jianping Shi, Jiaya Jia, ECCV'18

Semantic Segmentation



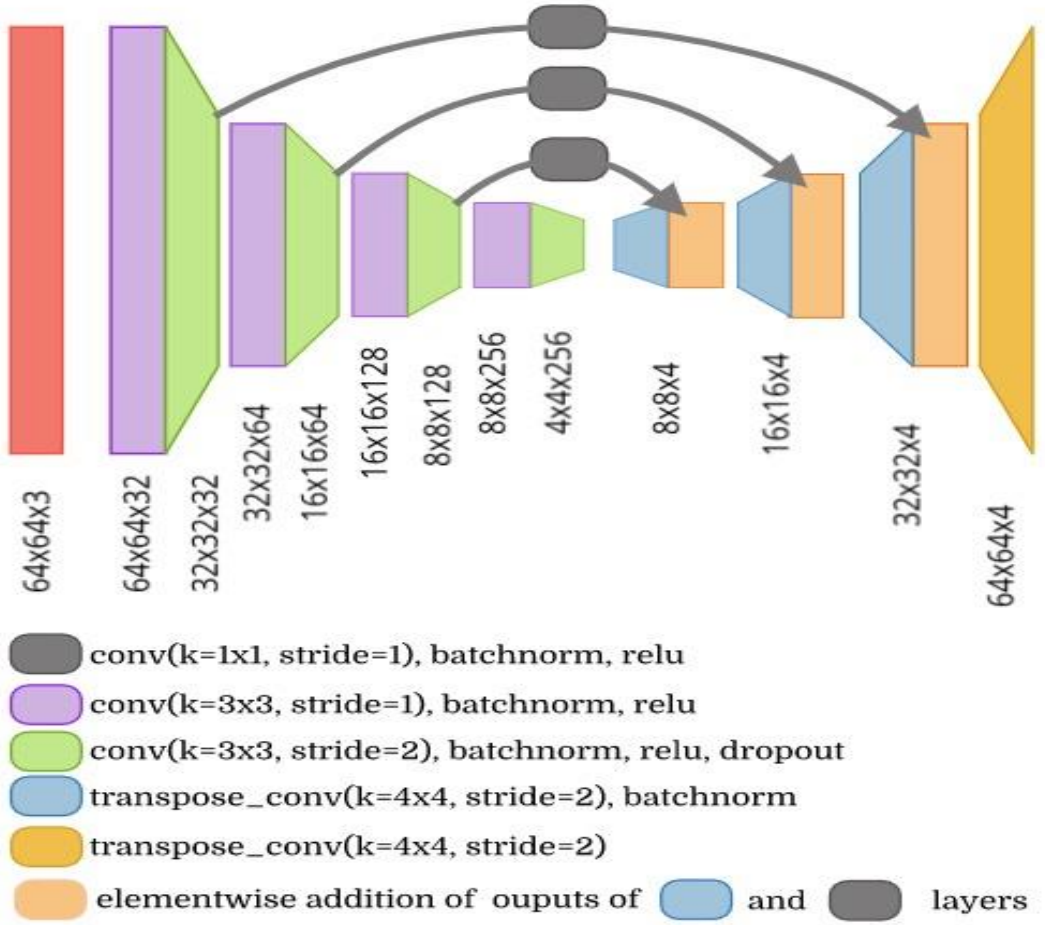
How Down sampling works?



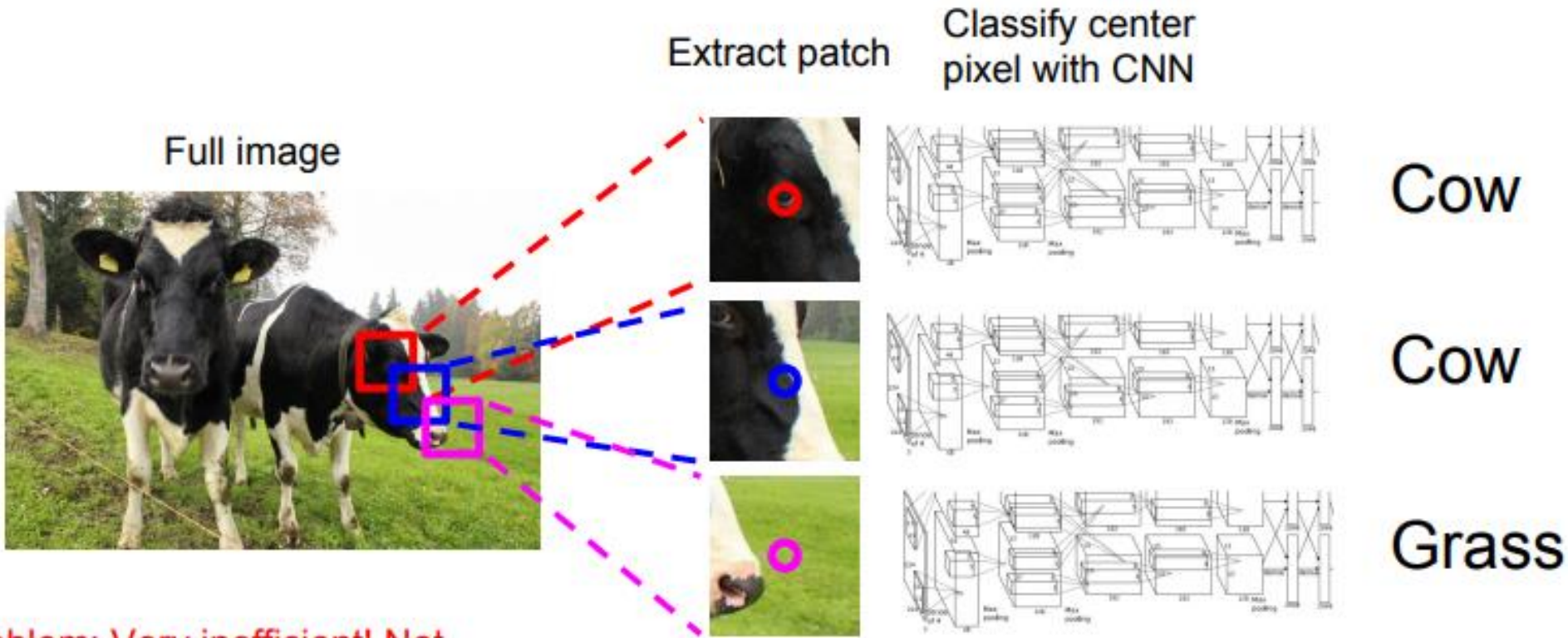


Working of semantic segmentation

- For the **downsampling** half of the architecture, it goes through pairs of convolutions. The first with a stride of 1, and the second with a stride of 2 to downsample by half.
- In the **upsampling** half, transpose convolutions with a kernel size of 4, and stride of 2 are used to upsample. The output of this goes through an elementwise addition with the output of the skip connections.



Semantic Segmentation Idea: Sliding Window



Problem: Very inefficient! Not reusing shared features between overlapping patches

Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

Pixel labels for training images must be known to train for semantic segmentation

void	road	sidewalk	building	wall
fence	pole	traffic light	traffic sign	vegetation
terrain	sky	person	rider	car
truck	bus	train	motorcycle	bicycle

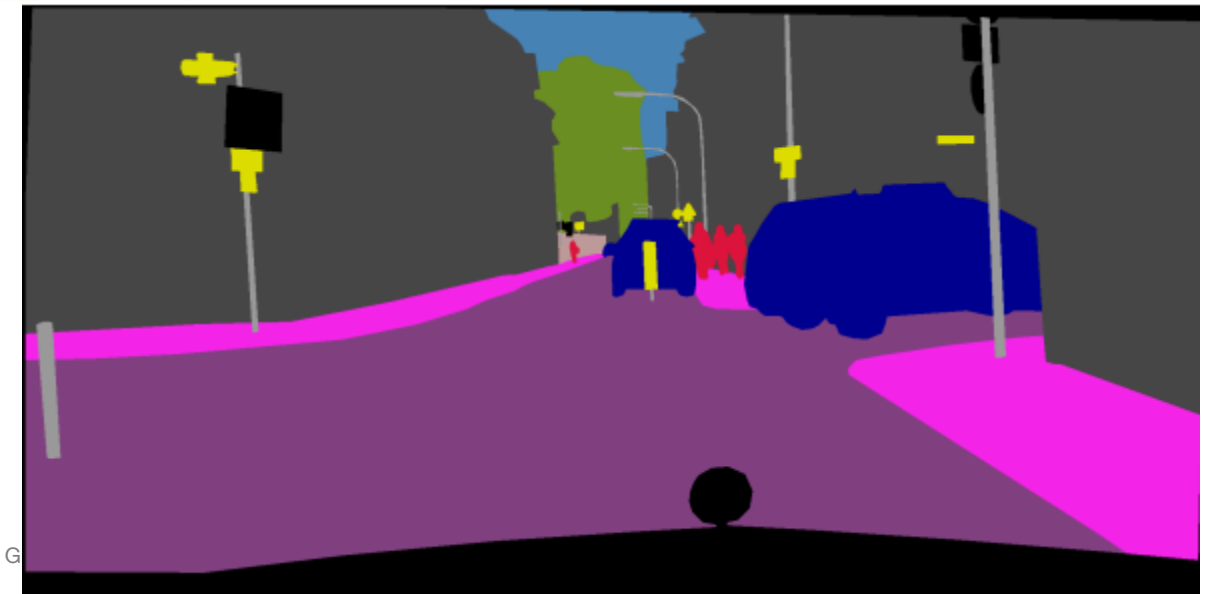
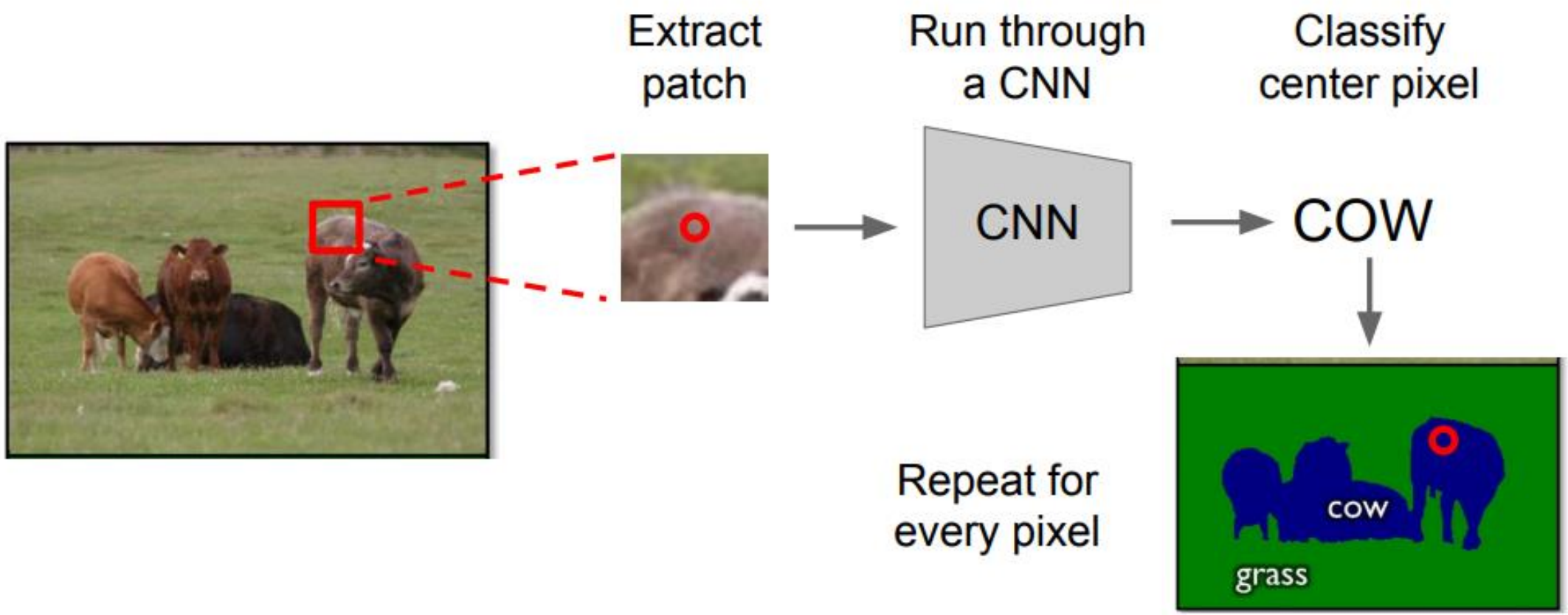


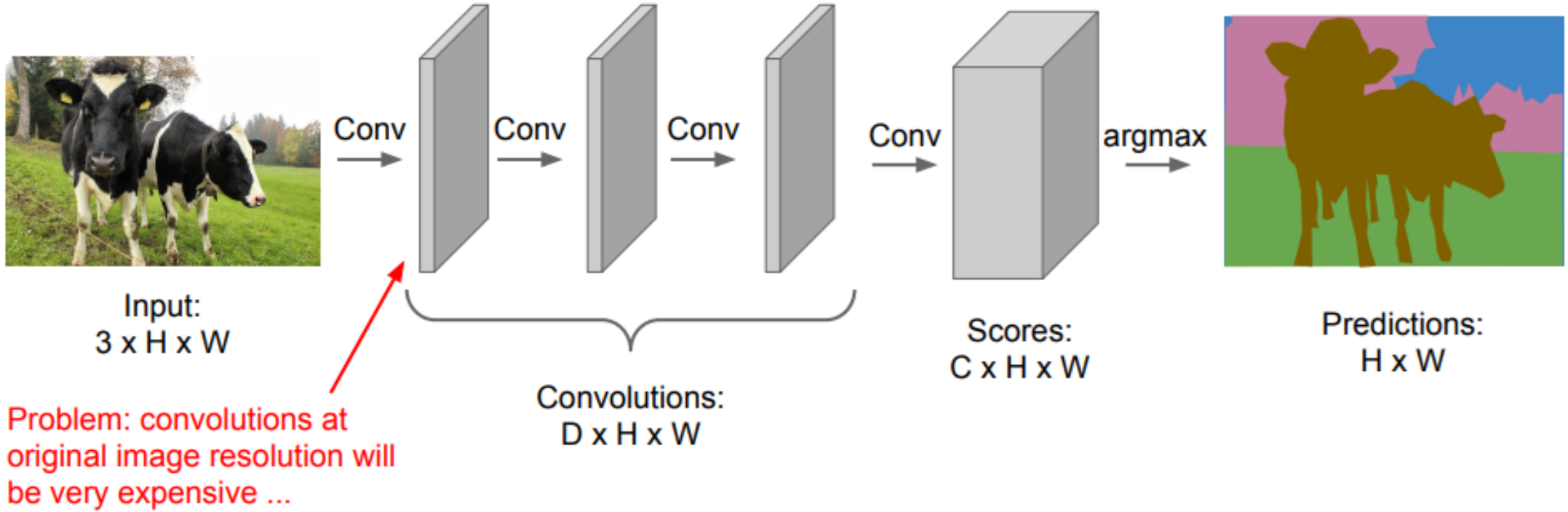
Image Source: "ICNet for Real-Time Semantic Segmentation on High-Resolution Images" Hengshuang Zhao1, Xiaojuan Qi, Xiaoyong Shen, Jianping Shi, Jiaya Jia, ECCV'18

Semantic Segmentation



Semantic Segmentation Idea: Fully Convolutional

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



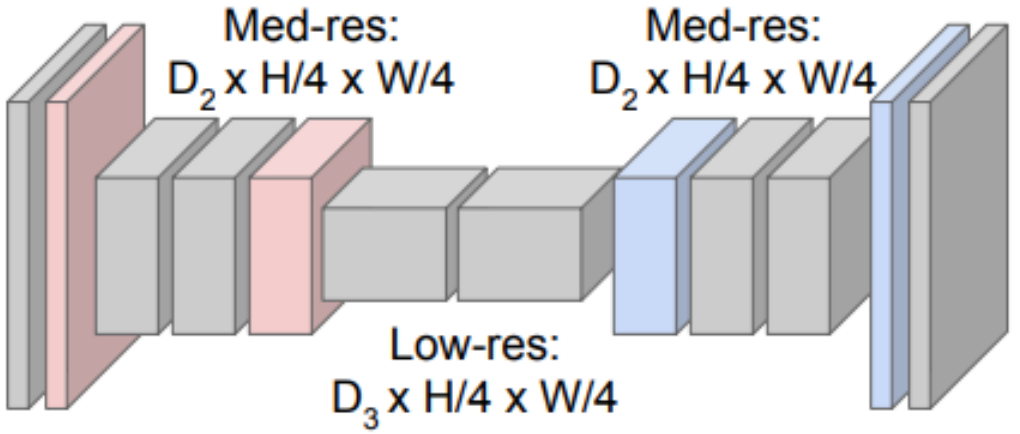
Semantic Segmentation Idea: Fully Convolutional

Downsampling:
Pooling, strided convolution

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

Upsampling:
Unpooling or strided transpose convolution

Input:
3 x H x W



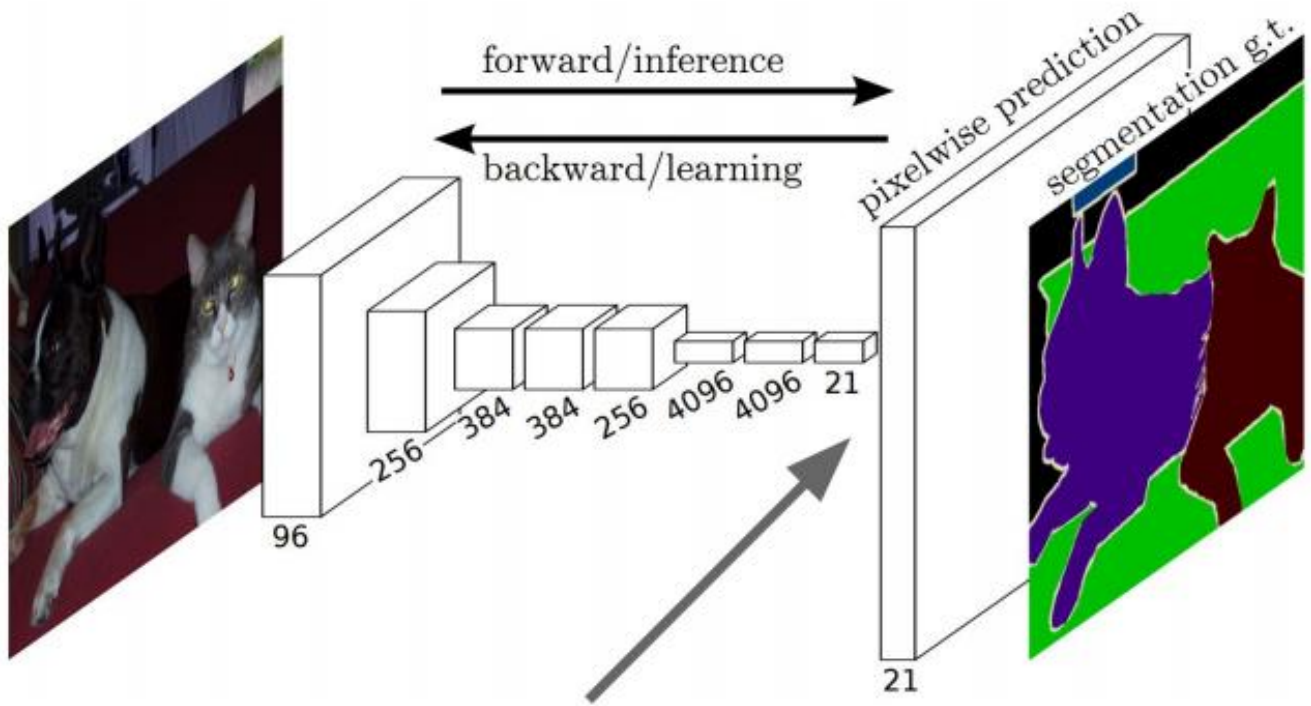
High-res:
D_1 x H/2 x W/2

High-res:
D_1 x H/2 x W/2

Predictions:
H x W

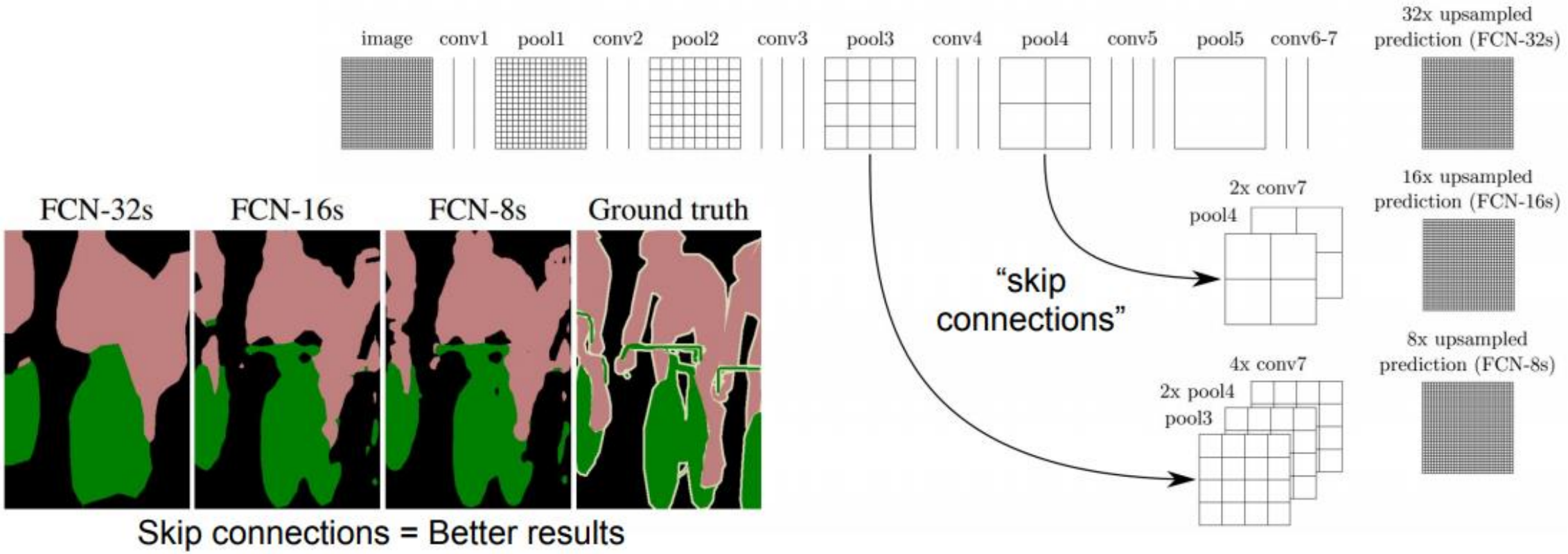
Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Semantic Segmentation: Upsampling



Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015

Semantic Segmentation: Upsampling



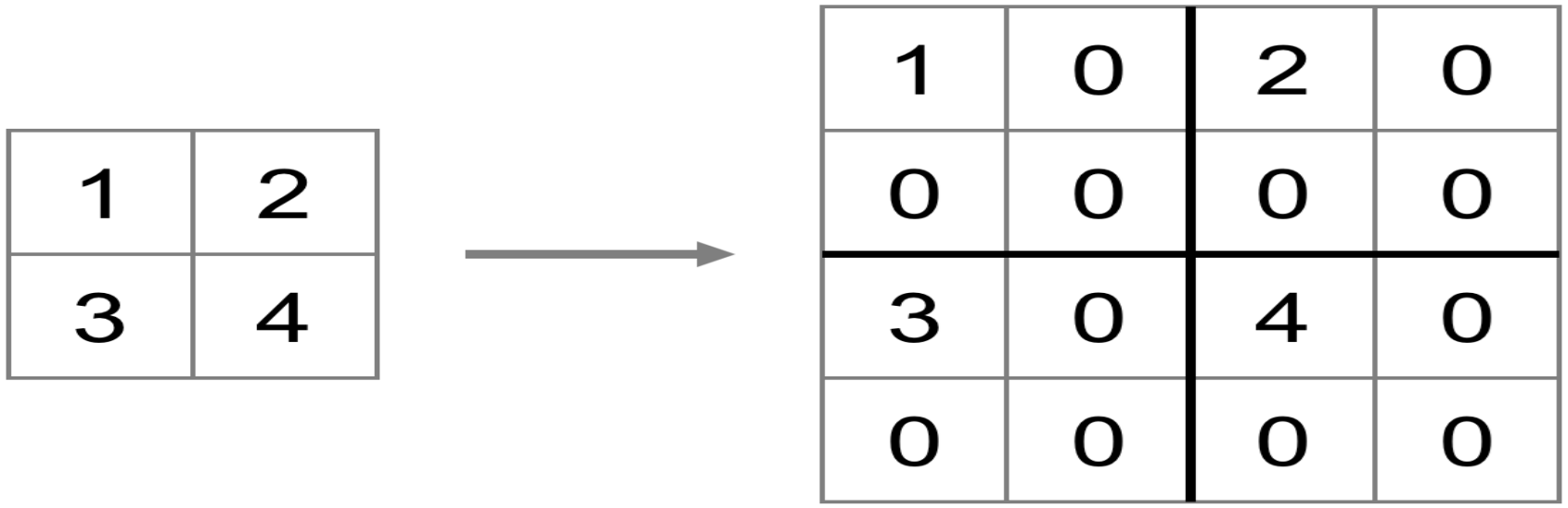
Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015

UpSampling

Transposed convolution / Deconvolution

Fractionally strided convolution

Max-unpooling: Preserves spatial information



In-Network upsampling: “Max Unpooling”

Max Pooling
Remember which element was max!

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

Input: 4 x 4

5	6
7	8

Output: 2 x 2

→ ... → Rest of the network

Max Unpooling
Use positions from pooling layer

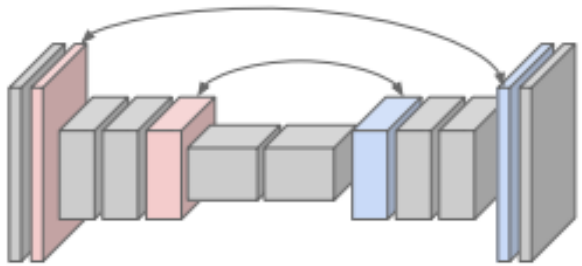
1	2
3	4

Input: 2 x 2

0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

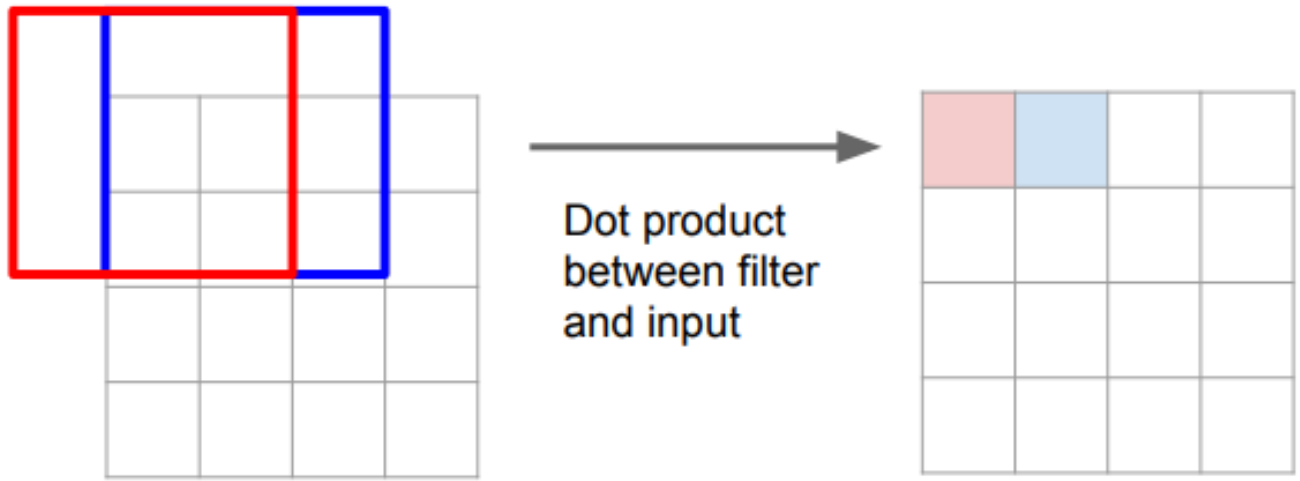
Output: 4 x 4

Corresponding pairs of downsampling and upsampling layers



Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 1 pad 1

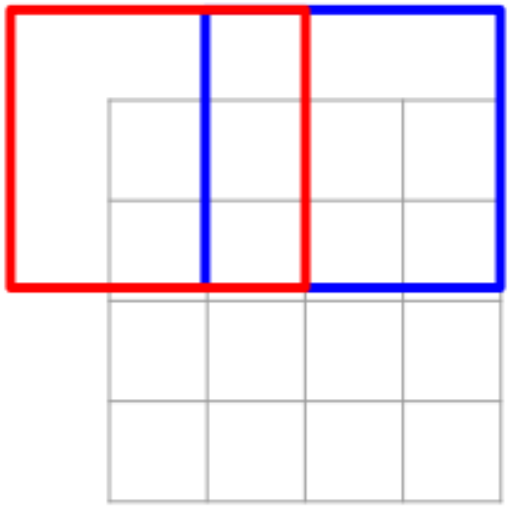


Input: 4 x 4

Output: 4 x 4

Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 2 pad 1



Input: 4 x 4

Dot product between filter and input

Output: 2 x 2

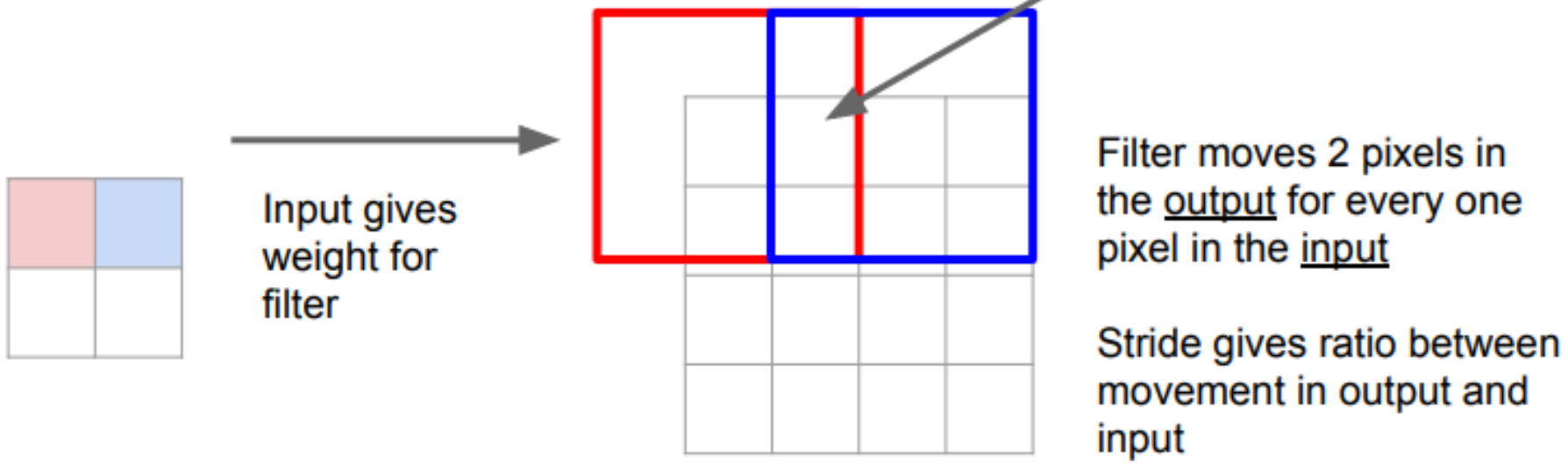
Filter moves 2 pixels in the input for every one pixel in the output

Stride gives ratio between movement in input and output

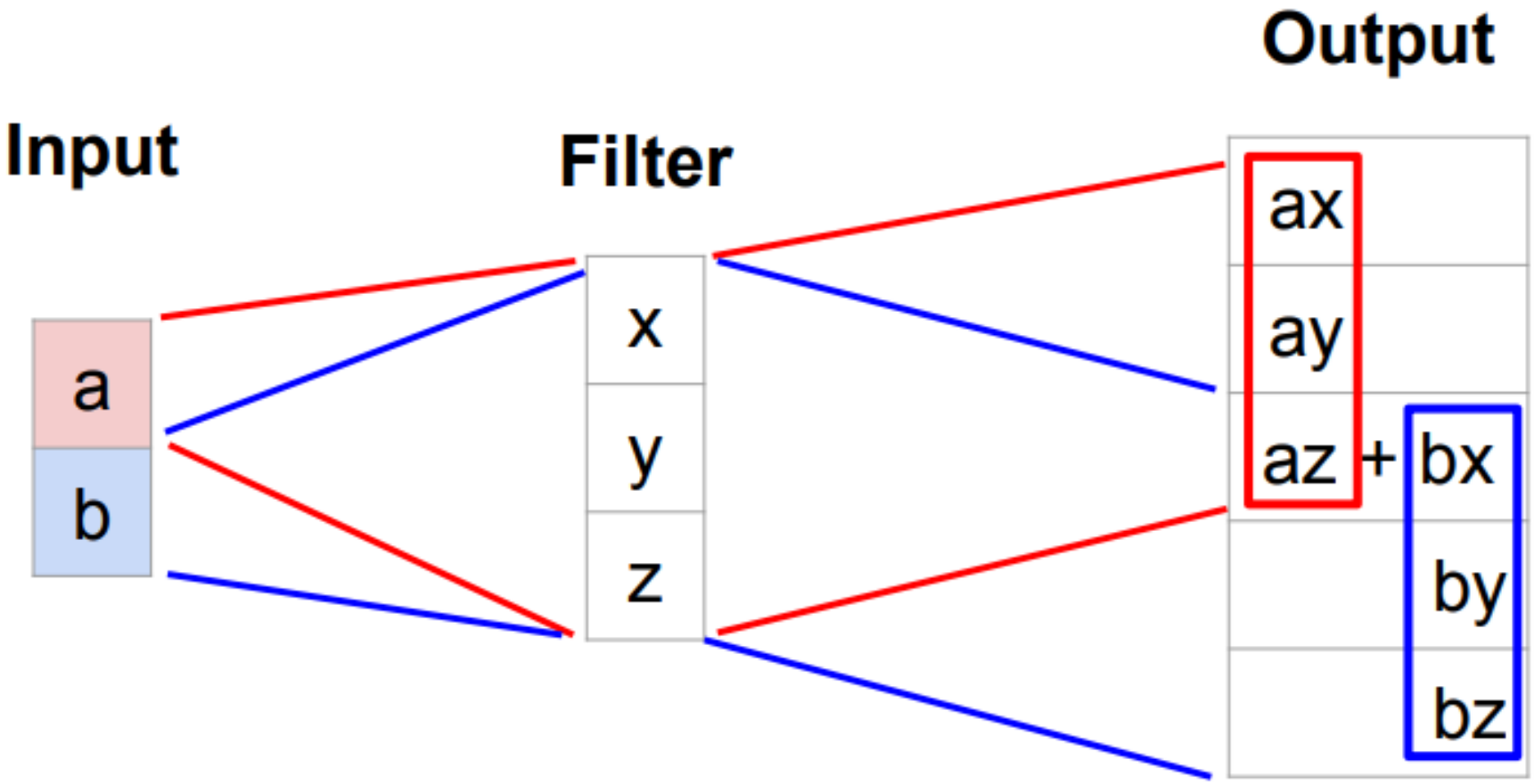
Learnable Upsampling: Transpose Convolution

Other names:
-Deconvolution (bad)
-Upconvolution
-Fractionally strided convolution
-Backward strided convolution

3 x 3 **transpose** convolution, stride 2 pad 1



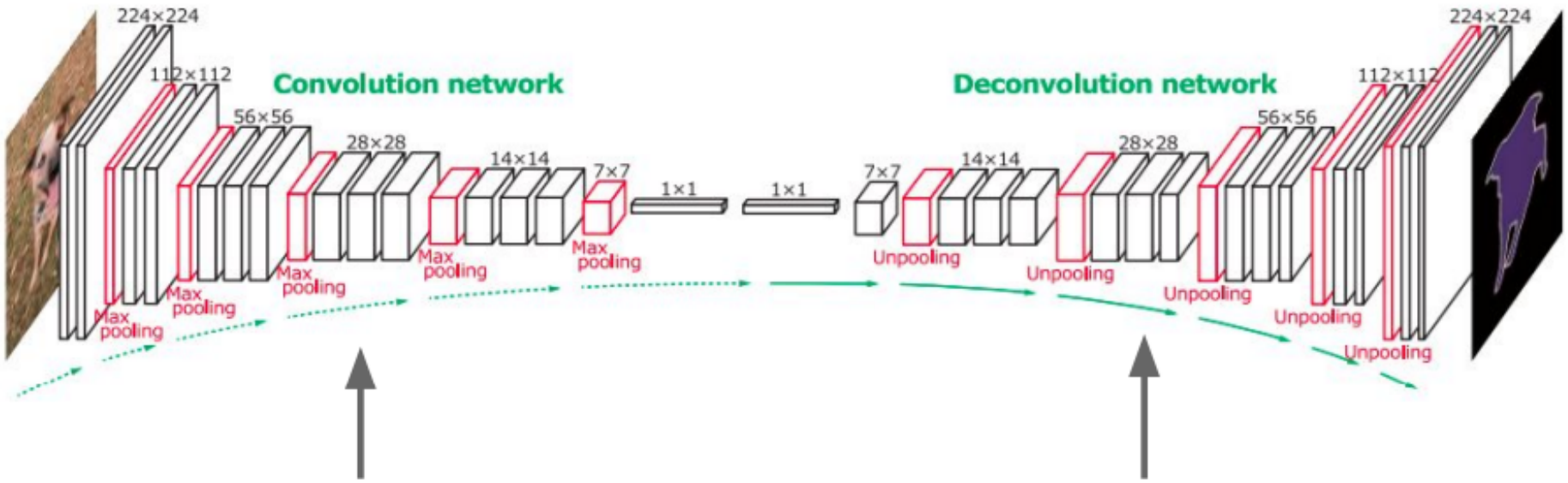
Learnable Upsampling: 1D Example



Output contains copies of the filter weighted by the input, summing at where at overlaps in the output

Need to crop one pixel from output to make output exactly 2x input

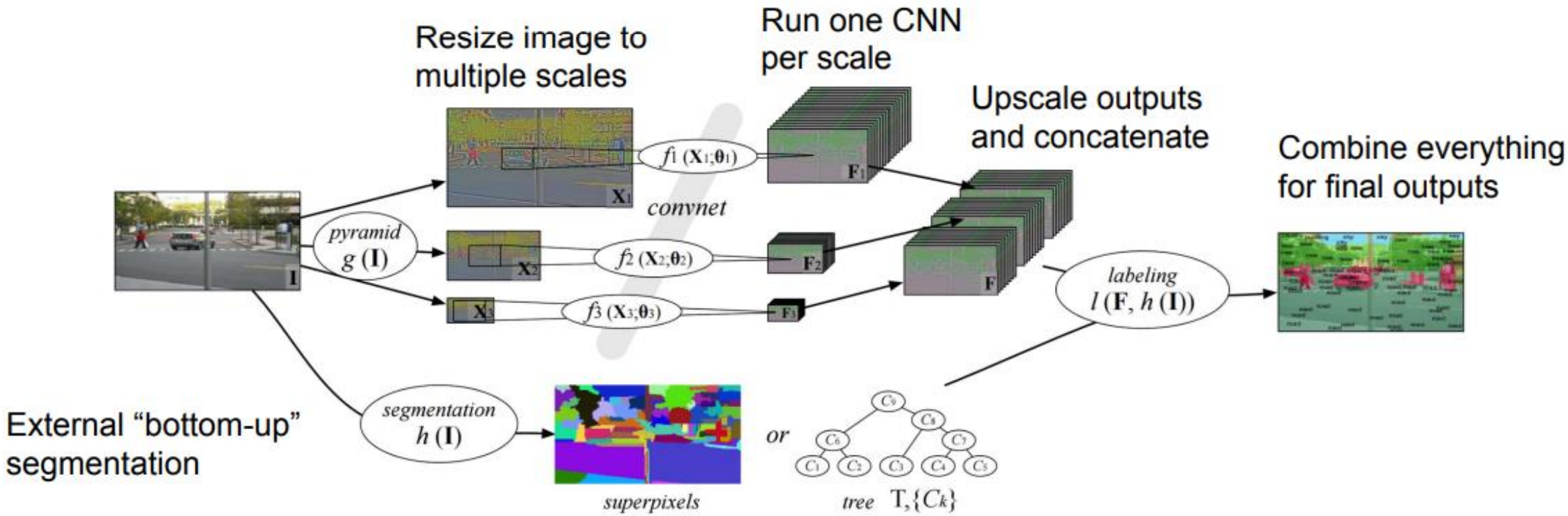
Semantic Segmentation: Upsampling



Noh et al, “Learning Deconvolution Network for Semantic Segmentation”, ICCV 2015

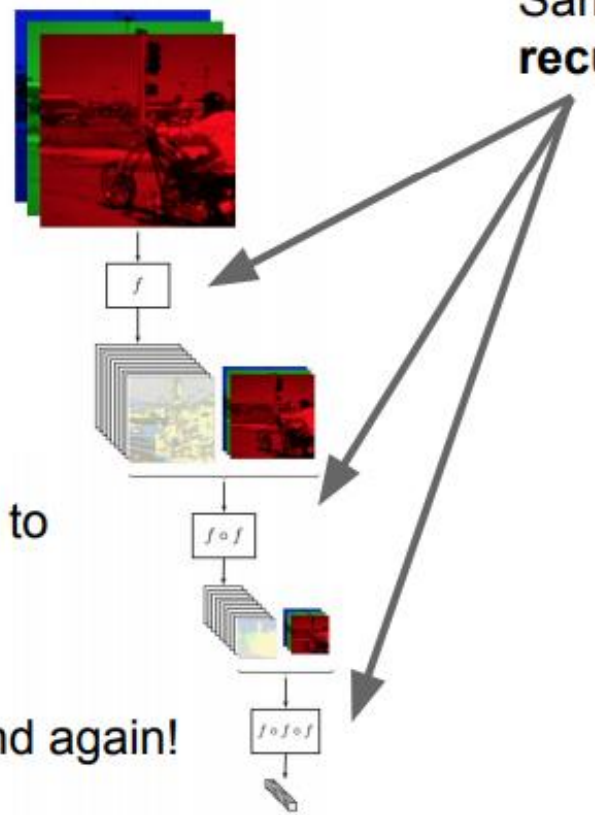
6 days of training on Titan X…

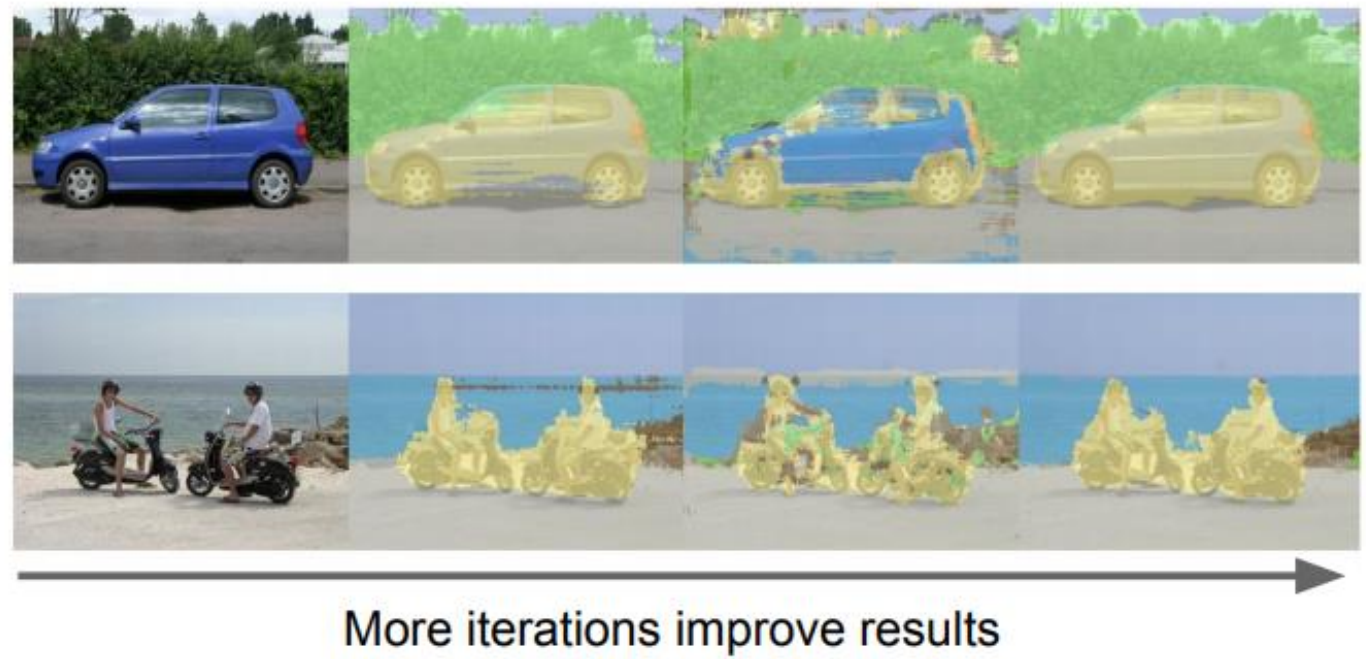
Semantic Segmentation: Multi-Scale



Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013

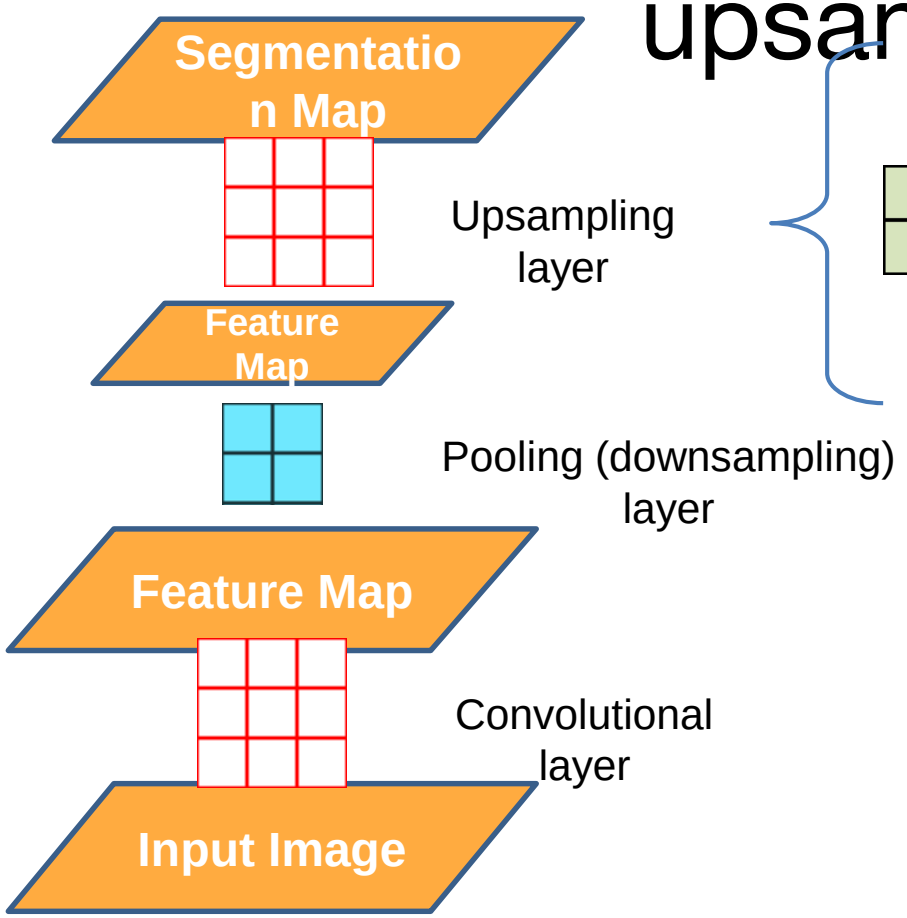
Semantic Segmentation: Refinement

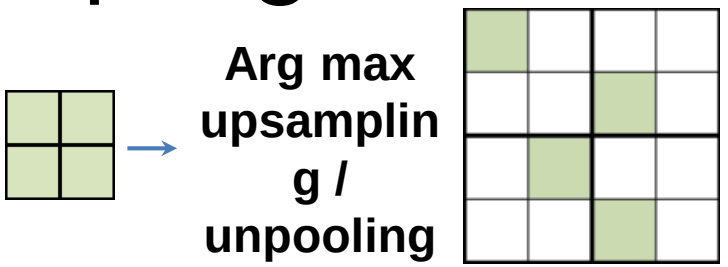




Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

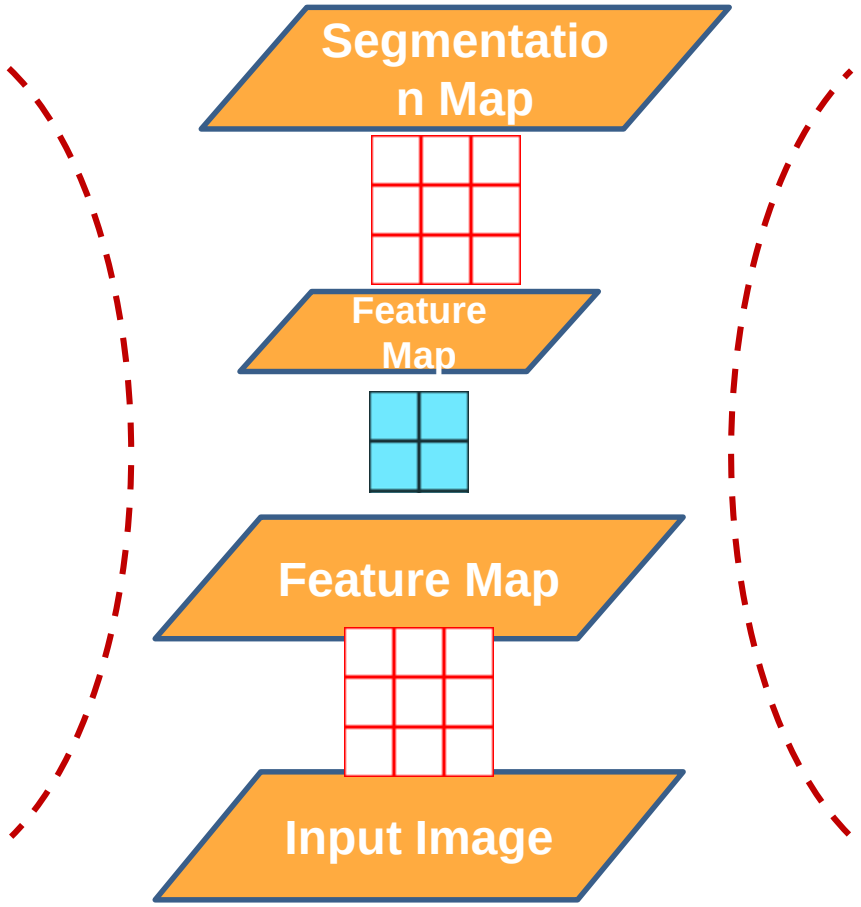
To produce a segmentation map downsampling is followed by upsampling



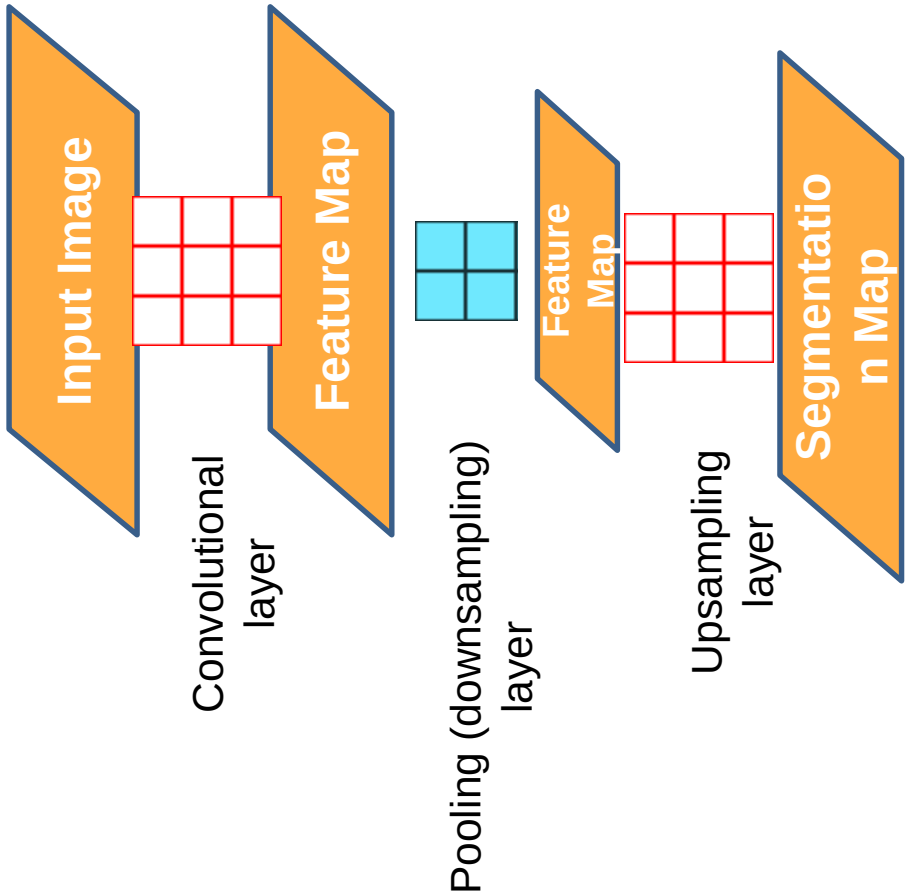


- Downsampling collects feature evidence from a larger area
- Upsampling distributes the information of a segment back to the original pixel domain

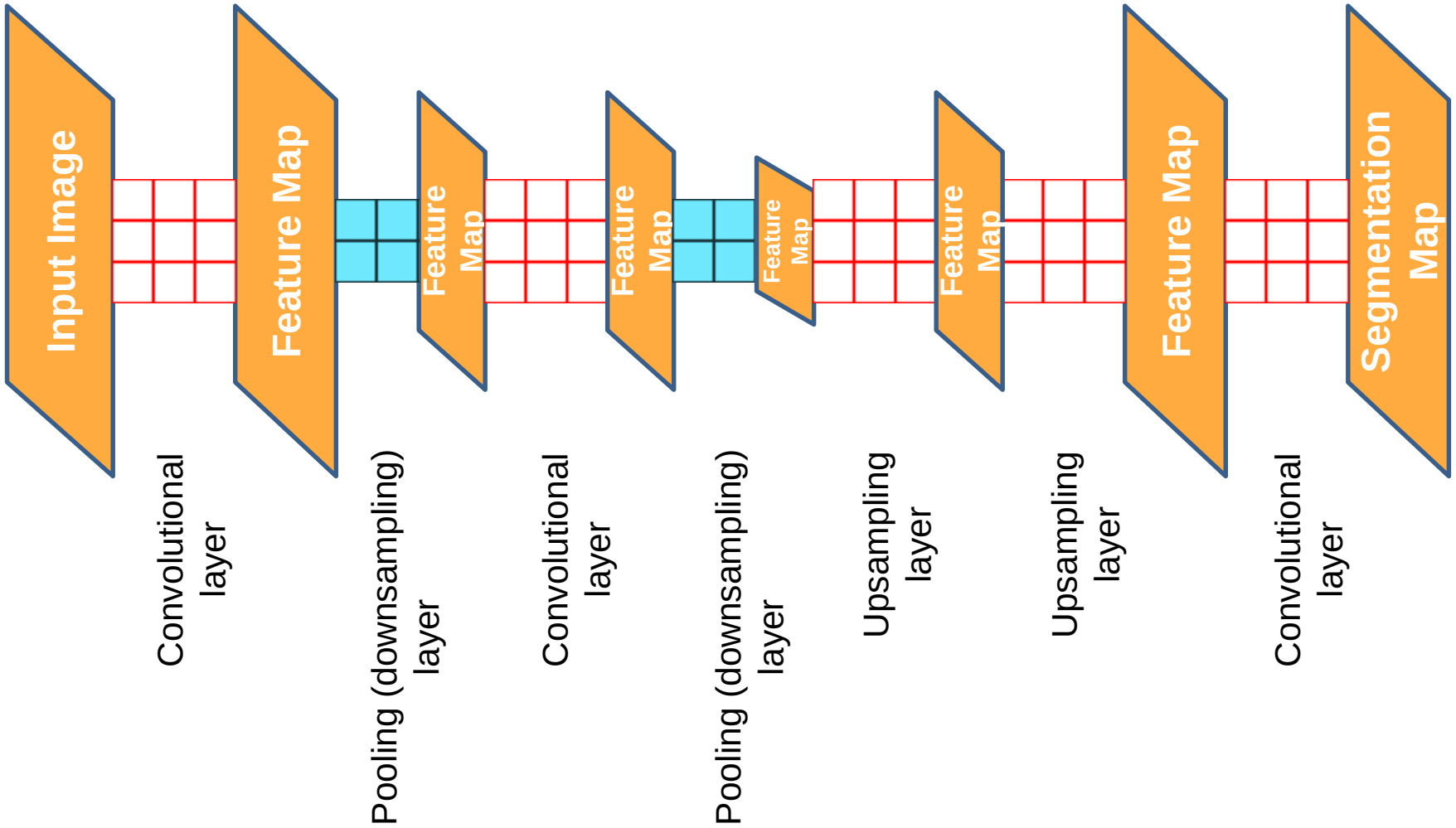
Downsampling and upsampling leads to an hour-glass structure



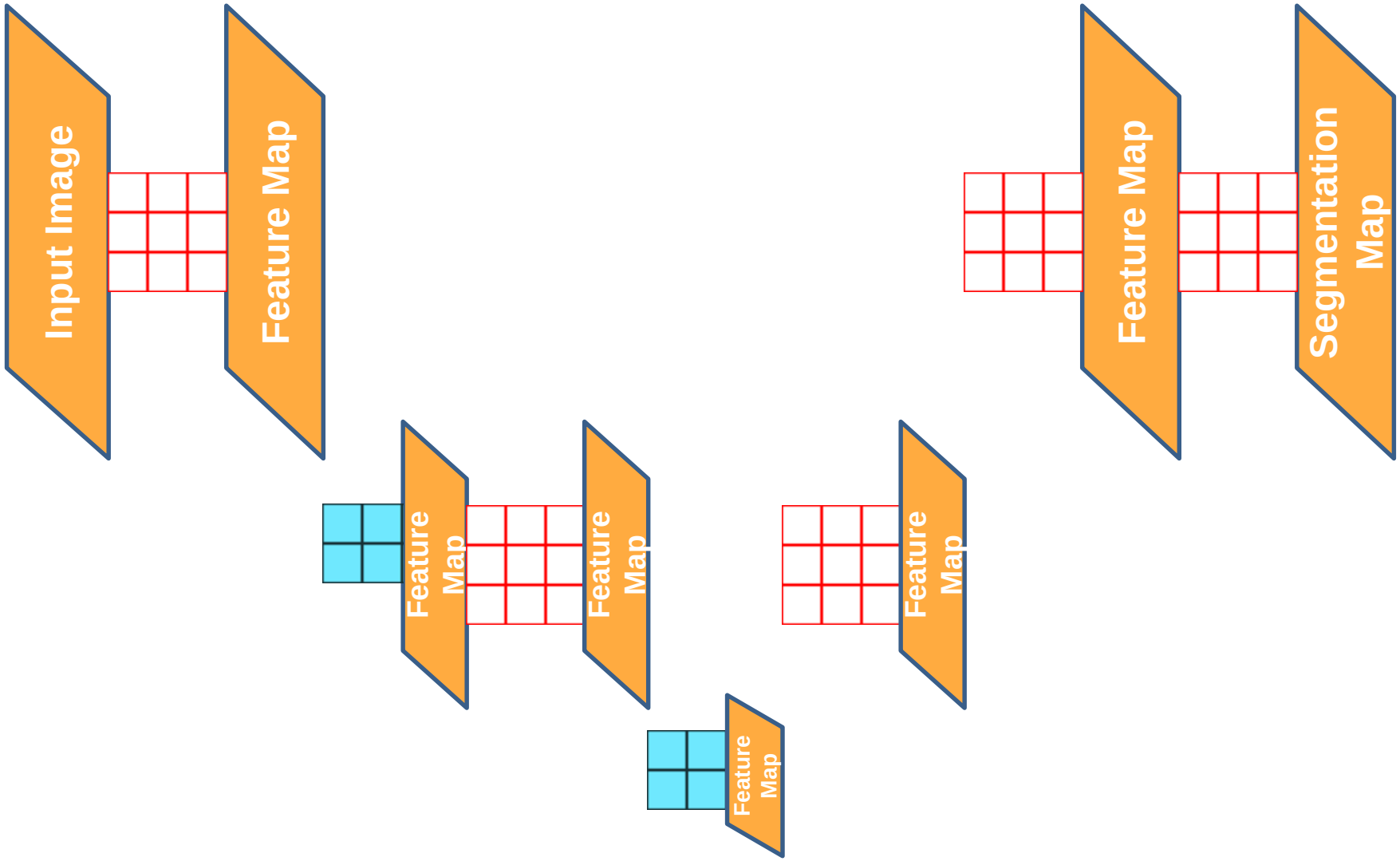
Let us rearrange the layers horizontally



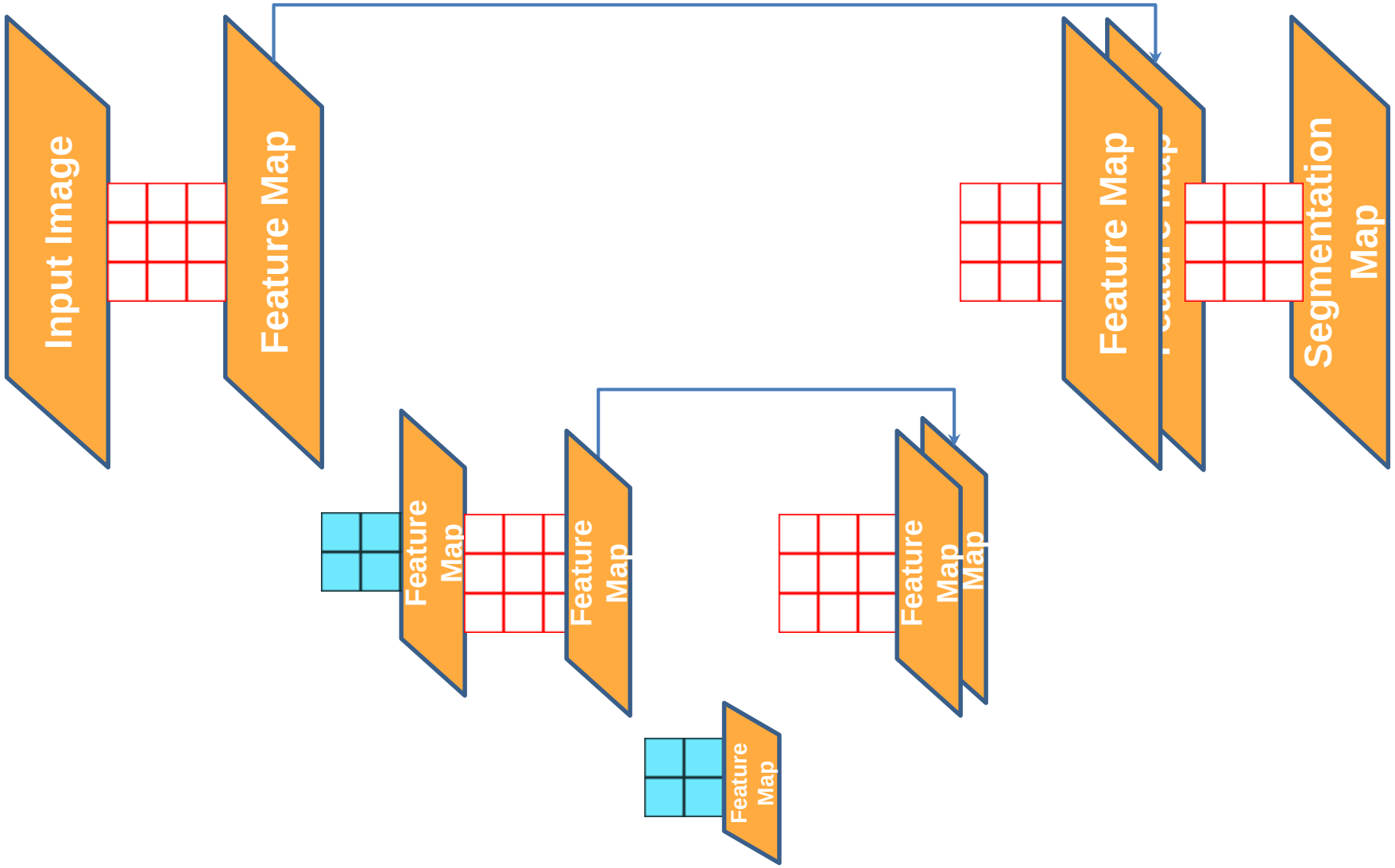
More layers can be added



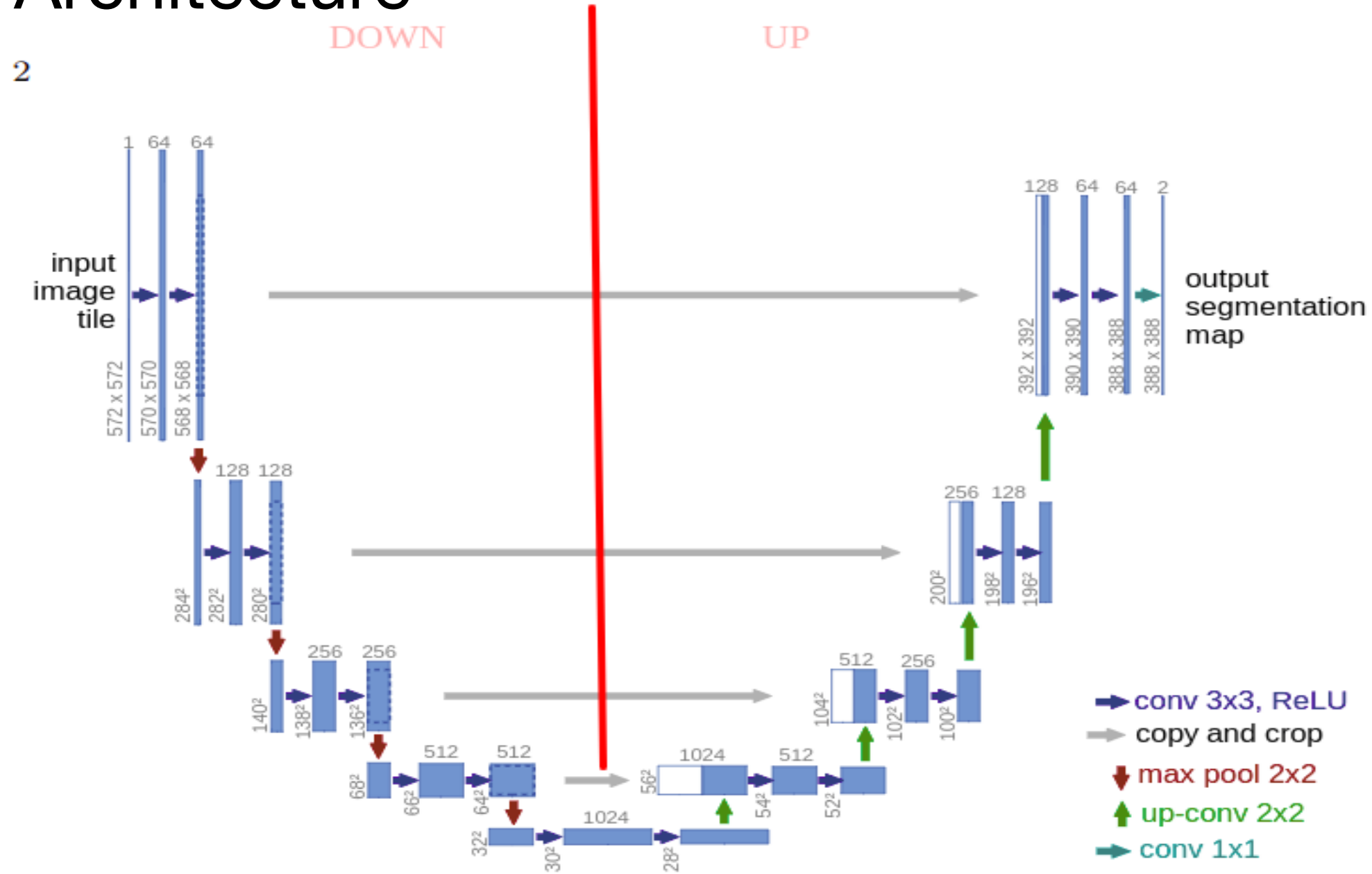
Visually rearrange layers in a big U



Concatenate previous feature maps for finer spatial context



U-Net Architecture



Source: "U-Net: Convolutional Networks for Biomedical Image Segmentation" Olaf Ronneberger, Philipp Fischer, Thomas Brox, 2015

A sample output for nucleus segmentation in pathology

A general representation of fully convolutional networks. The encoder is composed of convolutional and pooling layers for downsampling and the decoder is composed of deconvolutional layers for upsampling.

Loss function: Dice coefficient

The evaluation metric used for this competition is Dice coefficient. Dice coefficient is defined as follows:

$$\frac{2 * |X \cap Y|}{|X| + |Y|}$$

where X is the predicted set of pixels and Y is the ground truth. A higher dice coefficient is better.

A dice coefficient of 1 can be achieved when there is perfect overlap between X and Y. Since the denominator is constant, the only way to maximize this metric is to increase overlap between X and Y.

For more info on Dice coefficient: https://www.kaggle.com/c/carvana-image-masking-challenge#evaluation

Let’s move on to the python notebooks !!

MobileNet

ρ is introduced to **control the input image resolution** of the network

where ρ is between 0 to 1. And the input resolution is 224, 192, 160, and 128. When ρ=1, it is the baseline MobileNet

α is introduced to **control the input width of a layer**

where α is between 0 to 1, with typical settings of 1, 0.75, 0.5 and 0.25. When α=1

References

- http://ronny.rest/tutorials/module/seg_01
- https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf
- http://www.cs.toronto.edu/~tingwuwang/semantic_segmentation.pdf
- https://medium.com/datadriveninvestor/deep-learning-in-medical-imaging-3c1008431aaf
- https://www.mladdict.com/neural-network-simulator