

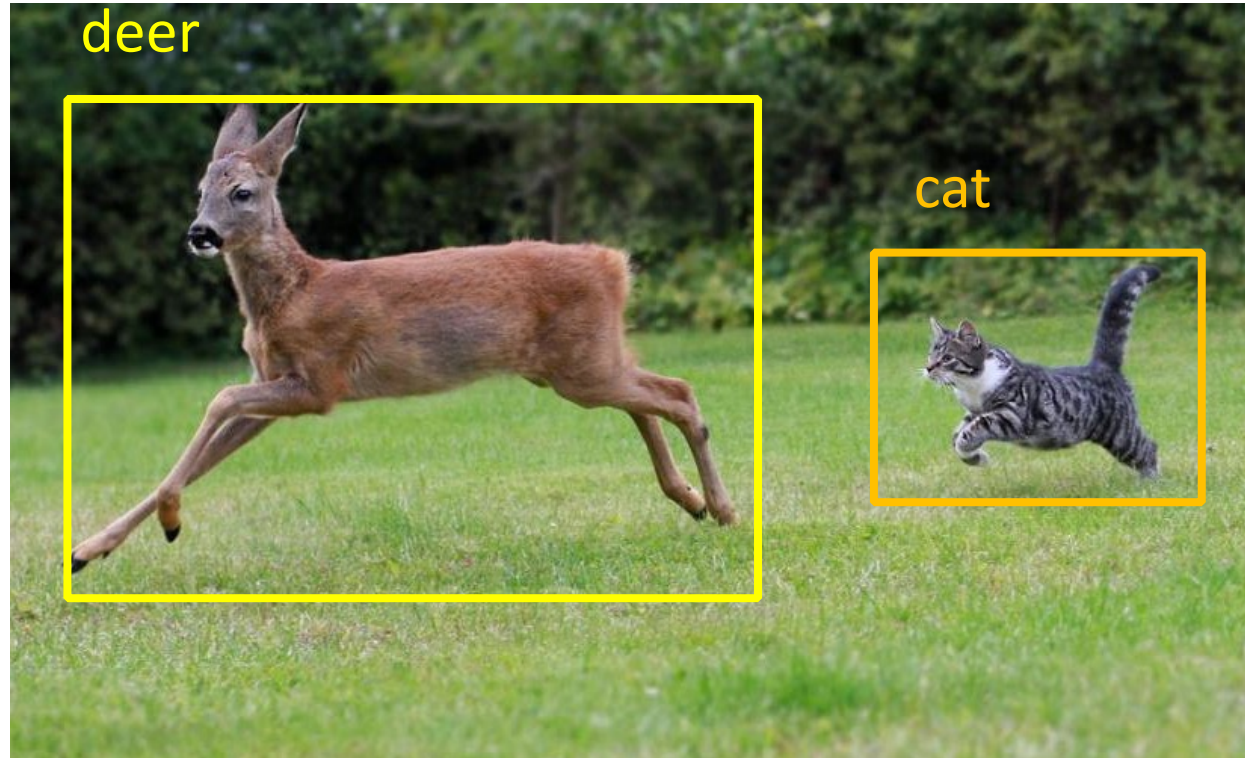


# Deep Learning for Vision & Language

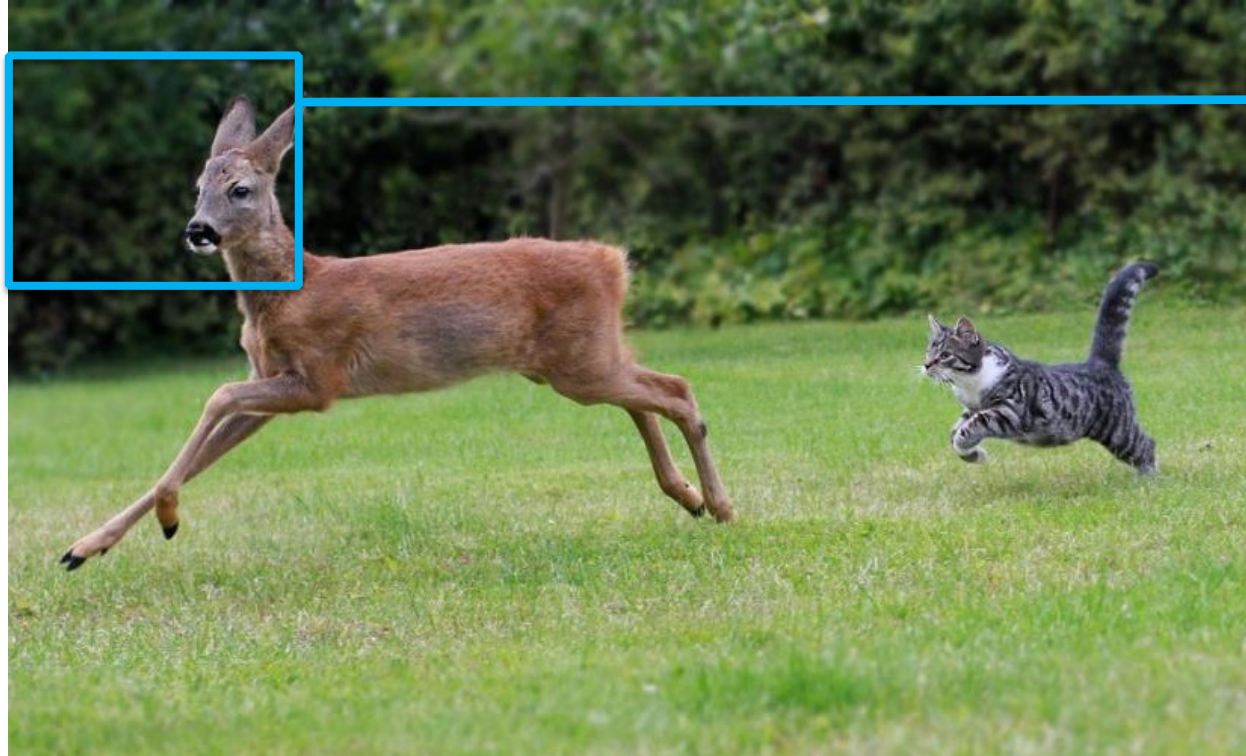
Convolutional Neural Networks for Detection and Segmentation



# Object Detection



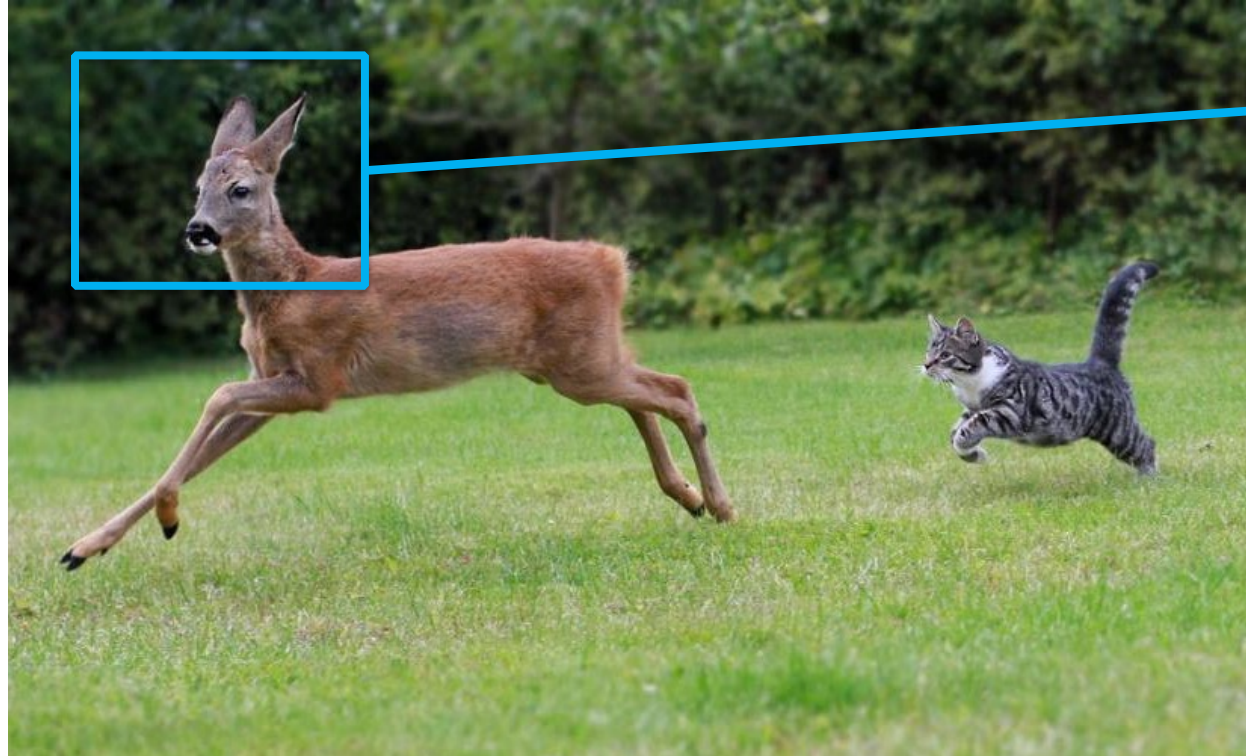
# Object Detection as Classification



deer?  
cat?  
background?

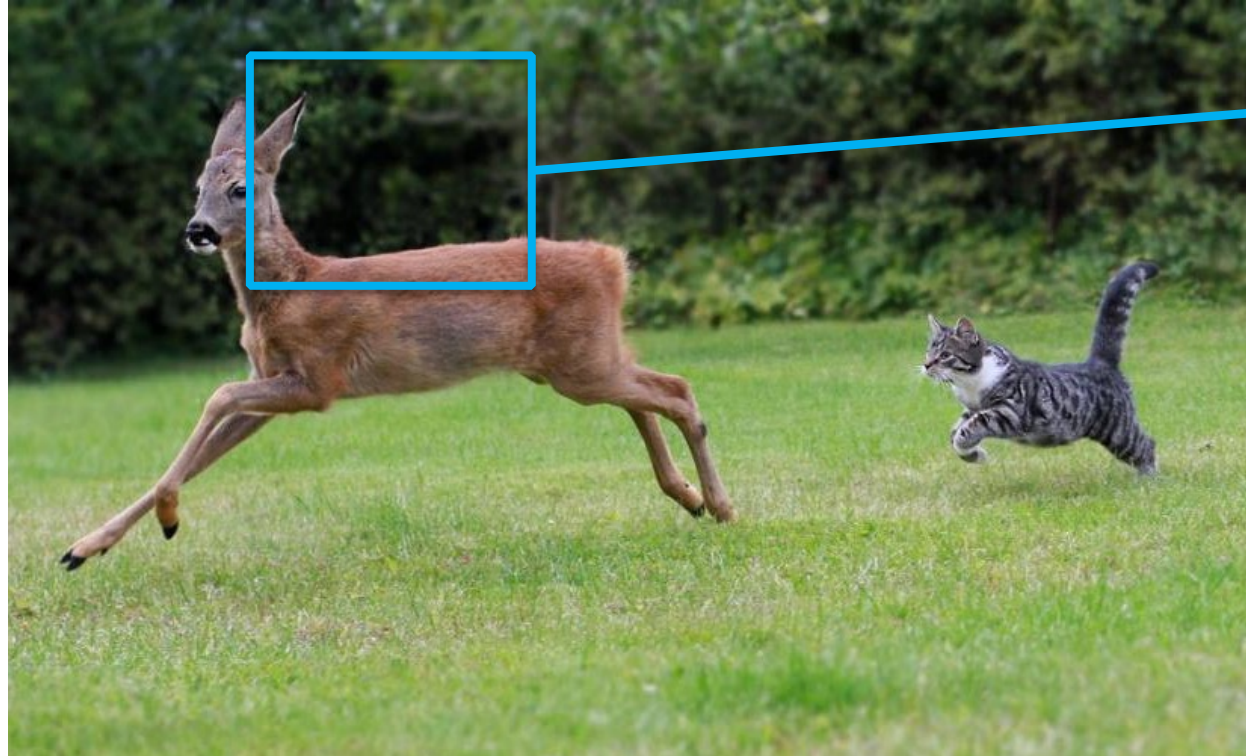


# Object Detection as Classification



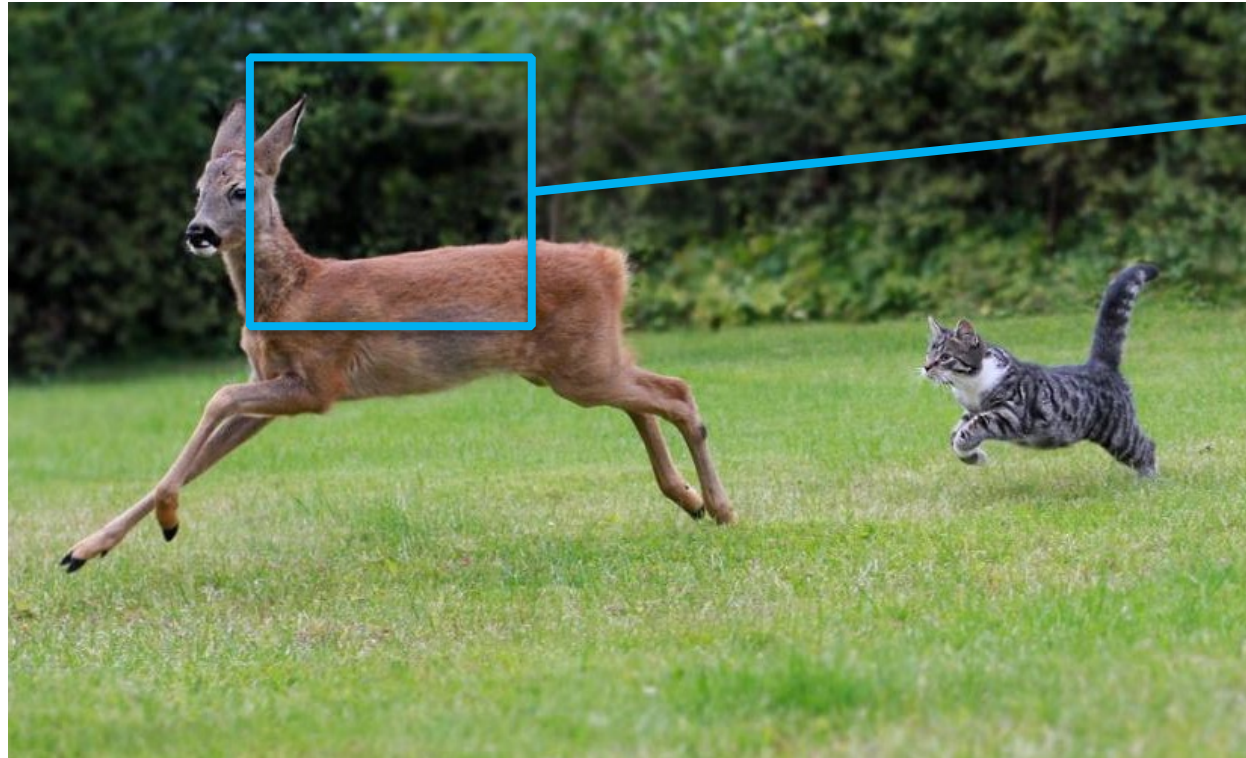
deer?  
cat?  
background?

# Object Detection as Classification



deer?  
cat?  
background?

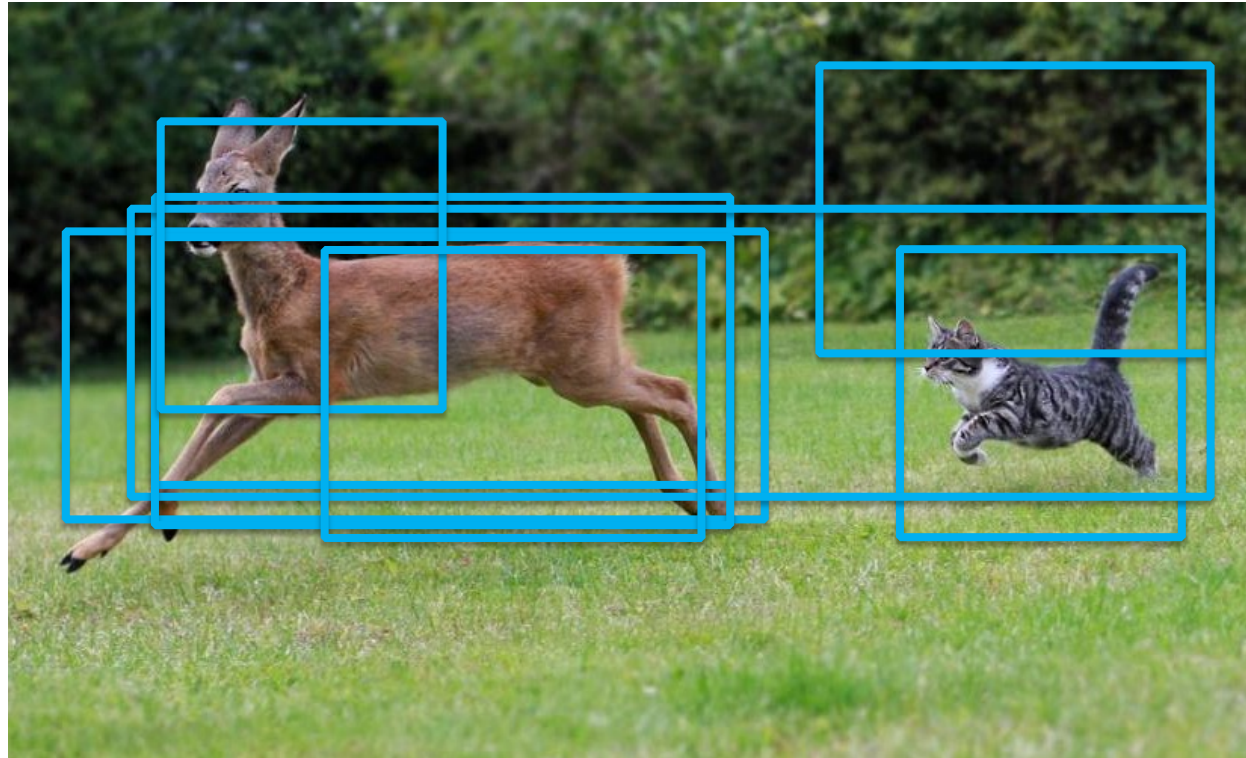
# Object Detection as Classification with Sliding Window



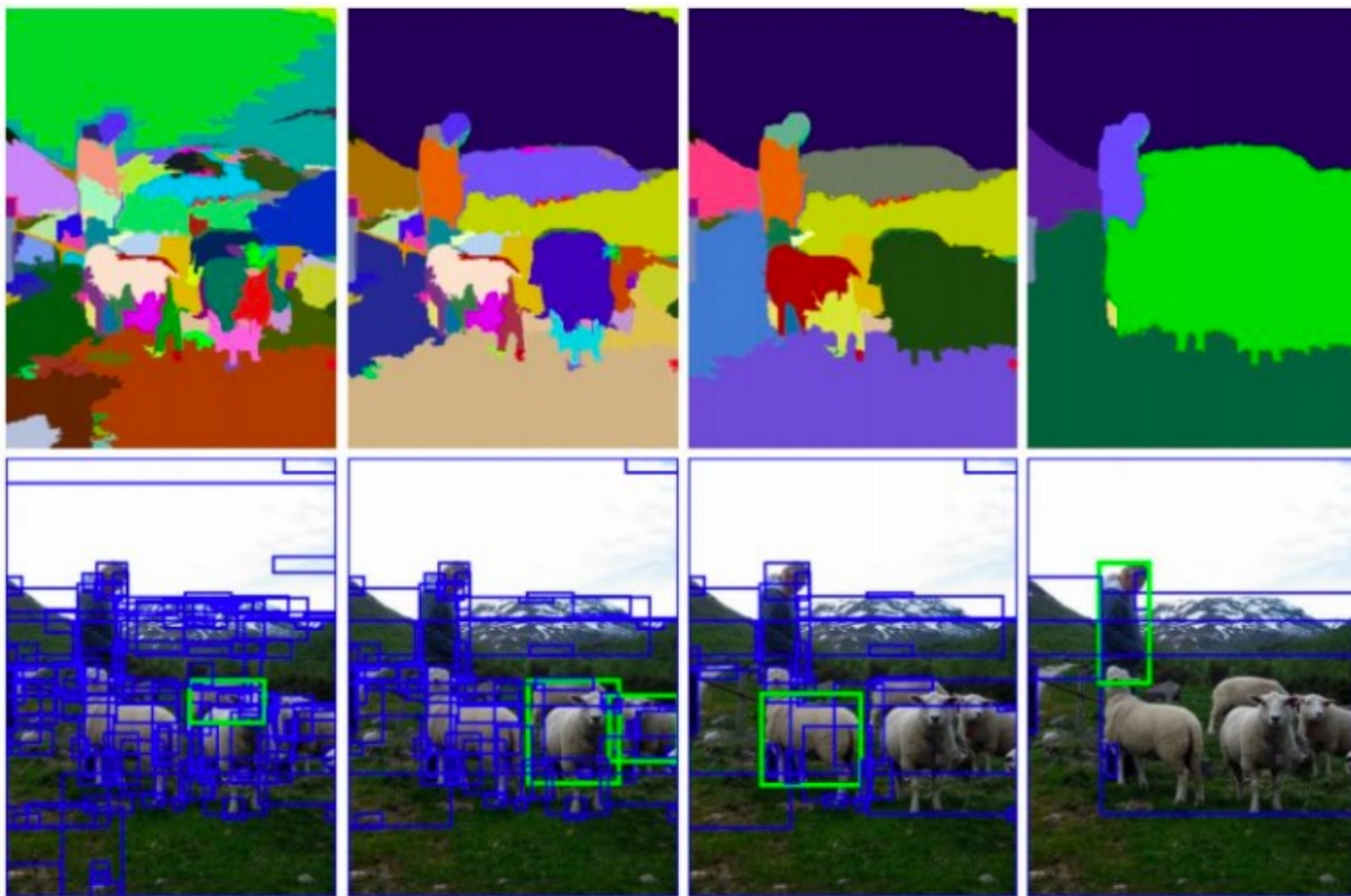
deer?  
cat?  
background?



# Object Detection as Classification with Box Proposals



# Box Proposal Method – SS: Selective Search

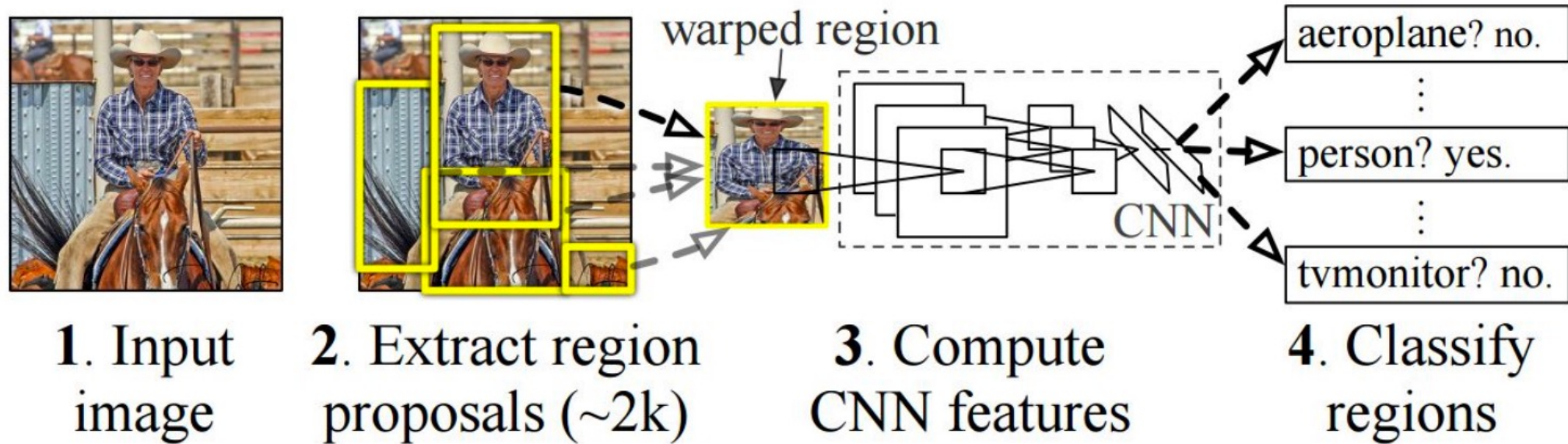


Segmentation As  
Selective Search for  
Object Recognition. van  
de Sande et al. ICCV  
2011



# RCNN

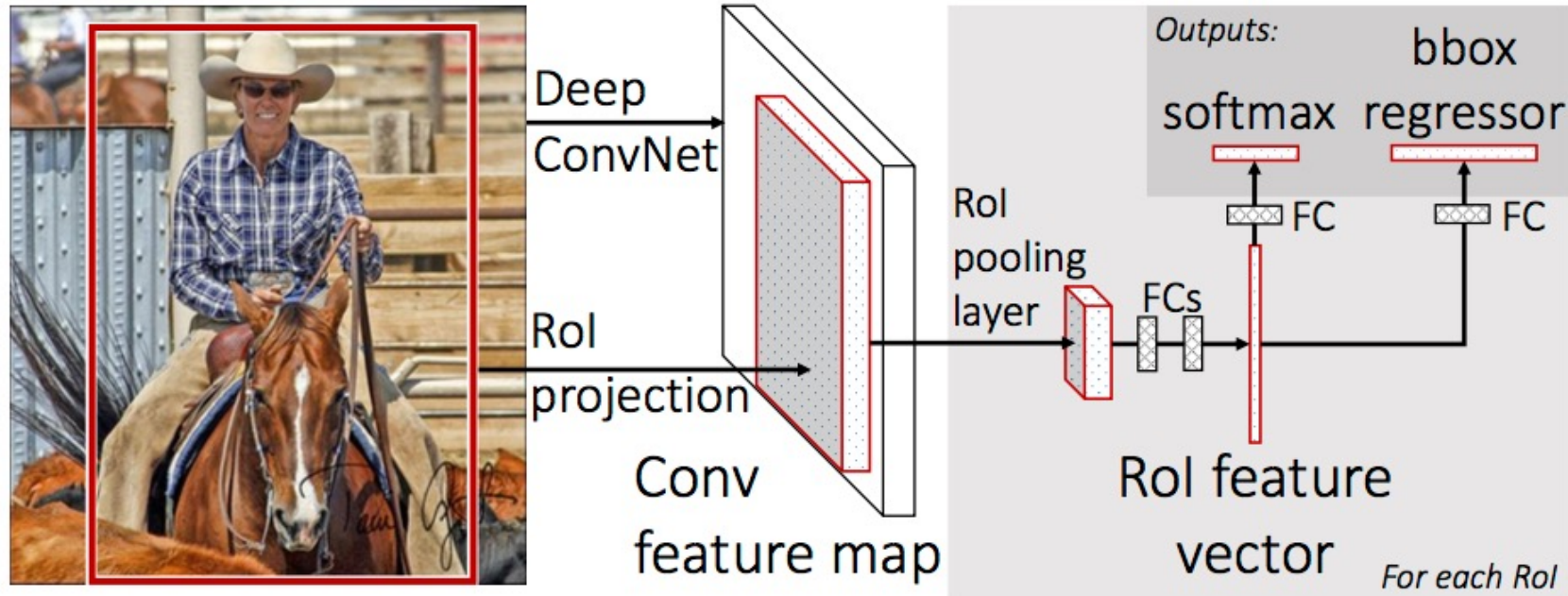
## R-CNN: *Regions with CNN features*



<https://people.eecs.berkeley.edu/~rbg/papers/r-cnn-cvpr.pdf>

Rich feature hierarchies for accurate object detection and semantic segmentation. Girshick et al. CVPR 2014.

# Fast-RCNN



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

<https://arxiv.org/abs/1504.08083>

Fast R-CNN. Girshick. ICCV 2015.

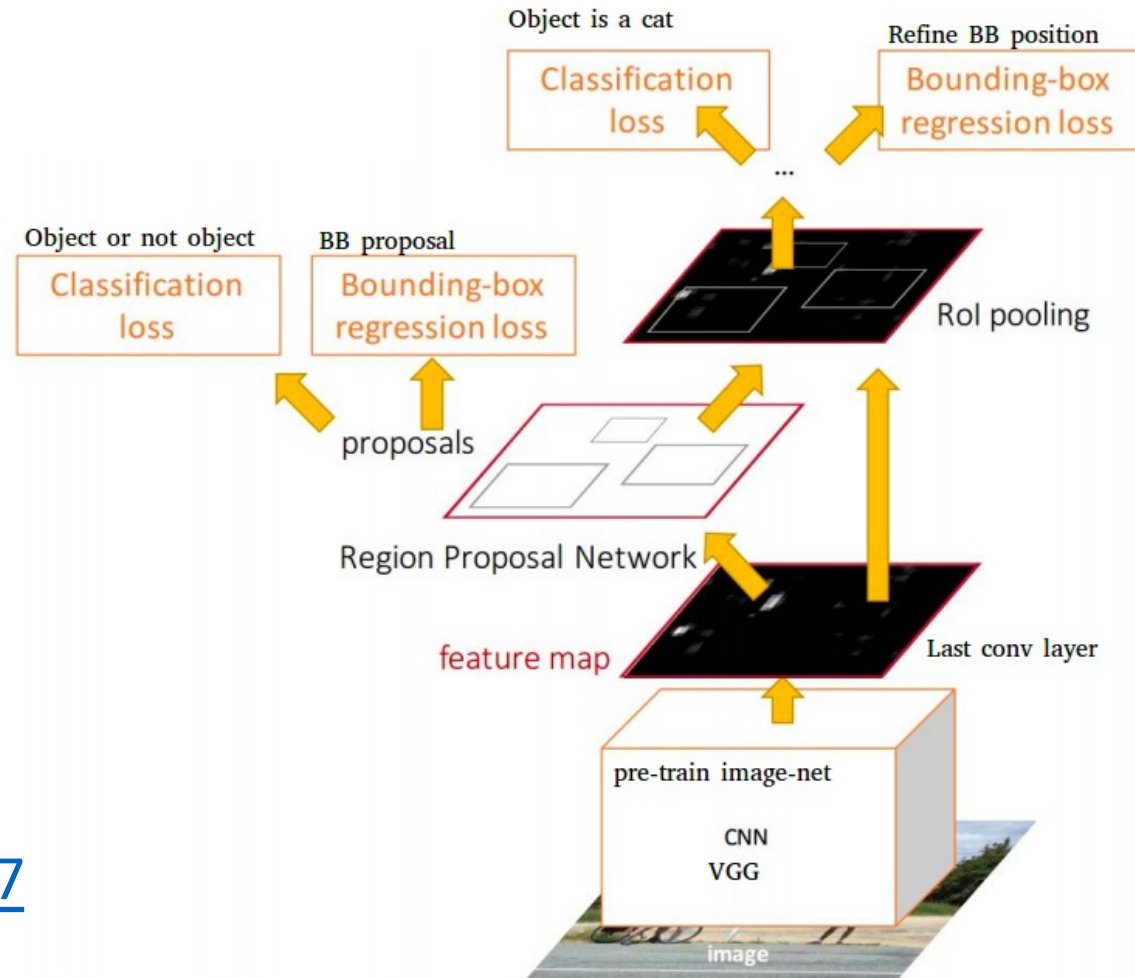
<https://github.com/sunshineatnoon/Paper-Collection/blob/master/Fast-RCNN.md>

# Faster-RCNN

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

<https://arxiv.org/abs/1506.01497>

Ren et al. NIPS 2015.



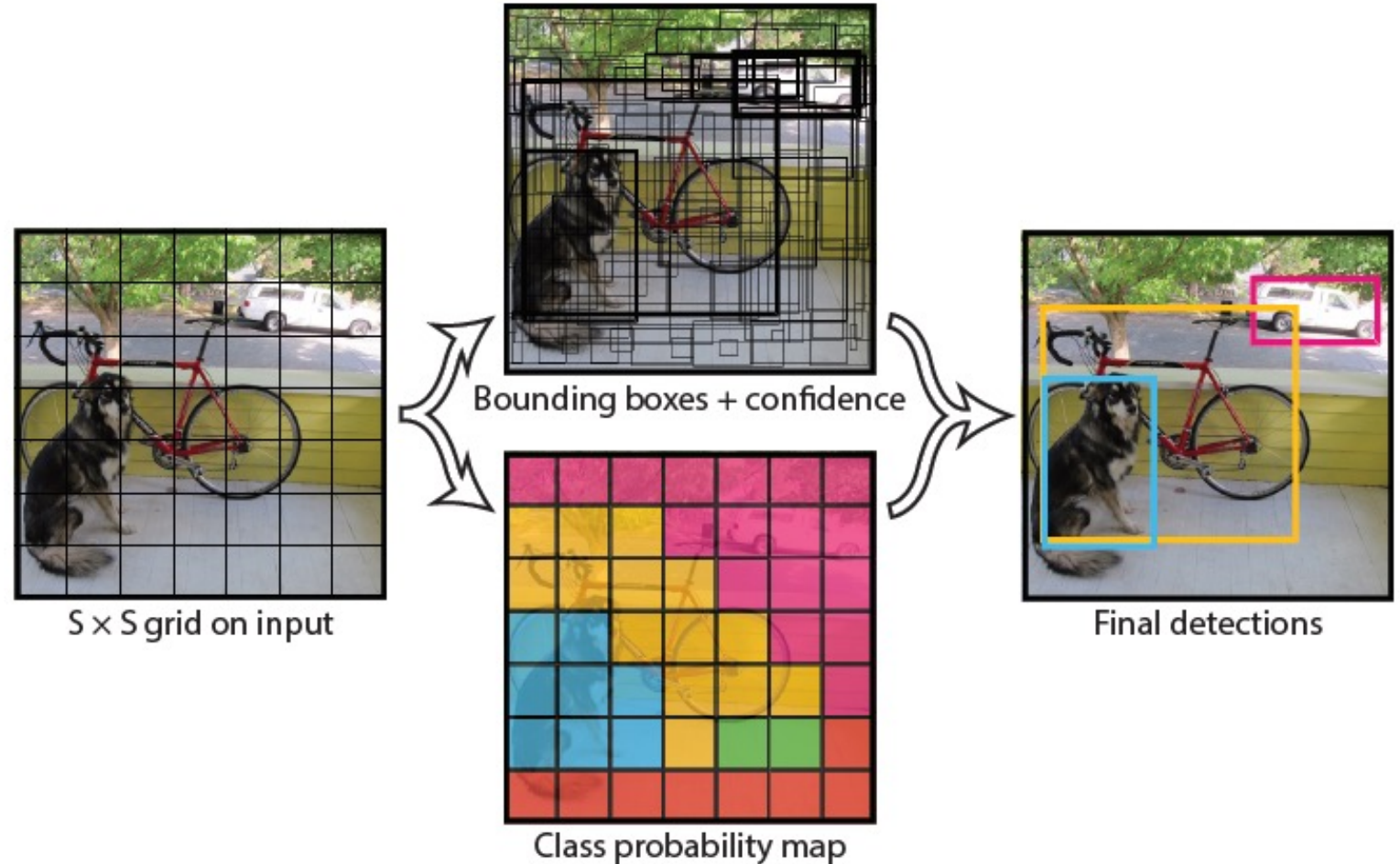


# Single-shot Object Detectors

- No two-steps of box proposals + Classification
- Anchor Points for predicting boxes

# YOLO- You Only Look Once

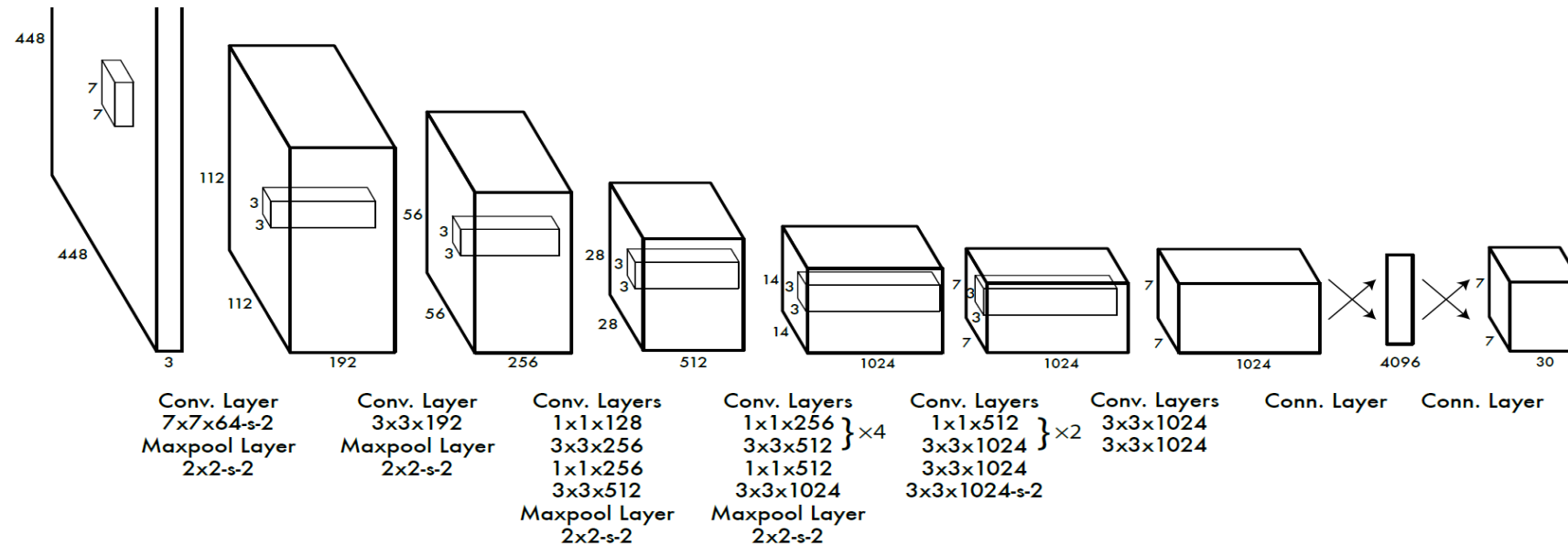
Idea: No bounding box proposals.  
Predict a class and a box for every location in a grid.



<https://arxiv.org/abs/1506.02640>

Redmon et al. CVPR 2016.

# YOLO- You Only Look Once



Divide the image into 7x7 cells.

Each cell trains a detector.

The detector needs to predict the object's class distributions.

The detector has 2 bounding-box predictors to predict bounding-boxes and confidence scores.

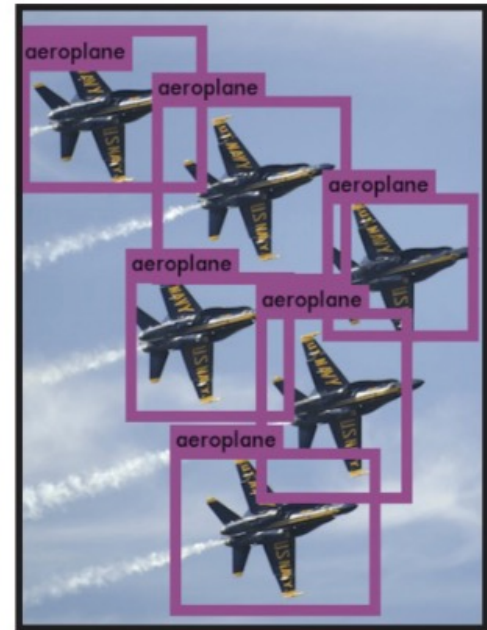
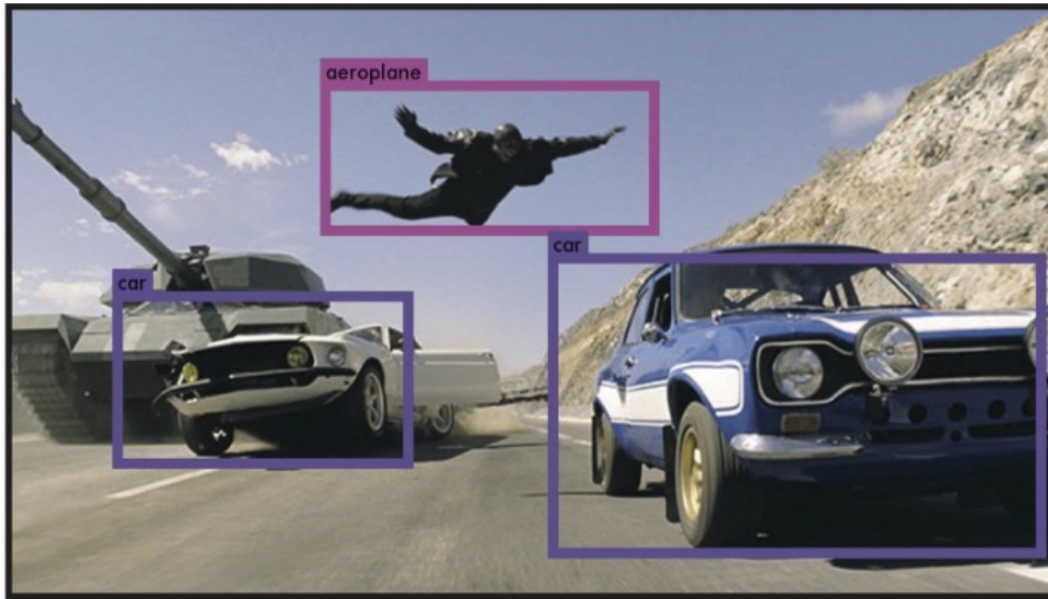
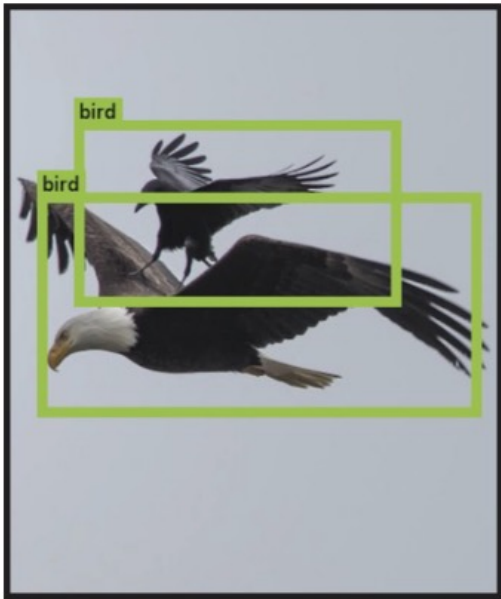
<https://arxiv.org/abs/1506.02640>

Redmon et al. CVPR 2016.

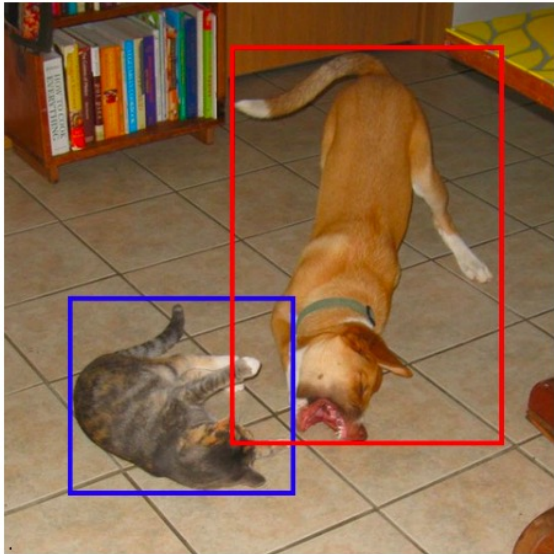


# YOLO - Loss Function

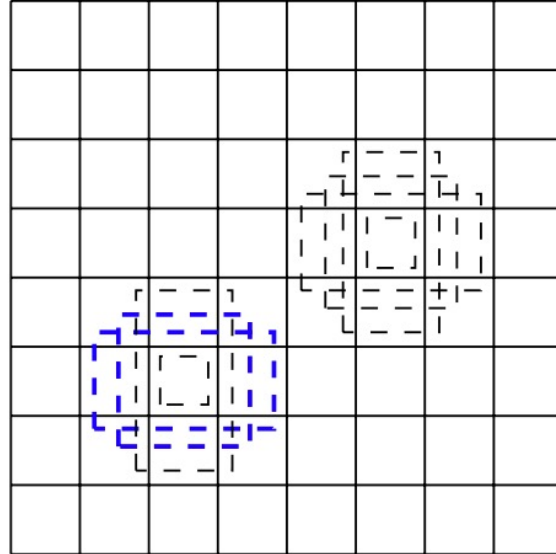
$$\begin{aligned} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ \left( \sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left( \sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\ & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\ & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \end{aligned}$$



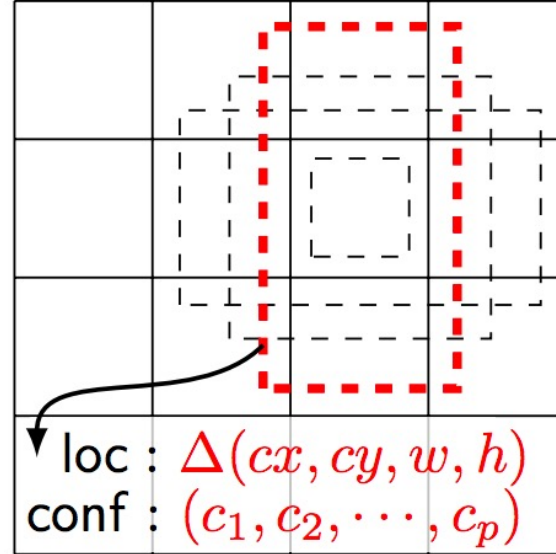
# SSD: Single Shot Detector



(a) Image with GT boxes



(b)  $8 \times 8$  feature map

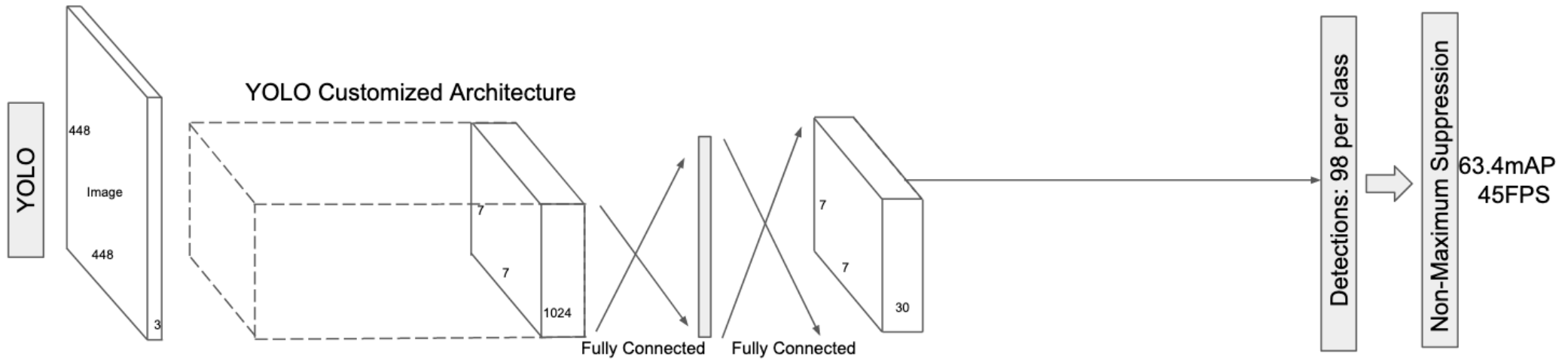
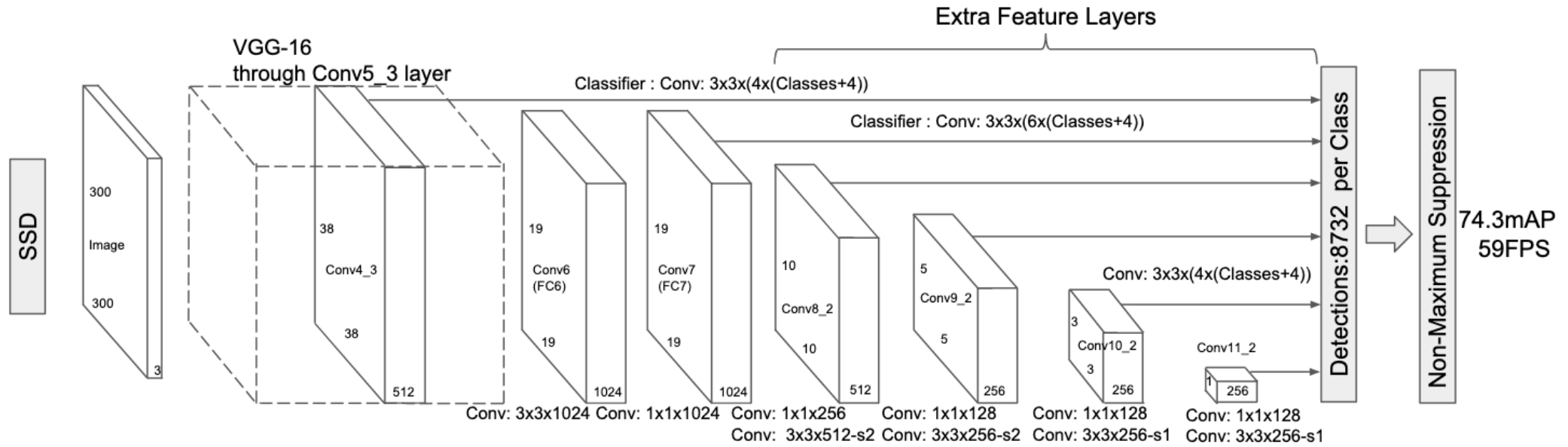


(c)  $4 \times 4$  feature map

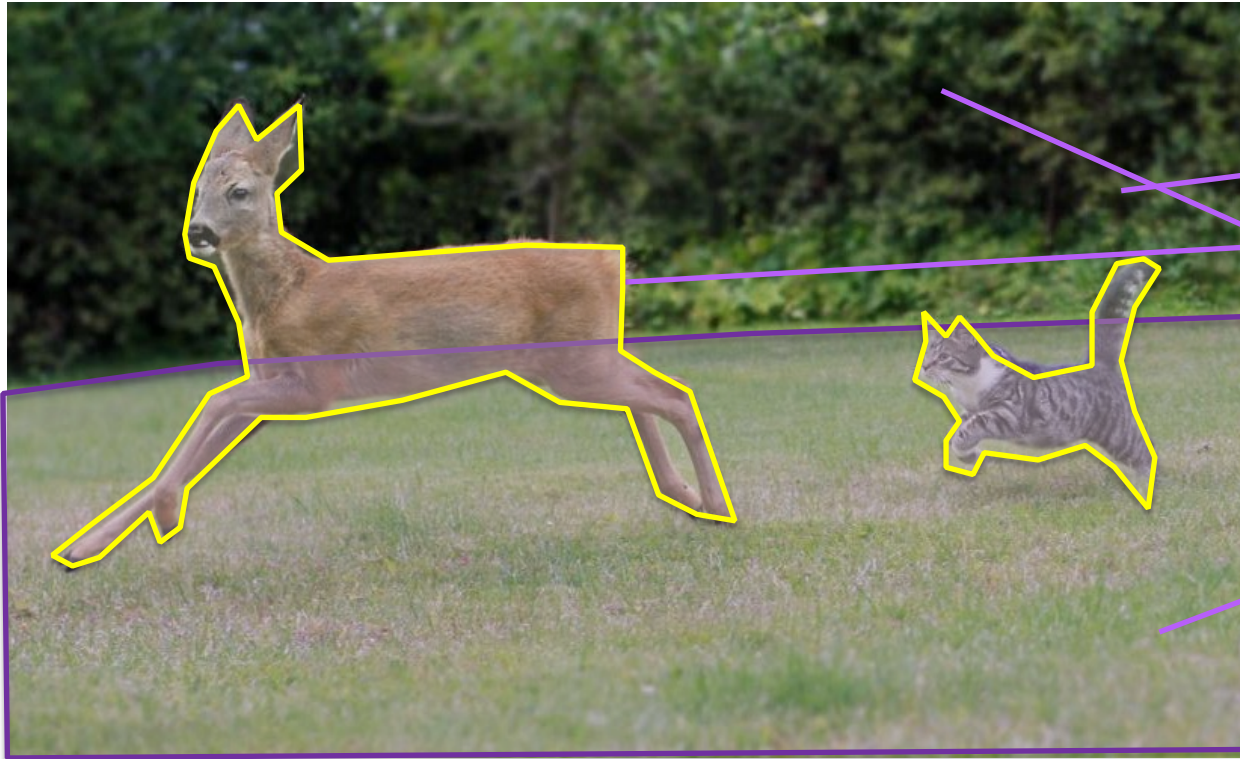
Idea: Similar to YOLO, but denser grid map, multiscale grid maps. + Data augmentation + Hard negative mining + Other design choices in the network.



# SSD vs YOLO

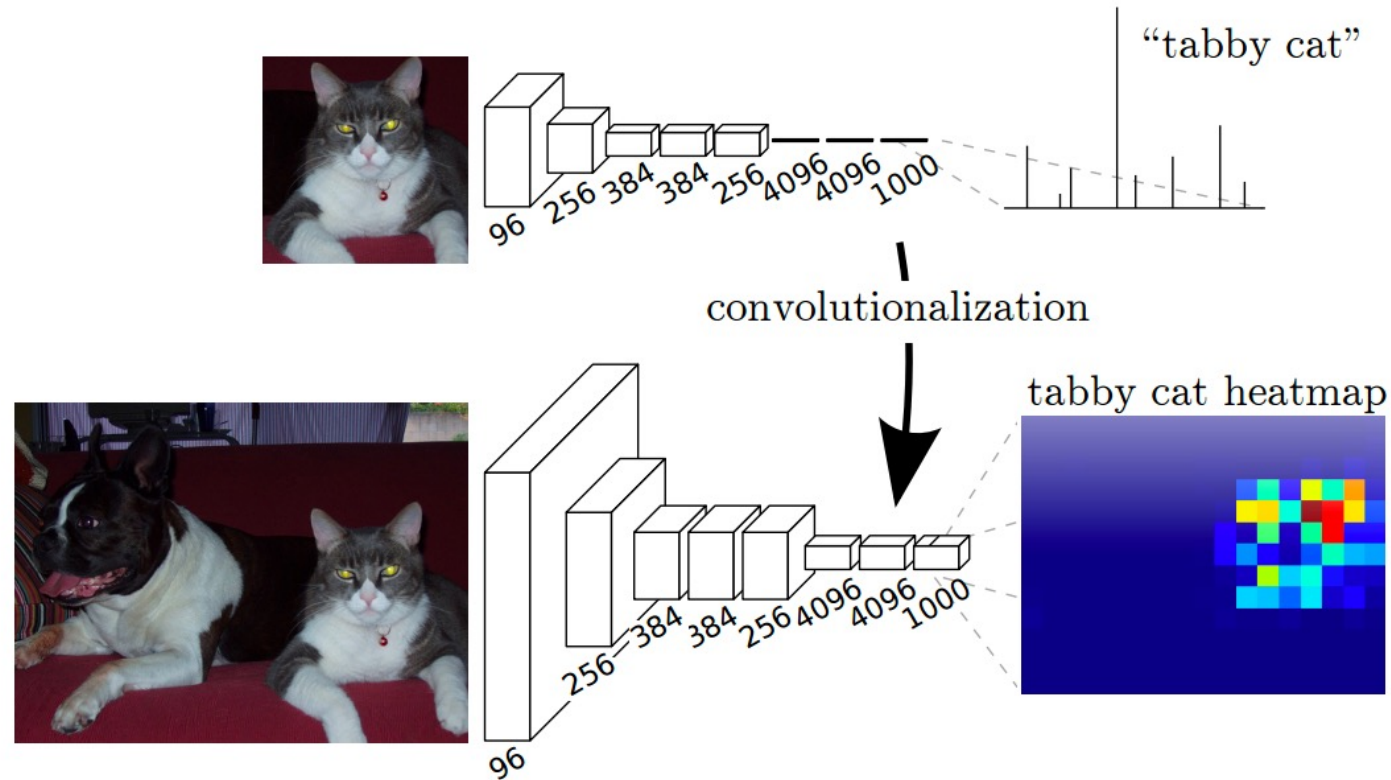


# Semantic Segmentation / Image Parsing



deer  
cat  
trees  
grass

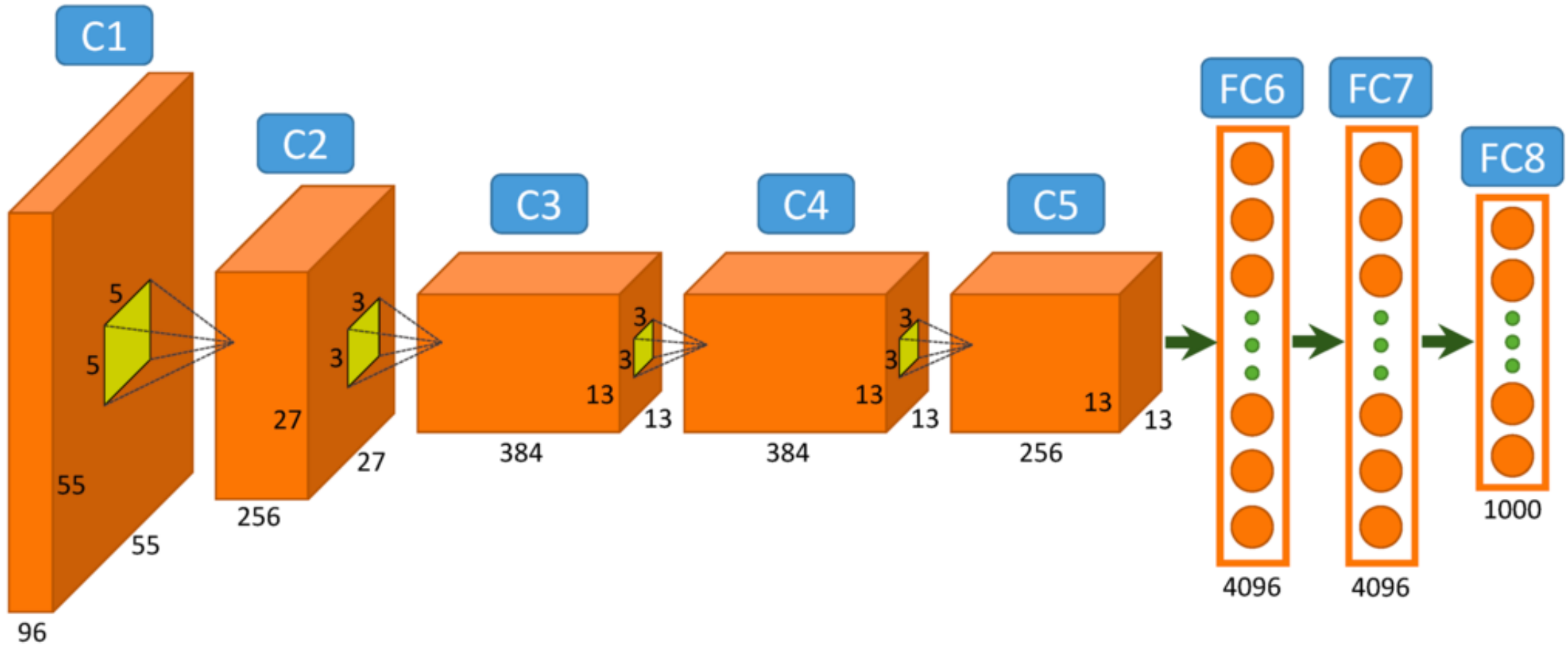
# Idea 1: Convolutionalization



However resolution of the segmentation map is low.

[https://people.eecs.berkeley.edu/~jonlong/long\\_shelhamer\\_fcn.pdf](https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf)

# Alexnet





# Idea 1: Convolutionalization

`nn.Linear(4096, 1000)` == `nn.Conv2D(4096, 1000, kernel_size = 1, stride = 1)`

input tensor:  
4096



Linear-layer  
W: 4096 x 1000  
b: 1000



output tensor:  
1000



≡

input tensor:  
4096x1x1



SpatialConv  
W: 1000x4096x1x1  
b: 1000



output tensor:  
1000x1x1



# Fully Convolutional Networks (CVPR 2015)

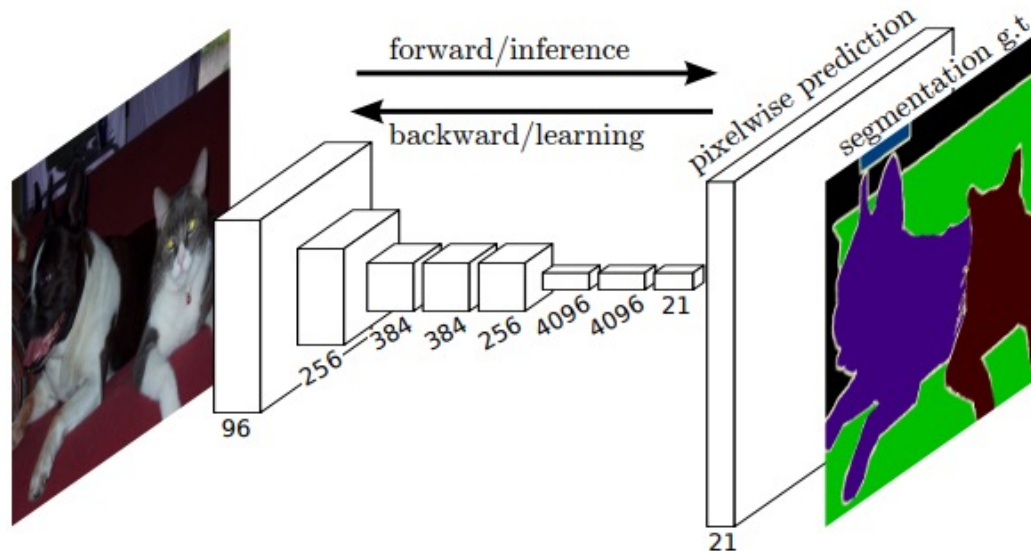
## Fully Convolutional Networks for Semantic Segmentation

Jonathan Long\*

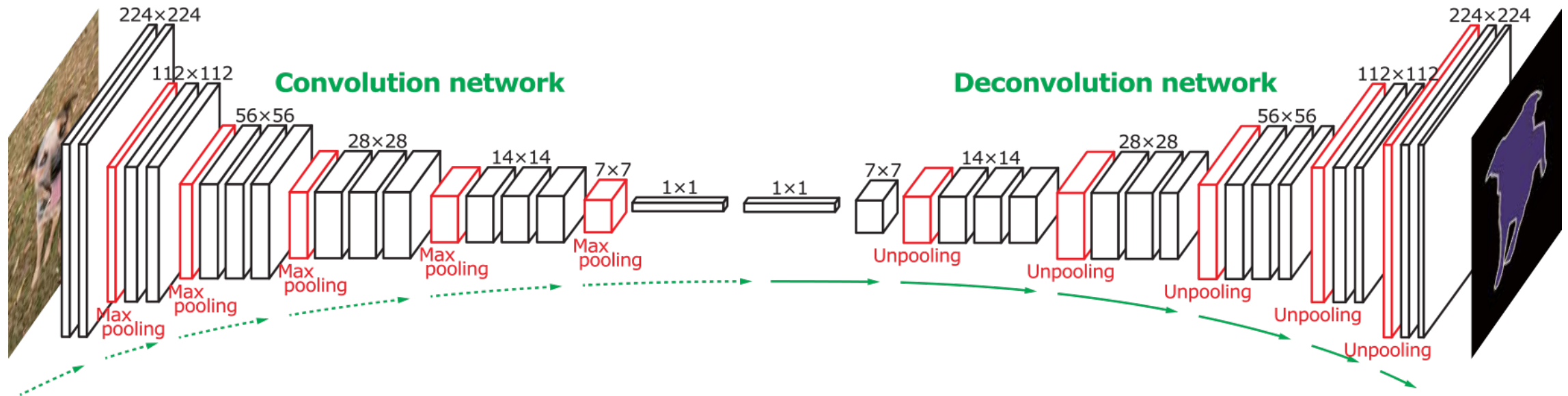
Evan Shelhamer\*  
UC Berkeley

Trevor Darrell

{jonlong, shelhamer, trevor}@cs.berkeley.edu



# Idea 2: Up-sampling Convolutions or "Deconvolutions"

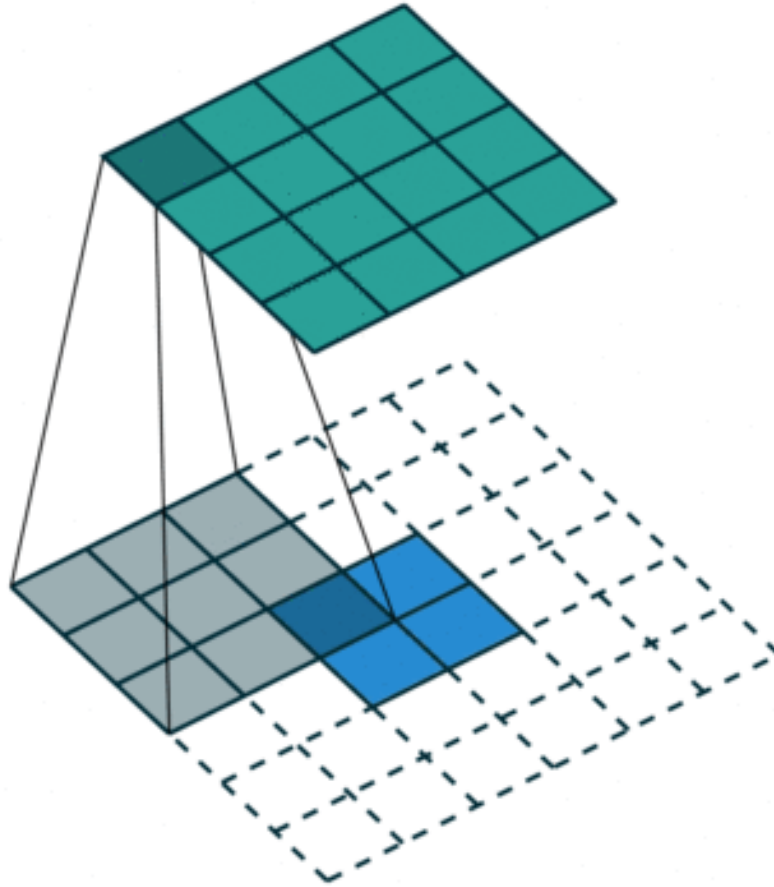


## Learning Deconvolution Network for Semantic Segmentation

Hyeonwoo Noh      Seunghoon Hong      Bohyung Han  
Department of Computer Science and Engineering, POSTECH, Korea  
{hyeonwoonoh\_, maga33, bhhan}@postech.ac.kr

<http://cvlab.postech.ac.kr/research/deconvnet/>

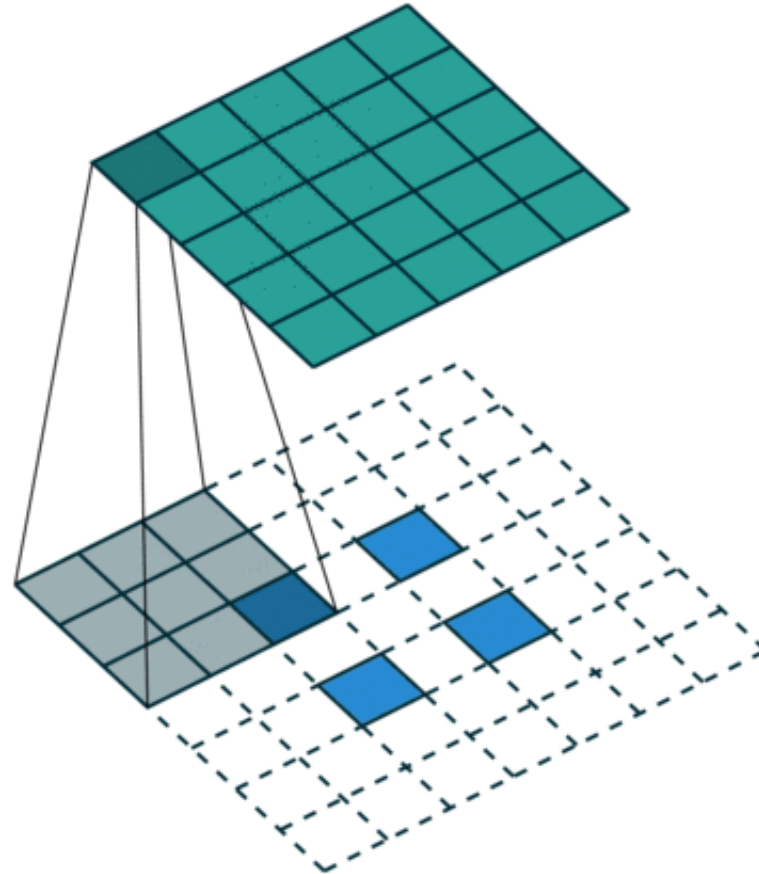
## Idea 2: Up-sampling Convolutions or "Deconvolutions"



[https://github.com/vdumoulin/conv\\_arithmetic](https://github.com/vdumoulin/conv_arithmetic)



## Idea 2: Up-sampling Convolutions or "Deconvolutions"



[https://github.com/vdumoulin/conv\\_arithmetic](https://github.com/vdumoulin/conv_arithmetic)

## Idea 2: Up-sampling Convolutions or "Deconvolutions"

Deconvolutional Layers

Upconvolutional Layers

Backwards Strided  
Convolutional Layers

Fractionally Strided  
Convolutional Layers

Transposed  
Convolutional Layers

Spatial Full  
Convolutional Layers

# Pytorch

Docs > [torch.nn](#) > ConvTranspose2d



## CONVTRANSPOSE2D

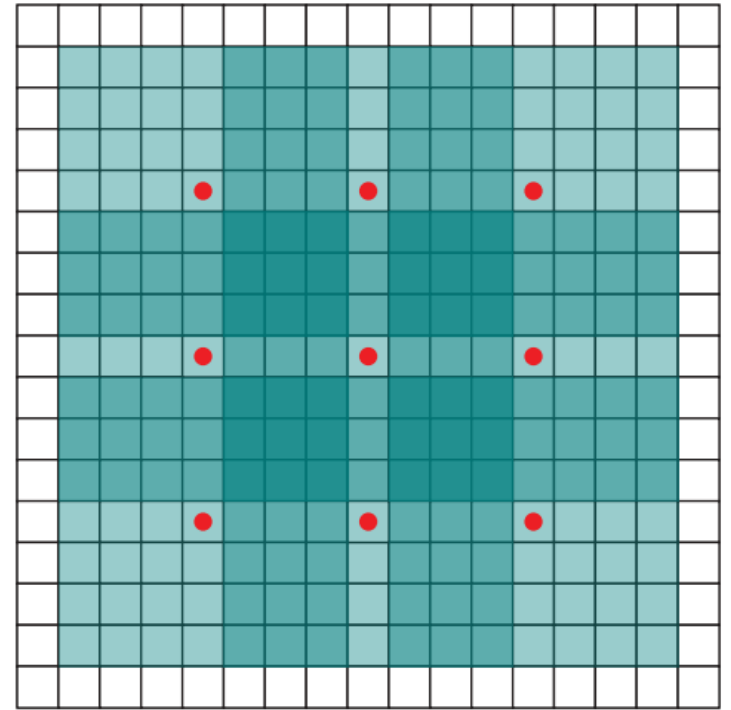
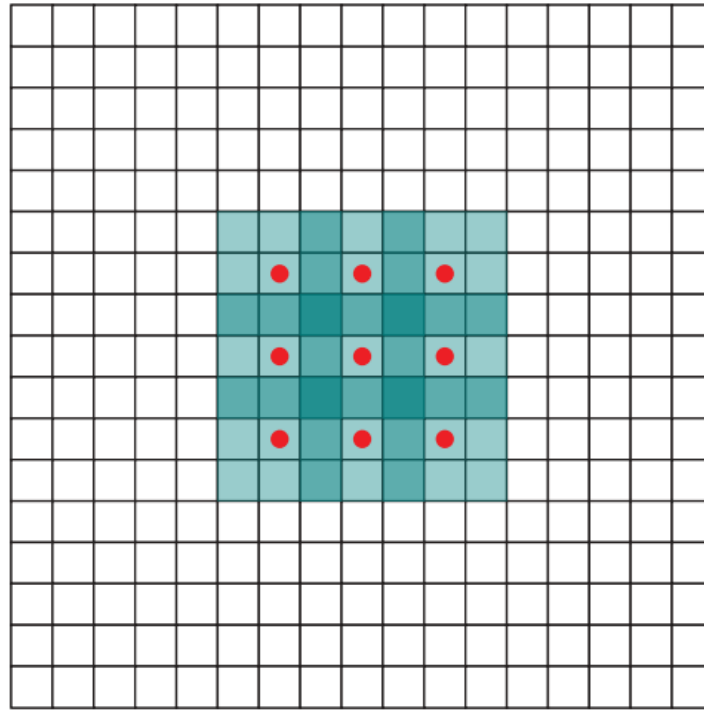
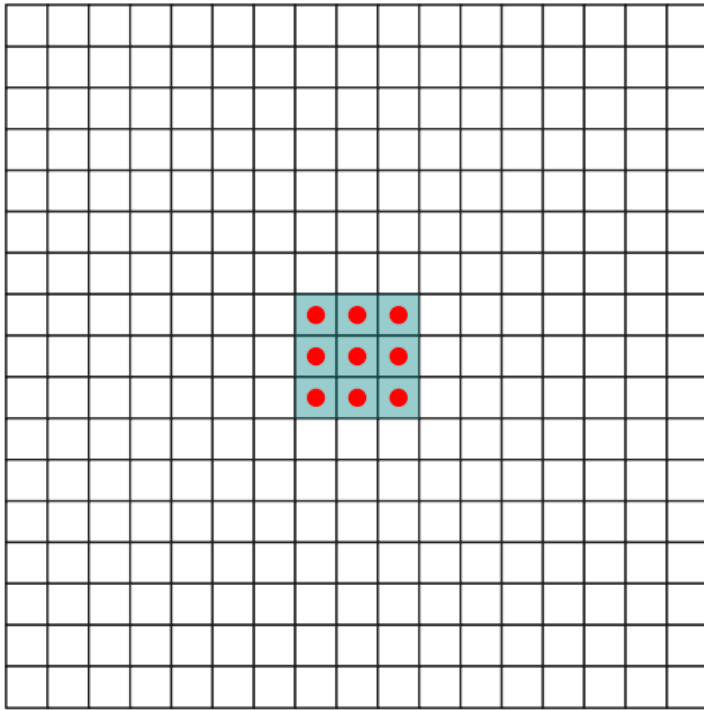
```
CLASS torch.nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride=1, padding=0,  
output_padding=0, groups=1, bias=True, dilation=1, padding_mode='zeros', device=None,  
dtype=None) \[SOURCE\]
```

Applies a 2D transposed convolution operator over an input image composed of several input planes.

This module can be seen as the gradient of Conv2d with respect to its input. It is also known as a fractionally-strided convolution or a deconvolution (although it is not an actual deconvolution operation as it does not compute a true inverse of convolution). For more information, see the visualizations [here](#) and the [Deconvolutional Networks](#) paper.

This module supports [TensorFloat32](#).

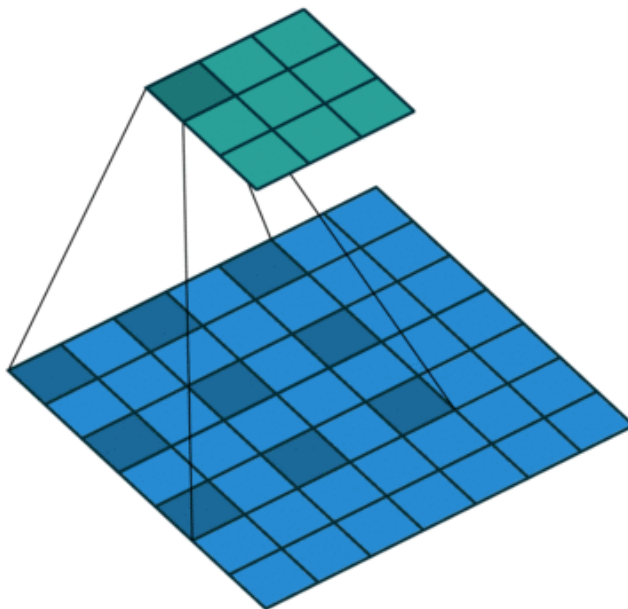
# Idea 3: Dilated Convolutions



MULTI-SCALE CONTEXT AGGREGATION BY  
DILATED CONVOLUTIONS



# Idea 3: Dilated Convolutions



MULTI-SCALE CONTEXT AGGREGATION BY  
DILATED CONVOLUTIONS

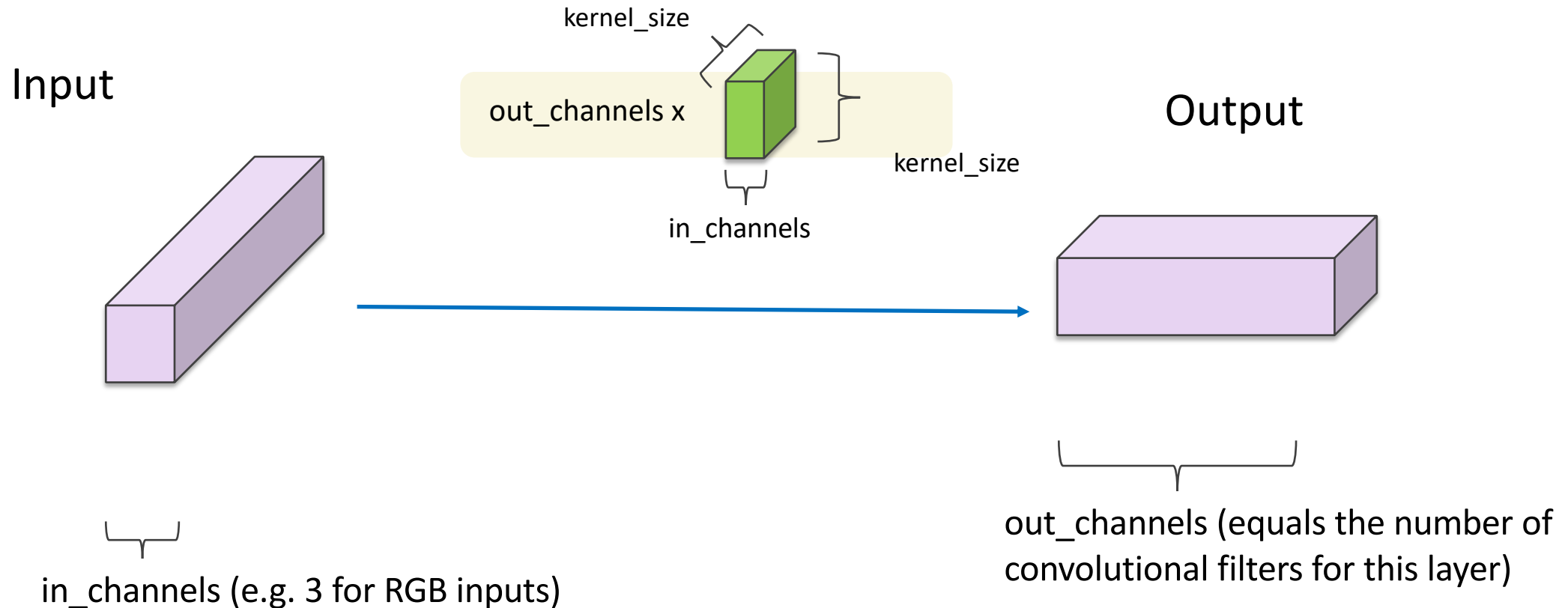
**Fisher Yu**  
Princeton University

**Vladlen Koltun**  
Intel Labs

ICLR 2016

# Convolutional Layer in pytorch

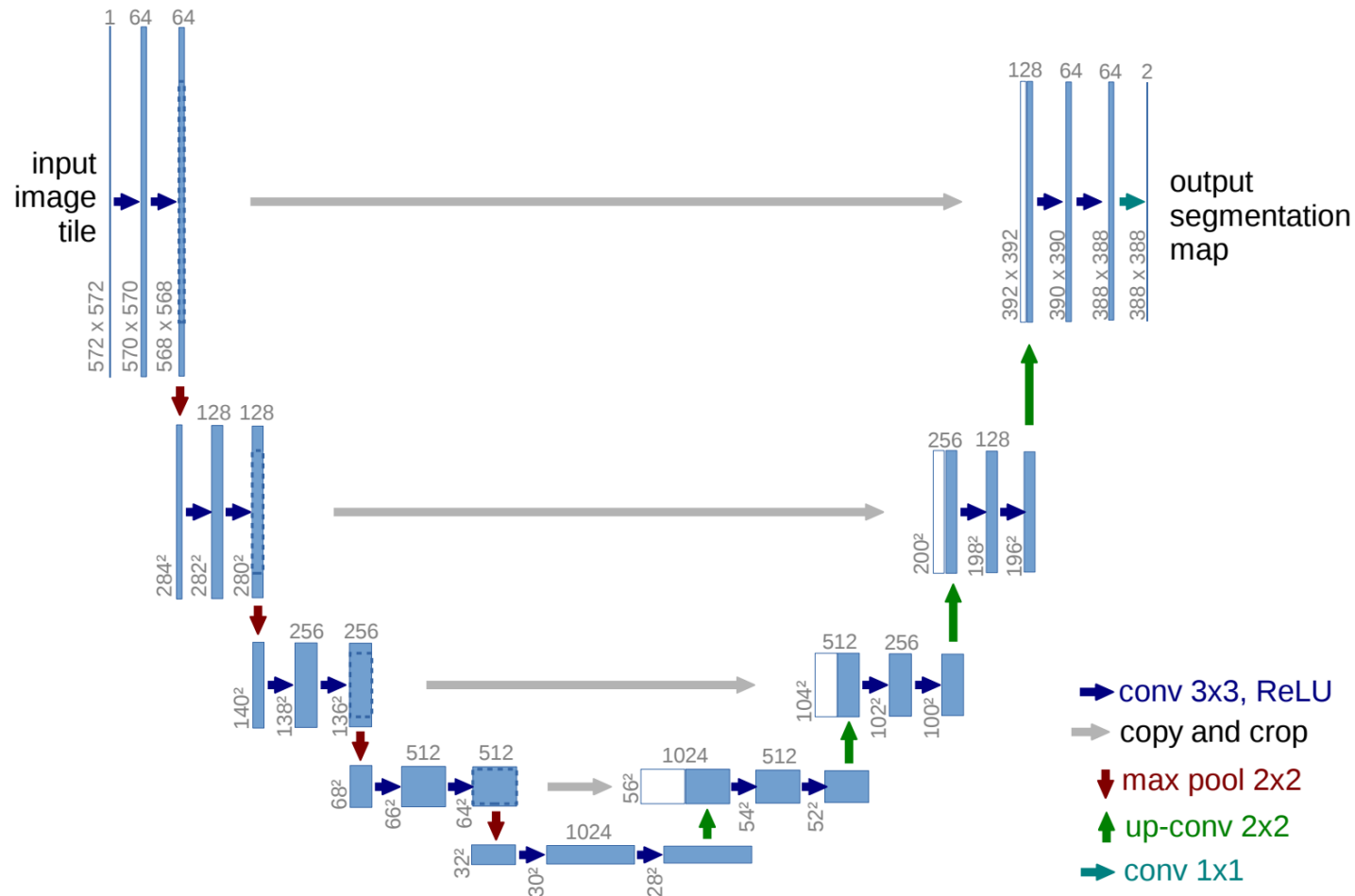
```
class torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True) \[source\]
```



# U-Net: Convolutional Networks for Biomedical Image Segmentation

Olaf Ronneberger, Philipp Fischer, and Thomas Brox

Computer Science Department and BIOS Centre for Biological Signalling Studies,  
University of Freiburg, Germany



<https://arxiv.org/abs/1505.04597>

<https://github.com/milesial/Pytorch-UNet>

<https://github.com/usuyama/pytorch-unet>

# UNet in Pytorch

```
from .unet_parts import *

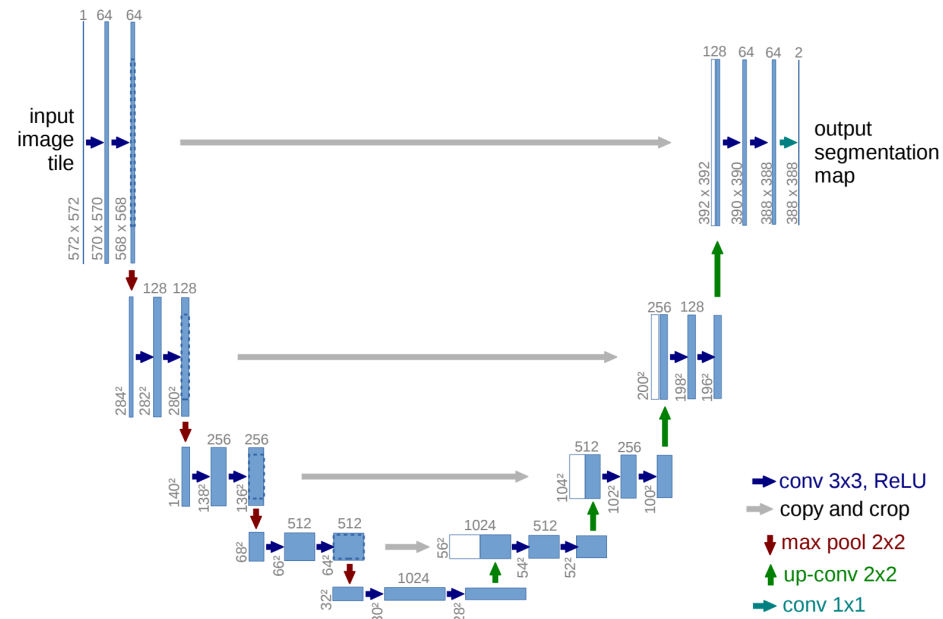
class UNet(nn.Module):
    def __init__(self, n_channels, n_classes, bilinear=False):
        super(UNet, self).__init__()
        self.n_channels = n_channels
        self.n_classes = n_classes
        self.bilinear = bilinear

        self.inc = (DoubleConv(n_channels, 64))
        self.down1 = (Down(64, 128))
        self.down2 = (Down(128, 256))
        self.down3 = (Down(256, 512))
        factor = 2 if bilinear else 1
        self.down4 = (Down(512, 1024 // factor))
        self.up1 = (Up(1024, 512 // factor, bilinear))
        self.up2 = (Up(512, 256 // factor, bilinear))
        self.up3 = (Up(256, 128 // factor, bilinear))
        self.up4 = (Up(128, 64, bilinear))
        self.outc = (OutConv(64, n_classes))

    def forward(self, x):
        x1 = self.inc(x)
        x2 = self.down1(x1)
        x3 = self.down2(x2)
        x4 = self.down3(x3)
        x5 = self.down4(x4)
        x = self.up1(x5, x4)
        x = self.up2(x, x3)
        x = self.up3(x, x2)
        x = self.up4(x, x1)
        logits = self.outc(x)
        return logits
```



# Chair segmentation - Training



## Chair Segments: A Compact Benchmark for the Study of Object Segmentation

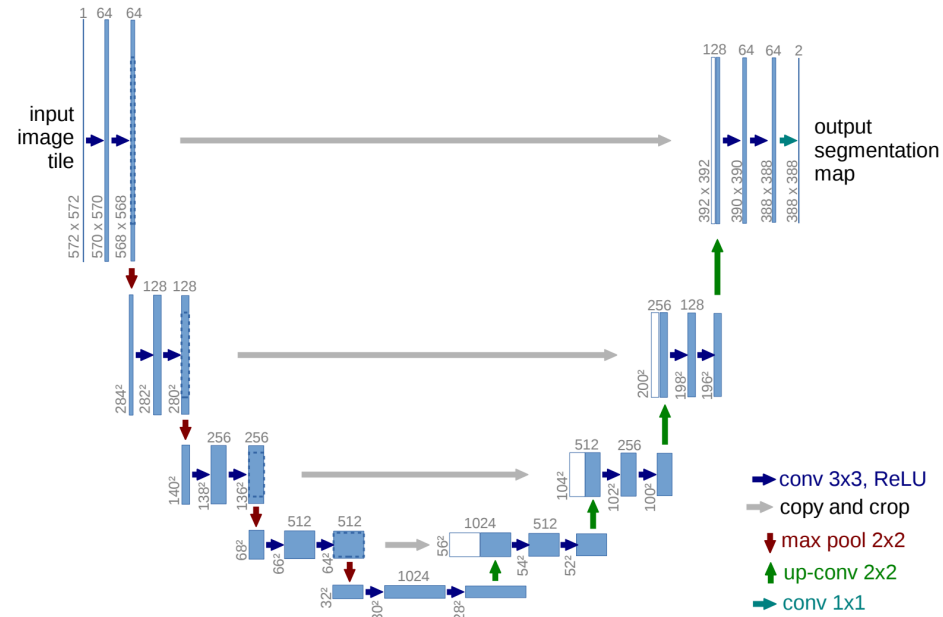
Leticia Pinto-Alva<sup>††</sup>, Ian K. Torres<sup>‡\*</sup>, Rosangel Garcia<sup>§\*</sup>, Ziyang Yang<sup>†</sup>, Vicente Ordonez<sup>†</sup>

<sup>‡</sup>Universidad Católica San Pablo, <sup>‡</sup>University of Massachusetts, Amherst, <sup>§</sup>Le Moyne College,

<sup>†</sup>University of Virginia

lp2rv@virginia.edu, zy3cx@virginia.edu, vicente@virginia.edu

# Chair segmentation - Prediction



## Chair Segments: A Compact Benchmark for the Study of Object Segmentation

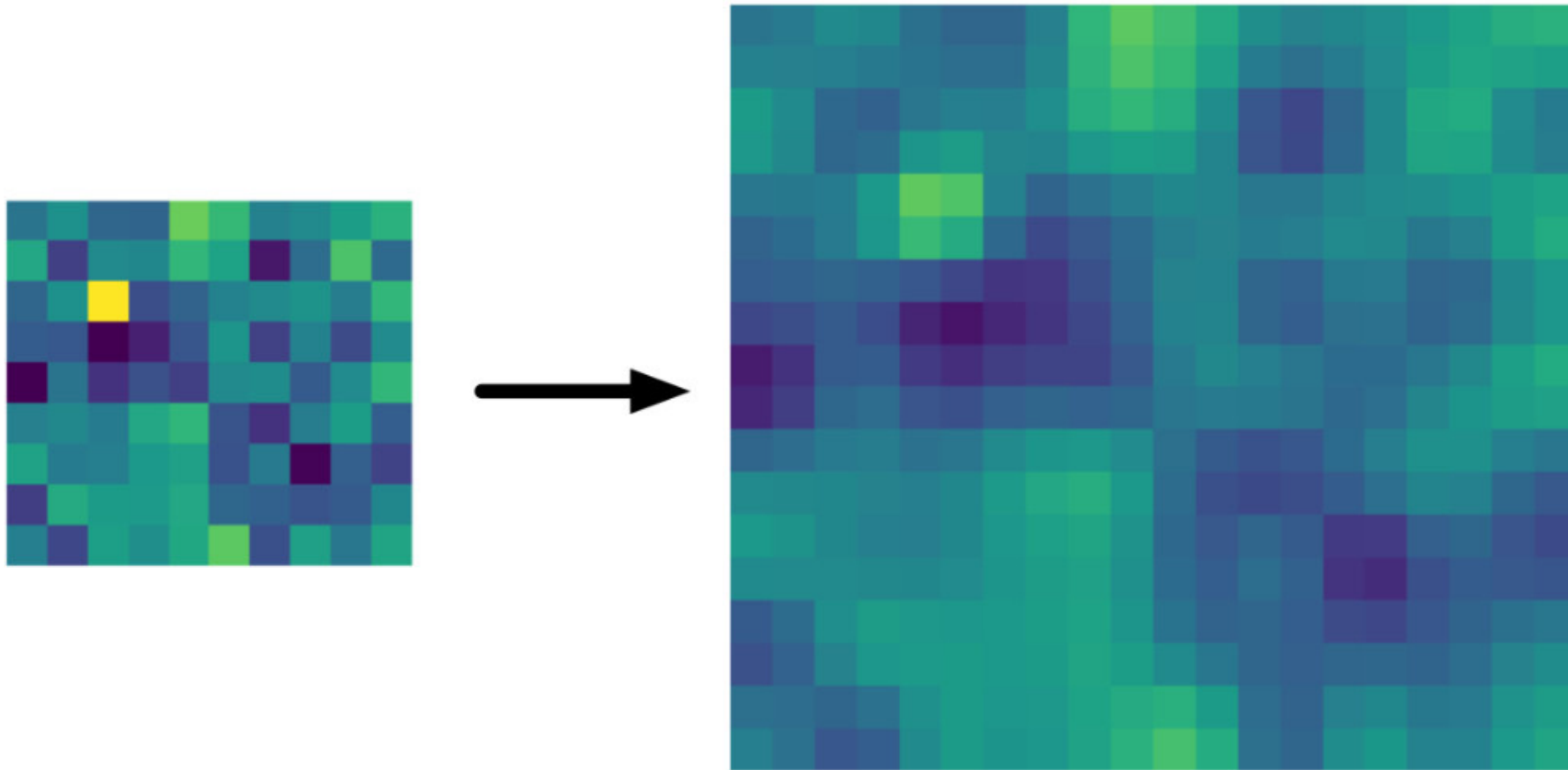
Leticia Pinto-Alva<sup>††</sup>, Ian K. Torres<sup>‡</sup>, Rosangel Garcia<sup>§\*</sup>, Ziyang Yang<sup>†</sup>, Vicente Ordonez<sup>†</sup>

<sup>‡</sup>Universidad Católica San Pablo, <sup>‡</sup>University of Massachusetts, Amherst, <sup>§</sup>Le Moyne College,

<sup>†</sup>University of Virginia

lp2rv@virginia.edu, zy3cx@virginia.edu, vicente@virginia.edu

# Bilinear Upsampling Layer

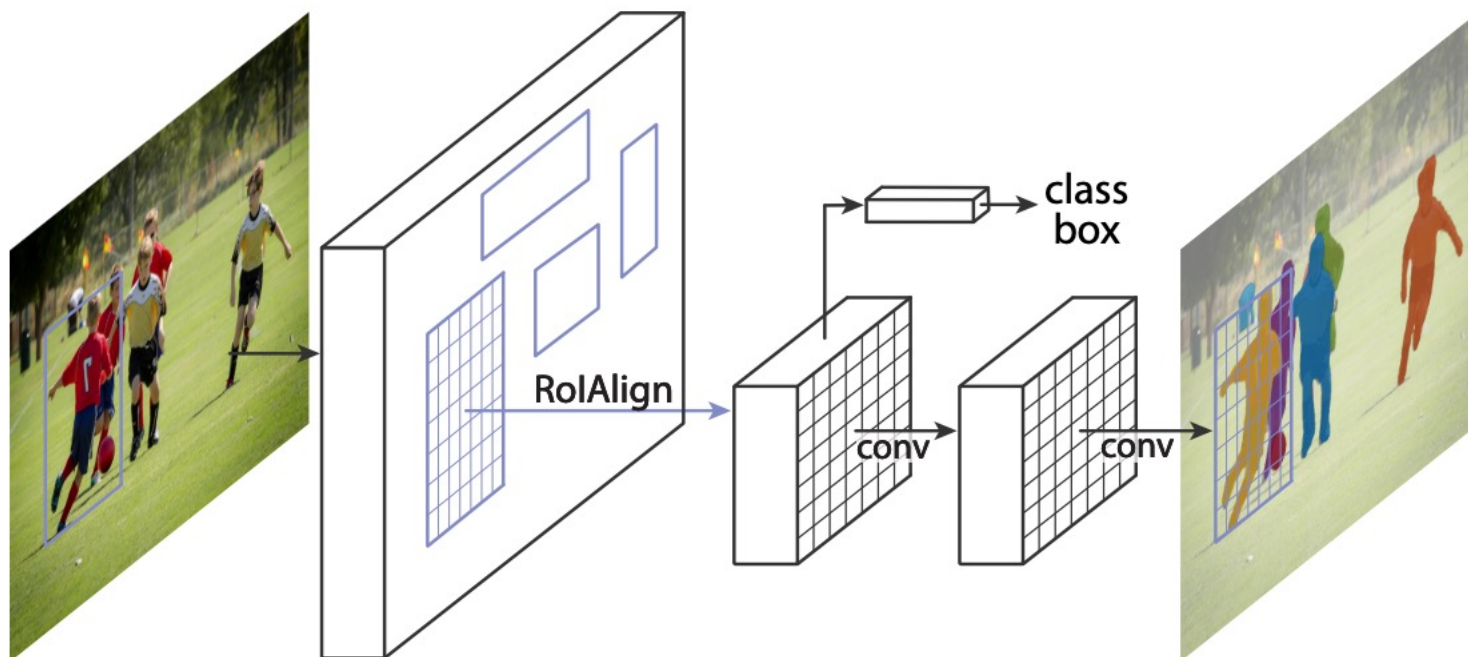


<https://machinethink.net/blog/coreml-upsampling/>

# Mask R-CNN

Kaiming He Georgia Gkioxari Piotr Dollár Ross Girshick

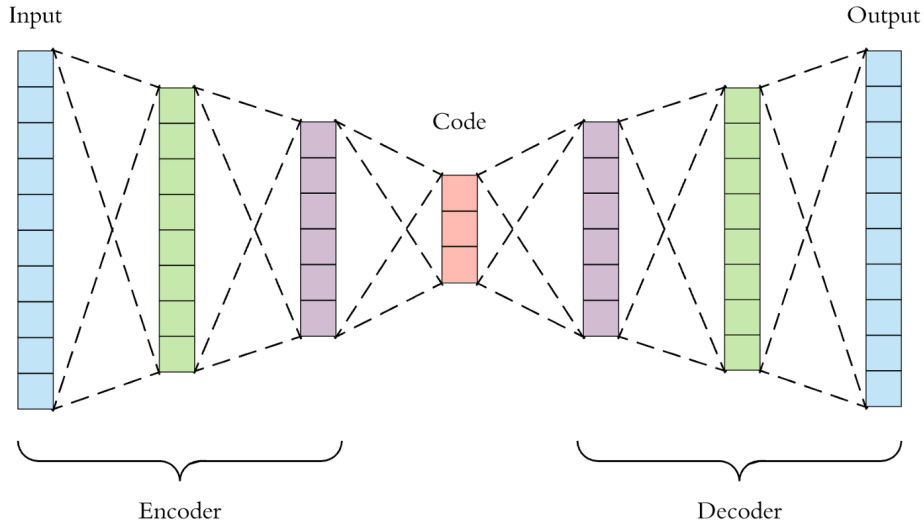
Facebook AI Research (FAIR)



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

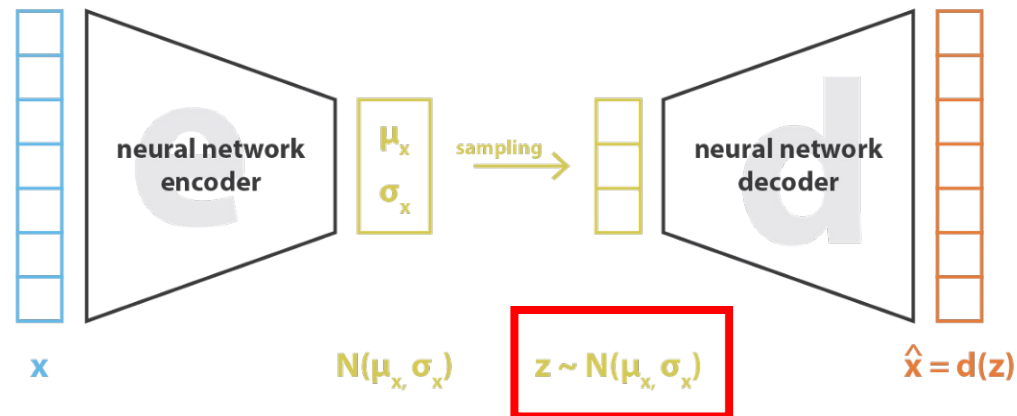
<https://arxiv.org/abs/1703.06870>

# AutoEncoder Models (Downsample, Upsample)





# Variational AutoEncoders (VAE)



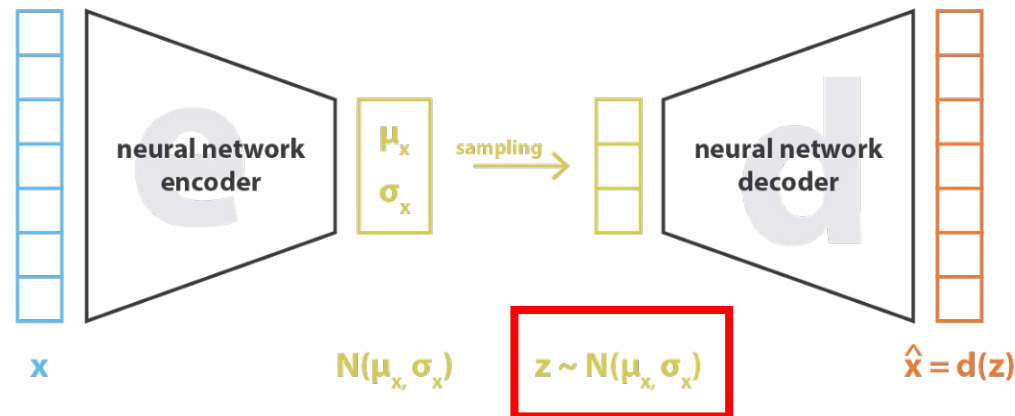
$$\text{loss} = \|x - \hat{x}\|^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)]$$

# Reparameterization “trick”

$$z = z\_mean + sigma * epsilon$$

$$sigma = exp(z\_log\_var/2)$$

$$\epsilon \sim \text{Normal}(0,1)$$



$$\text{loss} = \|x - \hat{x}\|^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)]$$

# Questions