

# Object Detection and Segmentation

Jisang Han

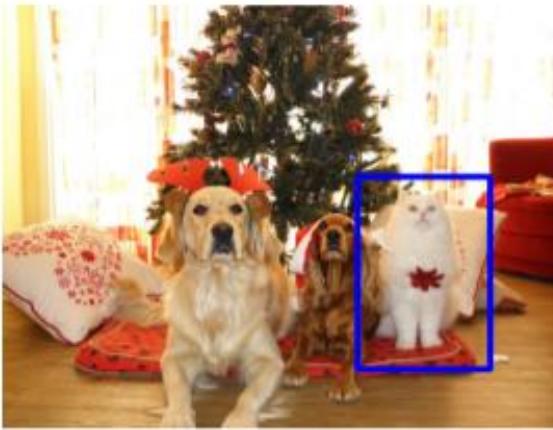
onground@korea.ac.kr

Artificial Intelligence in KU (AIKU)

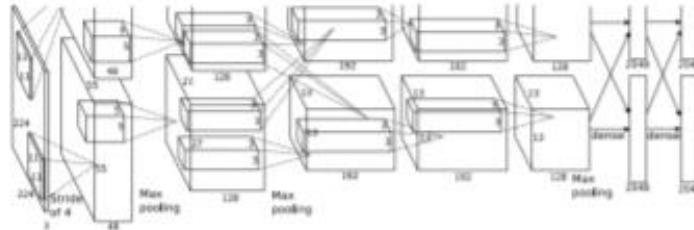
Department of Computer Science and Engineering, Korea University

**AIKU**

# Detecting Multiple Objects: Sliding Window



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

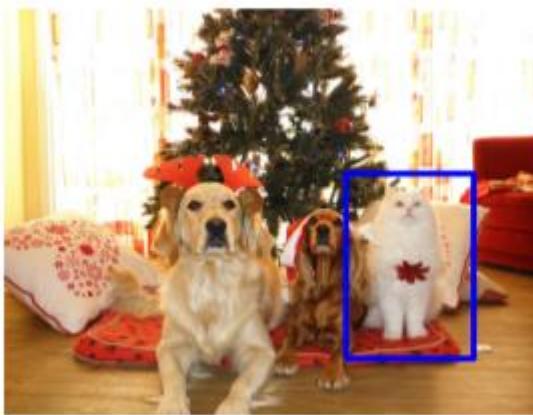


Dog? NO  
Cat? YES  
Background? NO

CNN is used for image classification. Therefore, we can solve the problem by reducing object detection to classification using CNN. Here, we add one more class called **background** in the basic classification.

We then **apply CNN** to many different regions of the input image to classify whether the region is dog, cat, or background.

# Detecting Multiple Objects: Sliding Window



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

800 x 600 image  
has ~58M boxes!  
No way we can evaluate them all

**Question:** How many possible boxes are there in an image of size  $H \times W$ ?

Consider a box of size  $h \times w$ :  
Possible x positions:  $W - w + 1$   
Possible y positions:  $H - h + 1$   
Possible positions:  
 $(W - w + 1) * (H - h + 1)$

Total possible boxes:

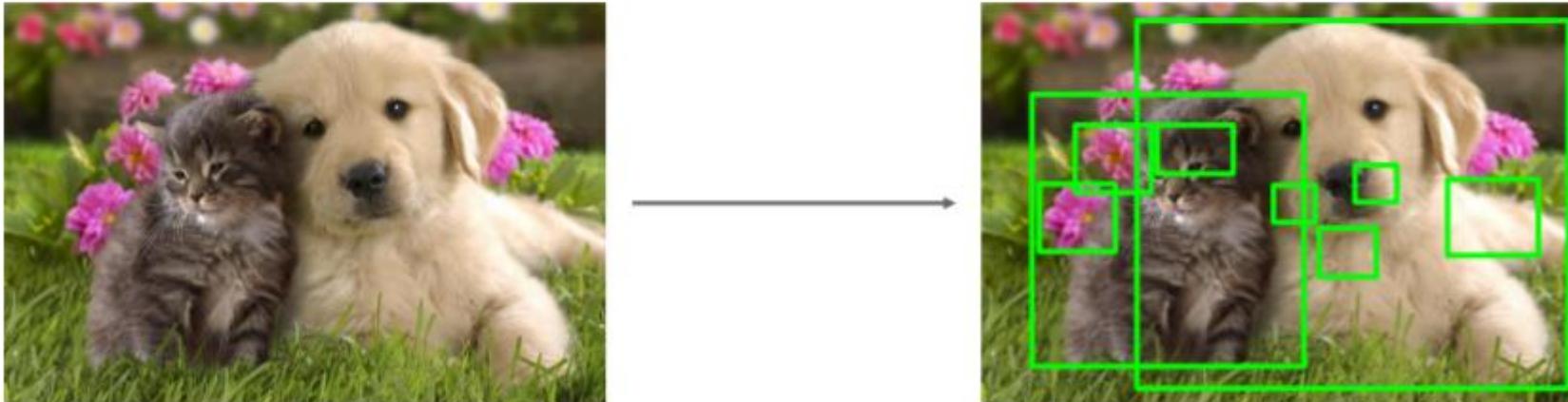
$$\sum_{h=1}^H \sum_{w=1}^W (W - w + 1)(H - h + 1)$$
$$= \frac{H(H + 1)}{2} \frac{W(W + 1)}{2}$$

**58 million boxes are required** to calculate this at 800x600. There is no way to calculate this. Even if calculation is possible, there will also be a problem of **detecting the same object in a large number of boxes.**

# Region Proposals

---

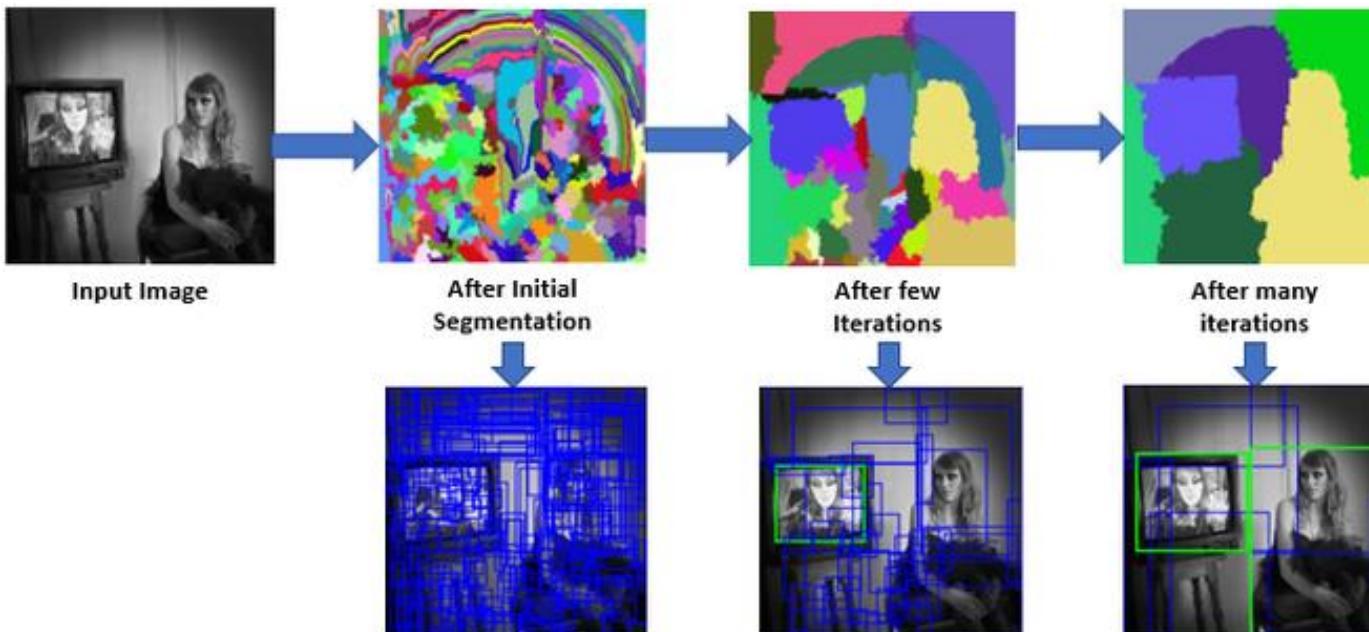
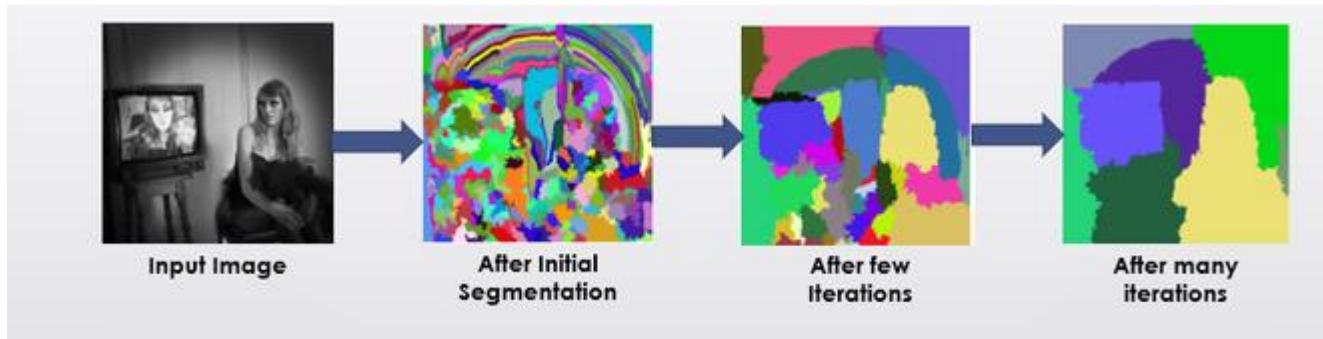
- Find a small set of boxes that are likely to cover all objects
- Often based on heuristics: e.g. look for “blob-like” image regions
- Relatively fast to run; e.g. Selective Search gives 2000 region proposals in a few seconds on CPU



Is a way to solve the problem.

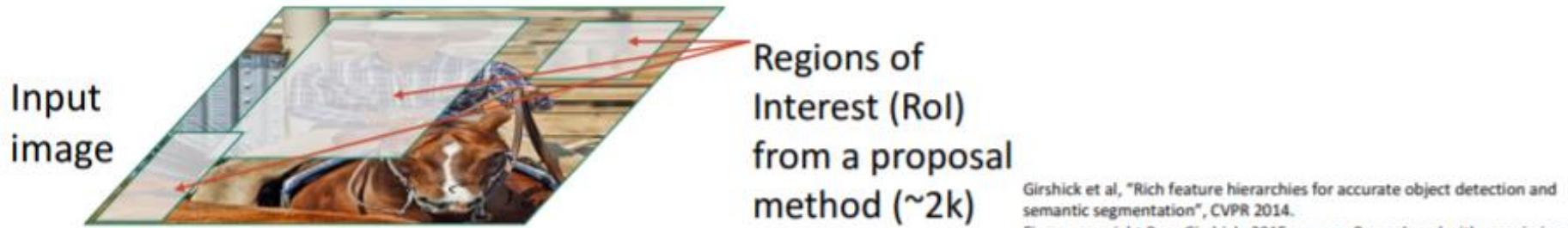
Instead of making all the boxes, make "**candidate regions**." Select regions to cover all objects in the image. (set of candidate regions) -> **Selective Search**

# Region Proposals



# R-CNN: Region-Based CNN

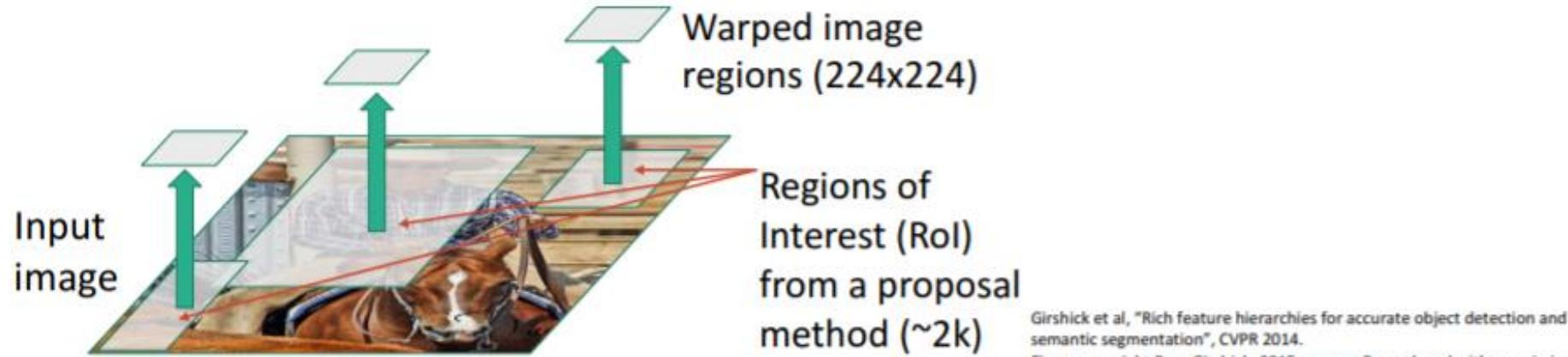
---



First, we prepare an input image and **extract about 2000 region proposals** using the region proposal algorithm.(=RoI)

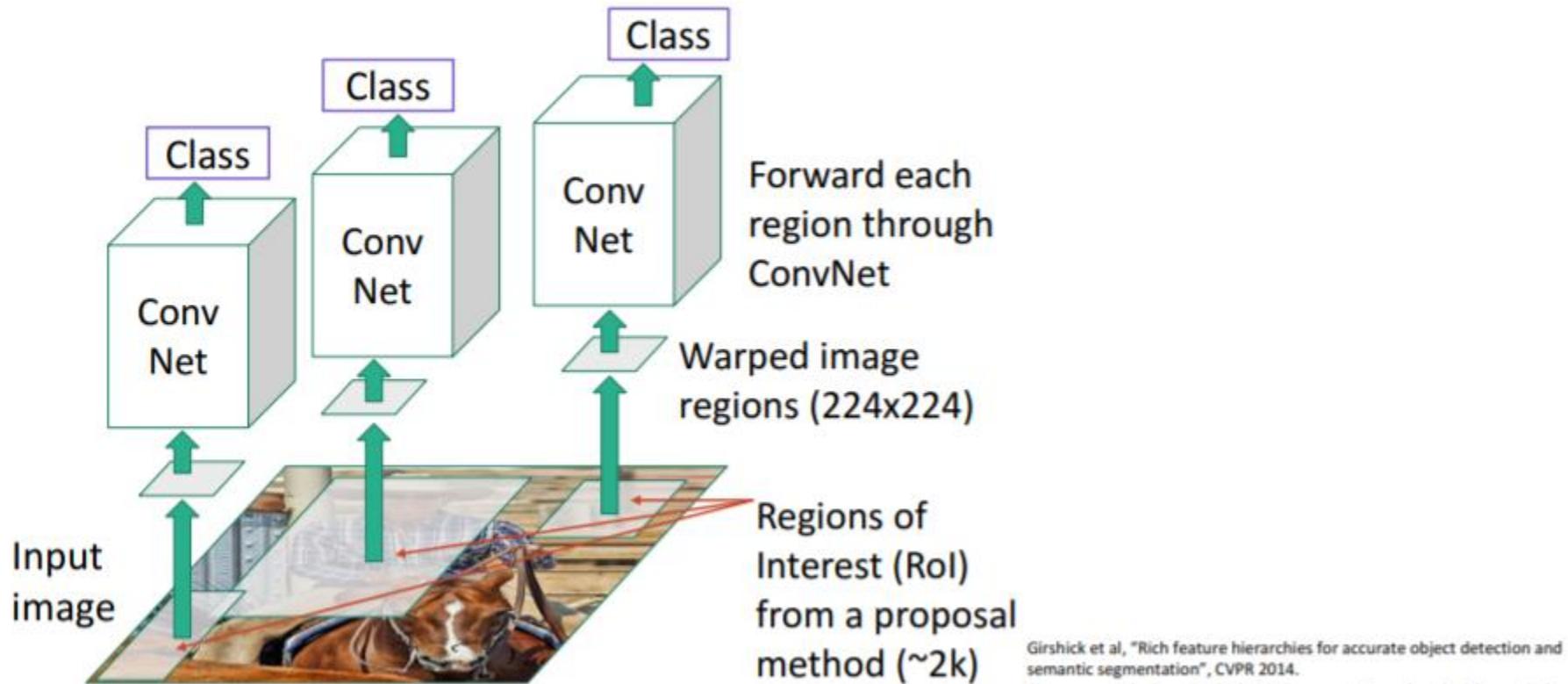
# R-CNN: Region-Based CNN

---



The extracted region proposals are **different in size and aspect ratio**, so they are all **warped** into **224x224 images**.

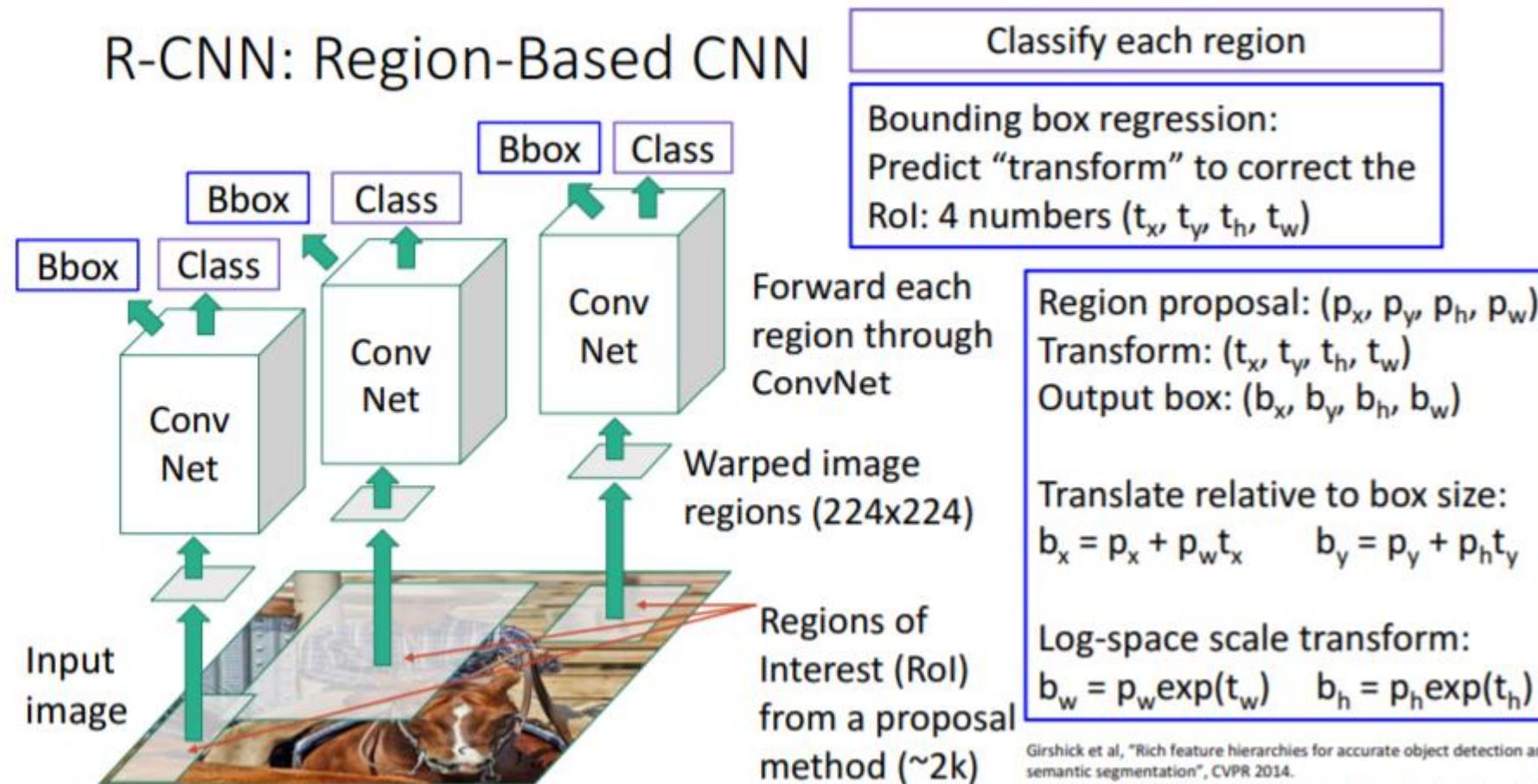
# R-CNN: Region-Based CNN



Then, the **class score** is calculated by passing the warped images **through the Conv Net**. This score tells you whether it is a background or “a” category.

# R-CNN: Region-Based CNN

But what if the **extracted region proposals don't exist on the object we want to detect?** Also, it is **not possible to learn region proposals boxes.** Therefore, the following methods are used.



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: Region-Based CNN

---

Use the **bounding box regression** to transform the **region proposal boxes** into **bounding boxes**. This modifies the region proposals to fit a little more into the object. ConvNet outputs how much to transform ( $= (t_x, t_y, t_h, t_w)$ ) and transforms existing **region proposal** ( $p_x, p_y, p_h, p_w$ )

$$b_x = p_x + p_w t_w$$

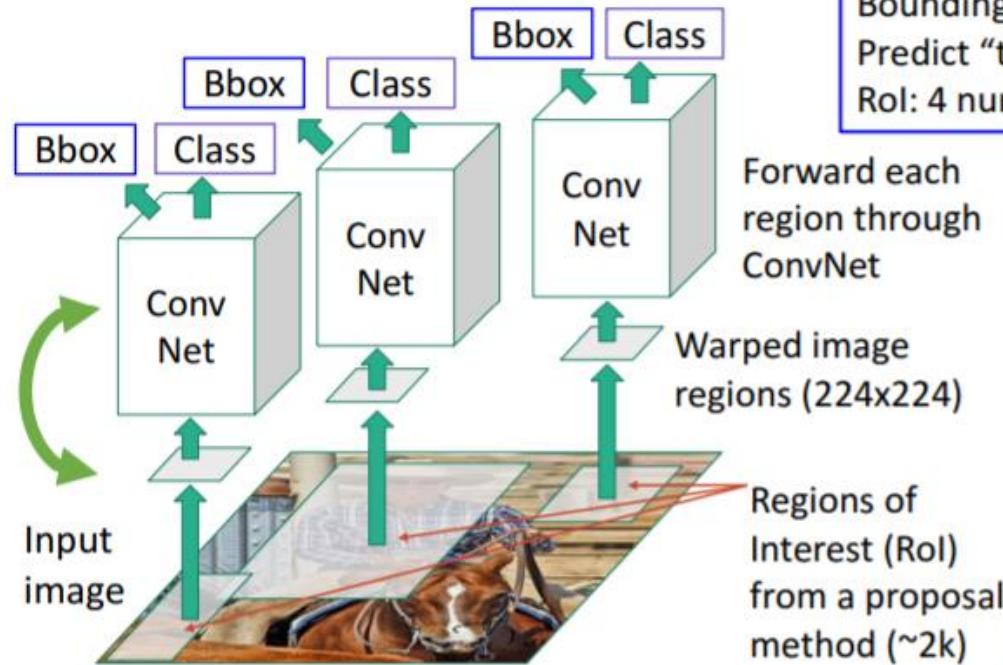
$$b_y = p_y + p_h t_y$$

$$b_w = p_w \exp(t_w)$$

$$b_h = p_h \exp(t_h)$$

# Fast R-CNN

R-CNN: Region-Based CNN



Classify each region

Bounding box regression:  
Predict “transform” to correct the  
RoI: 4 numbers ( $t_x, t_y, t_h, t_w$ )

Forward each  
region through  
ConvNet

Warped image  
regions (224x224)

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

**Problem:** Very slow!  
Need to do ~2k forward  
passes for each image!

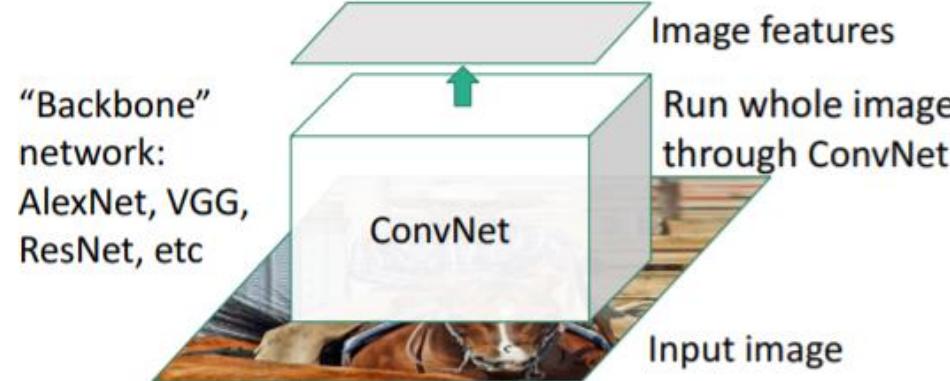
**Solution:** Run CNN  
\*before\* warping!

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.  
This work is licensed under a Creative Commons Attribution Non-Commercial-ShareAlike 4.0 International License.

The R-CNN has a problem. **The extracted 2000 region proposals should be put into CNN 2000 times as a forward pass. Very slow!**

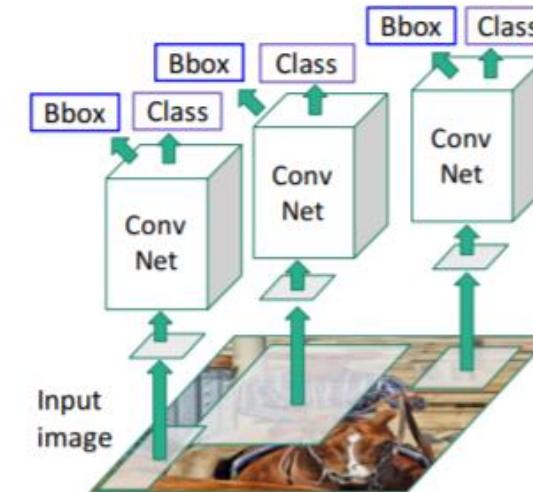
To solve this problem, they **changed the order of CNN and warping**.

# Fast R-CNN



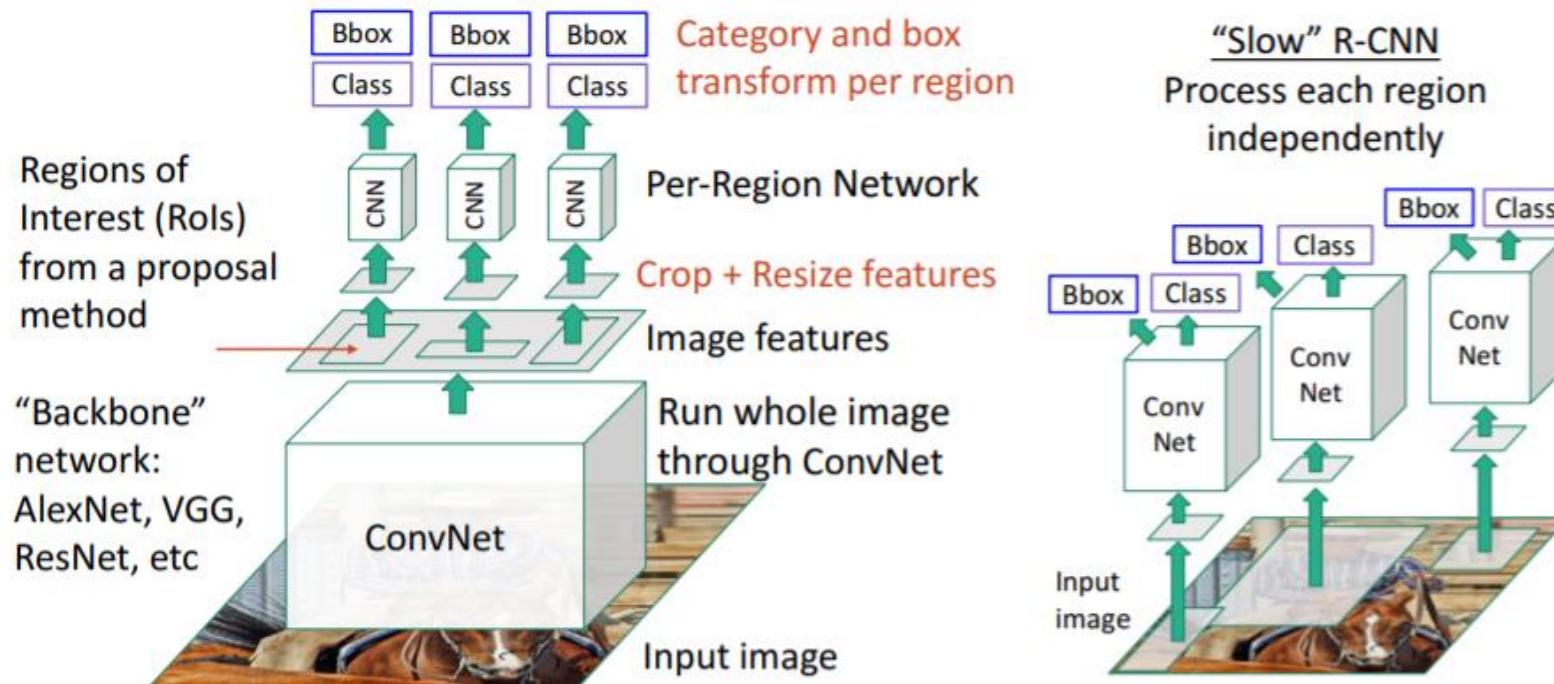
Run whole image through ConvNet

“Slow” R-CNN  
Process each region independently



Not as with original R-CNN, region proposals are not put into each CNN, but **the entire image is put into a single CNN with only Conv-layer without fc-layer**. As a result, a feature map can be obtained.

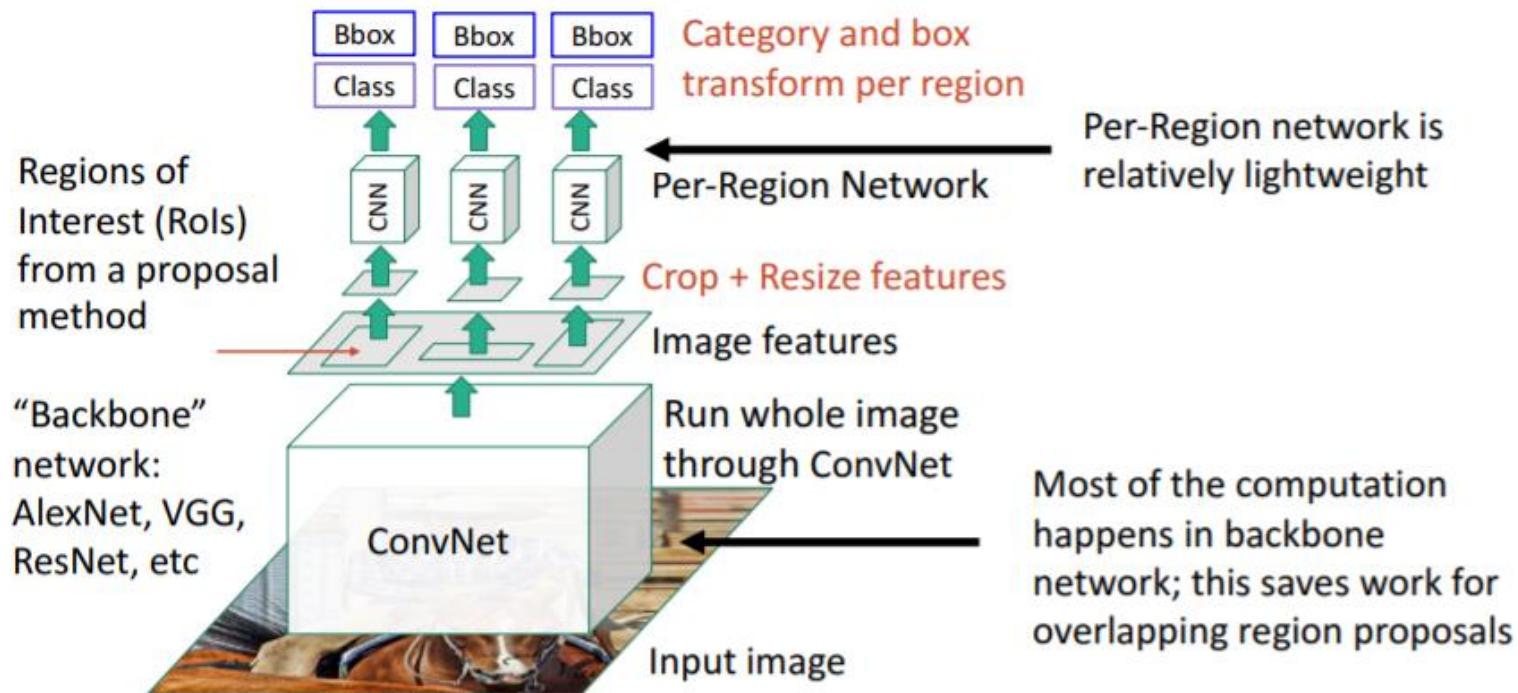
# Fast R-CNN



**On this feature map, region proposals are extracted in the same way as selective search.**

The output is calculated by cropping and resizing these feature region proposals and passing each through a small CNN.

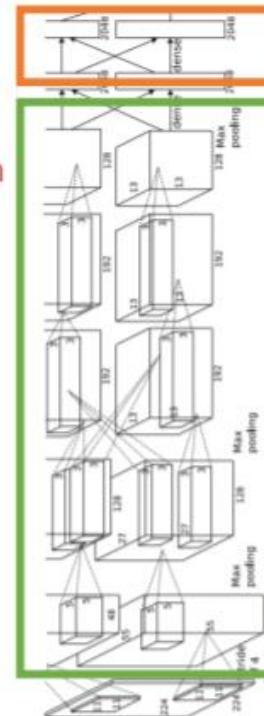
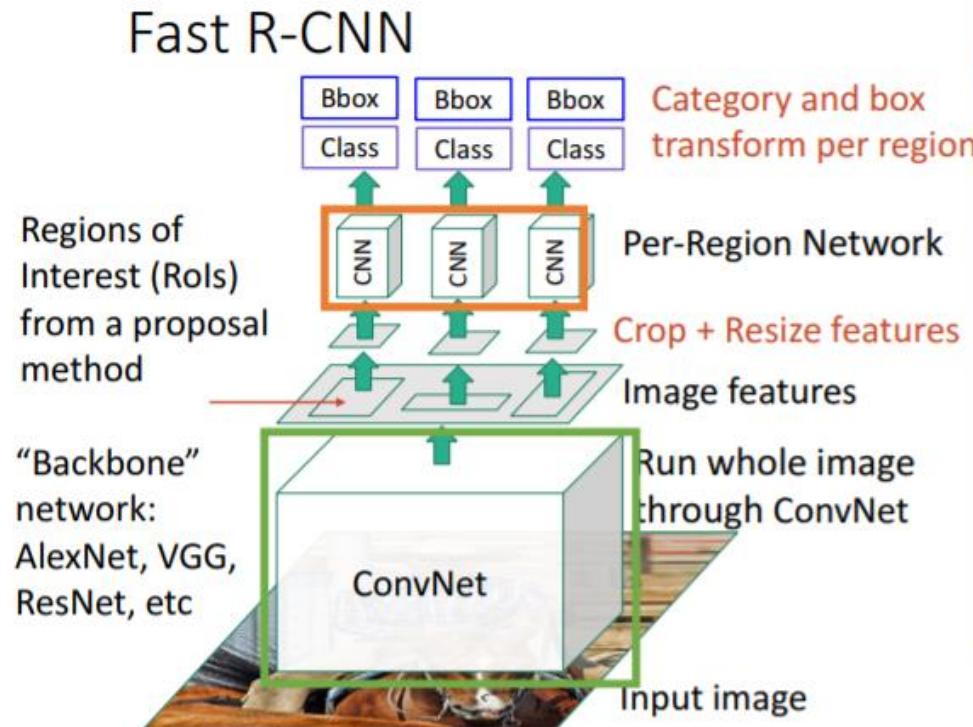
# Fast R-CNN



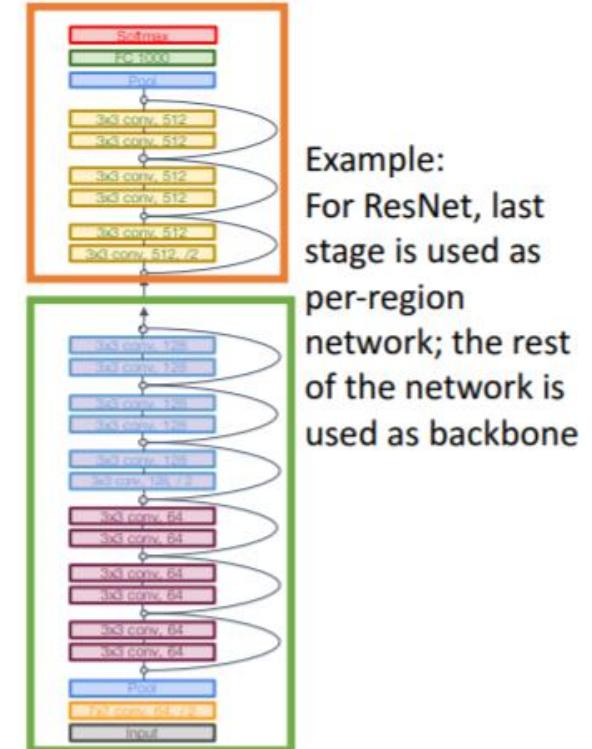
The reason **why this method is fast** is that **most of the calculations take place on the Backbone network** below, which **ends with a single calculation** without having to be repeated for overlapping region proposals.

And the **CNN above**, per-region network, is **relatively very light and small**.

# Fast R-CNN: Different Backbone



**Example:**  
When using AlexNet for detection, five conv layers are used for backbone and two FC layers are used for per-region network



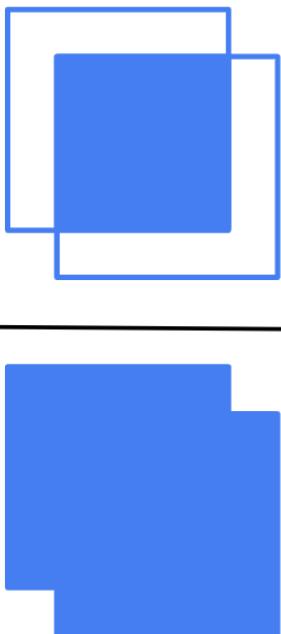
# Fast R-CNN: Test Time

---

## Non-Max Suppression (NMS)

$$IoU = \frac{\text{Area of Intersection}}{\text{Intersection over Union} \cdot \text{Area of Union}}$$

The diagram illustrates the calculation of Intersection over Union (IoU). It shows two overlapping blue rectangles. The top rectangle is labeled "Area of Intersection". The bottom rectangle is labeled "Area of Union".



# Fast R-CNN: Test Time

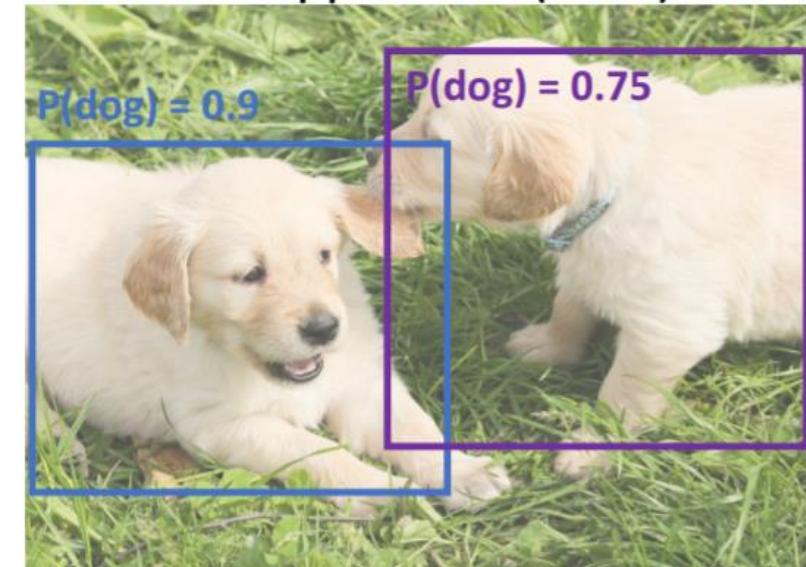
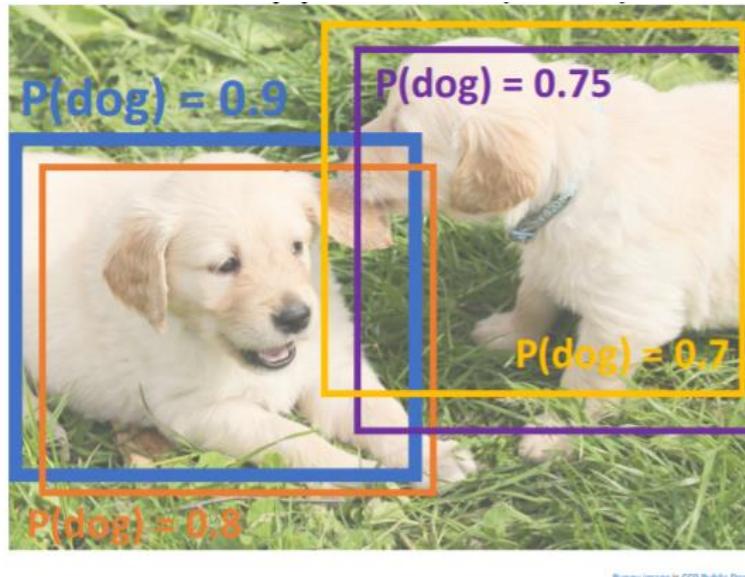
## Non-Max Suppression (NMS)

**Problem:** Object detectors often output many overlapping detections:

**Solution:** Post-process raw detections using **Non-Max Suppression (NMS)**

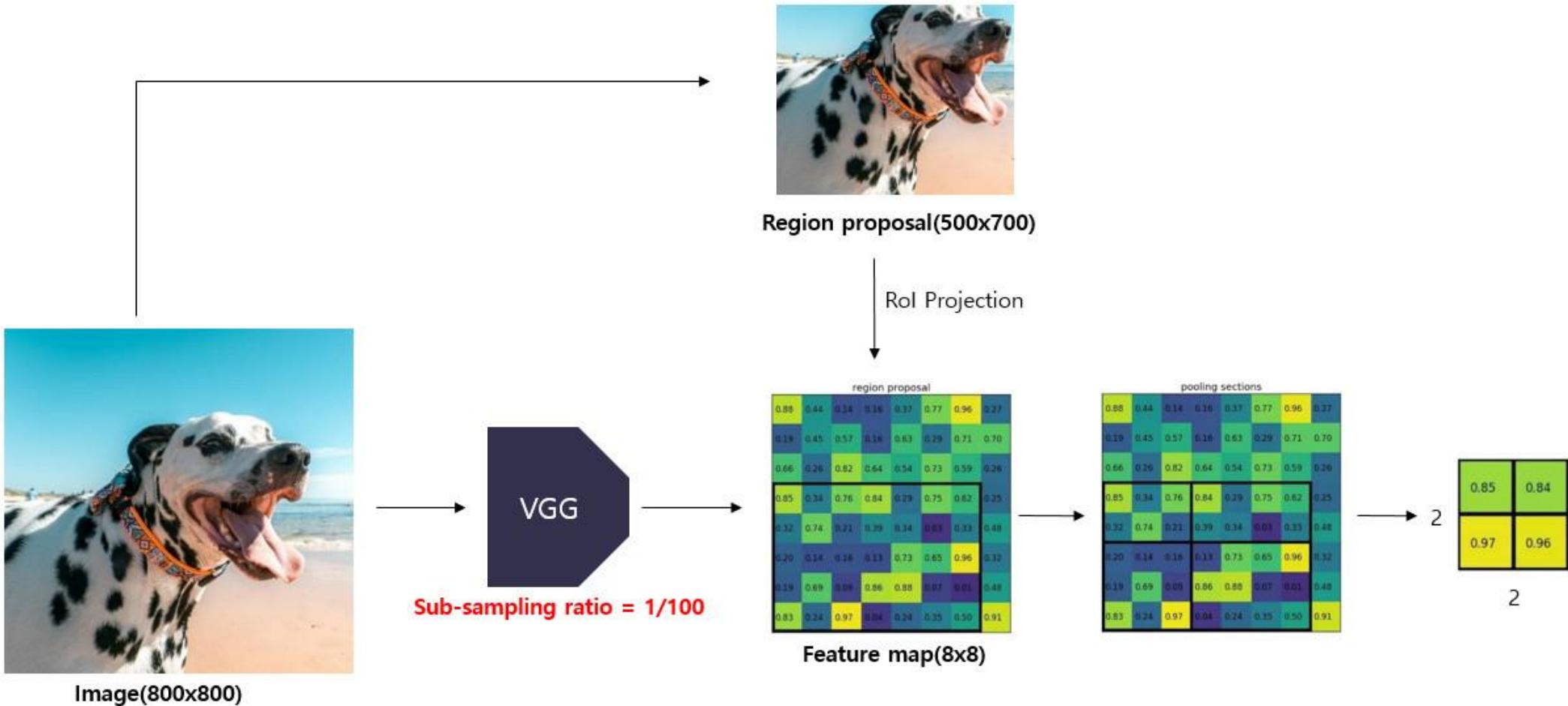
1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with  $\text{IoU} > \text{threshold}$  (e.g. 0.7)
3. If any boxes remain, GOTO 1

$$\begin{aligned}\text{IoU}(\text{blue}, \text{orange}) &= 0.78 \\ \text{IoU}(\text{blue}, \text{purple}) &= 0.05 \\ \text{IoU}(\text{blue}, \text{yellow}) &= 0.07\end{aligned}$$

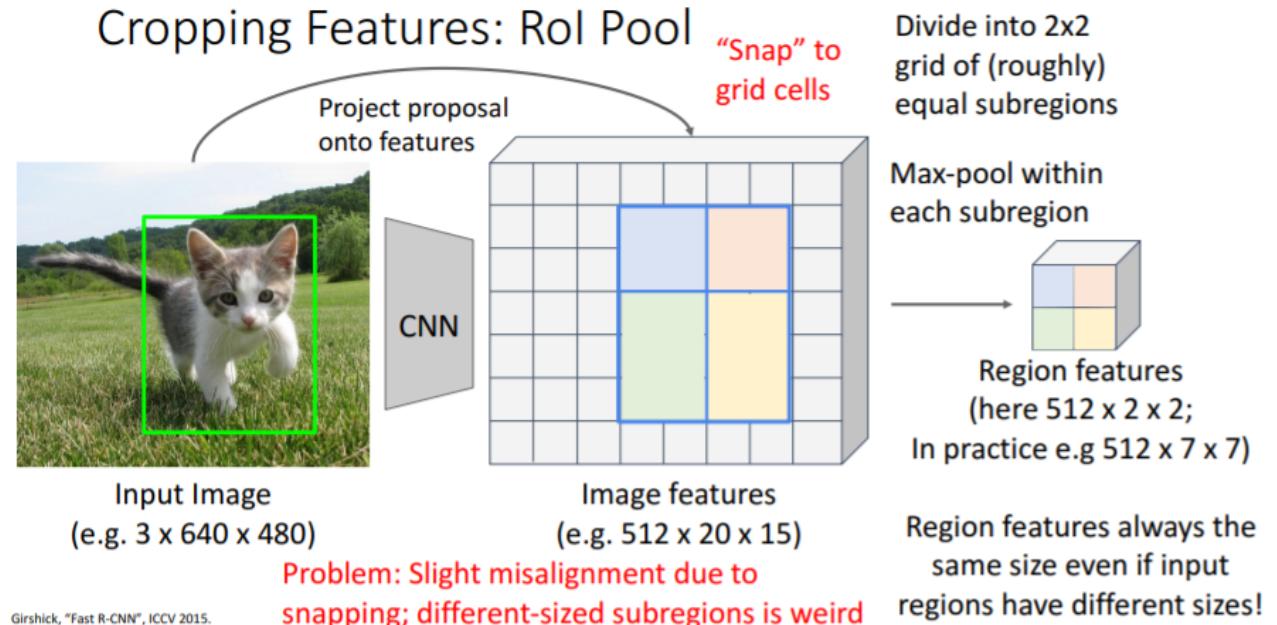
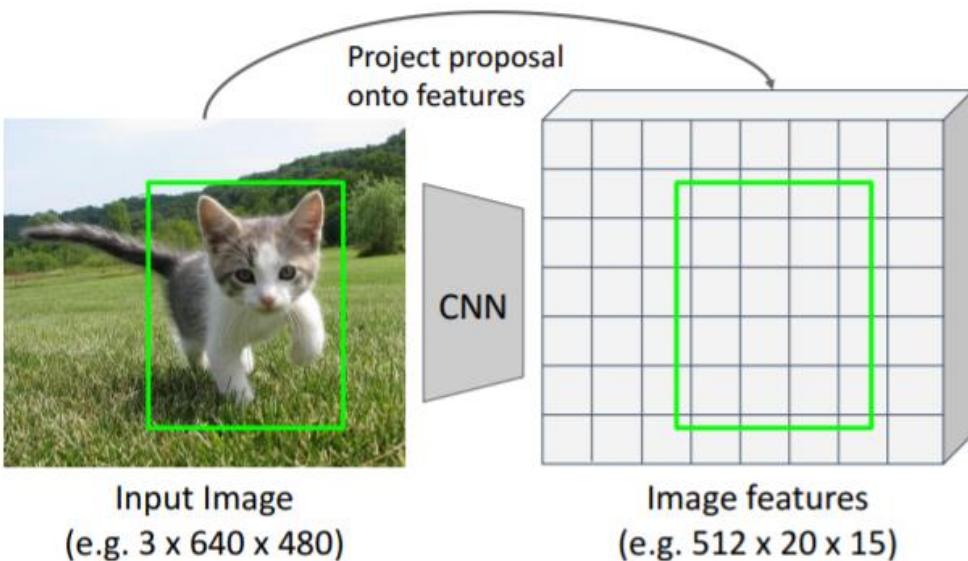


1. We get the image classification score for each bounding box. **Select next highest-scoring box**
2. For every other boxes, if  $\text{IoU} > \text{threshold} \rightarrow$  Eliminate
3. If any boxes remain, GOTO 1.

# Fast R-CNN: RoI Pooling



# Fast R-CNN: ROI Pooling

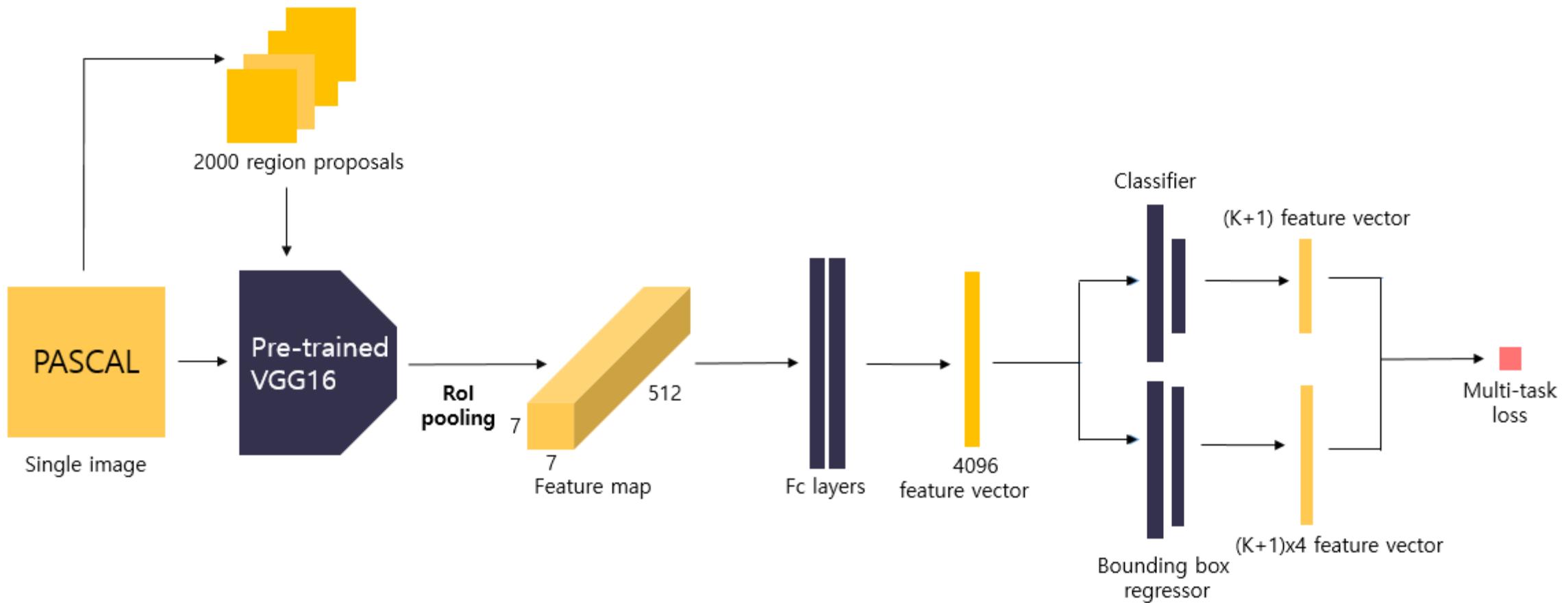


## Problems

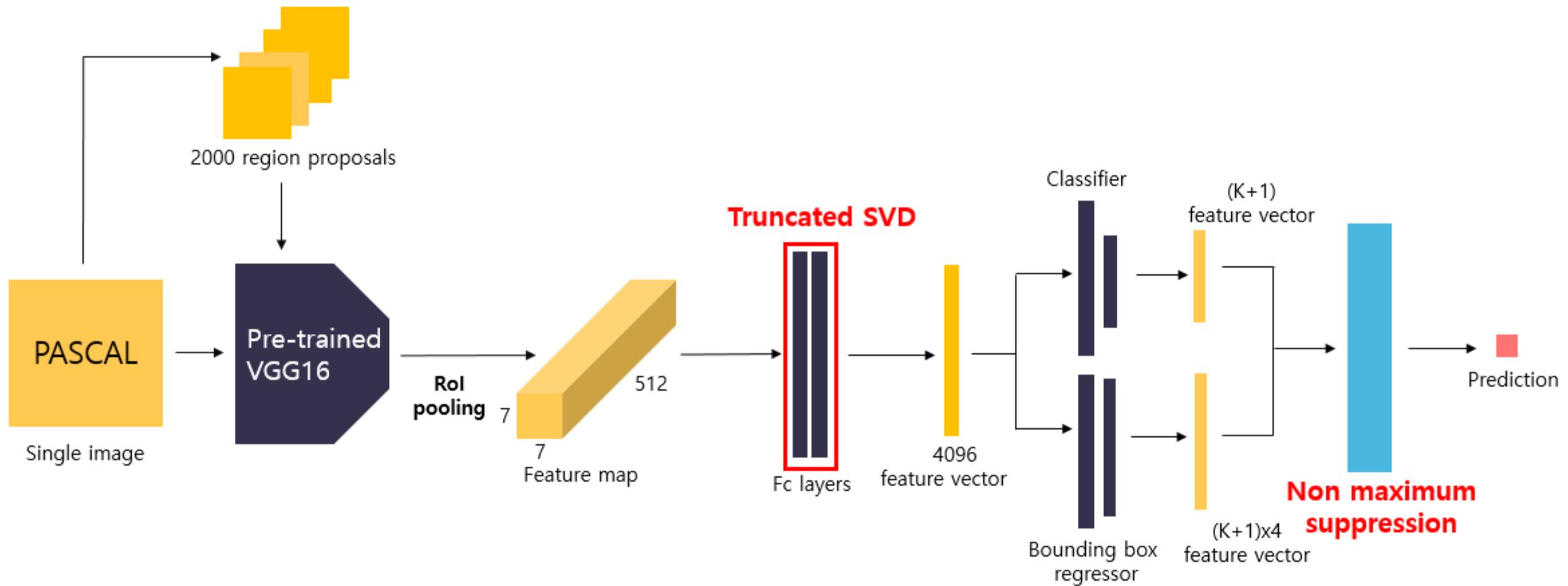
- Snapping cause misalignment
- Sub-regions are different-sized
- Cannot backpropagate to the original bounding box coordinates (snapped)

# Fast R-CNN

---

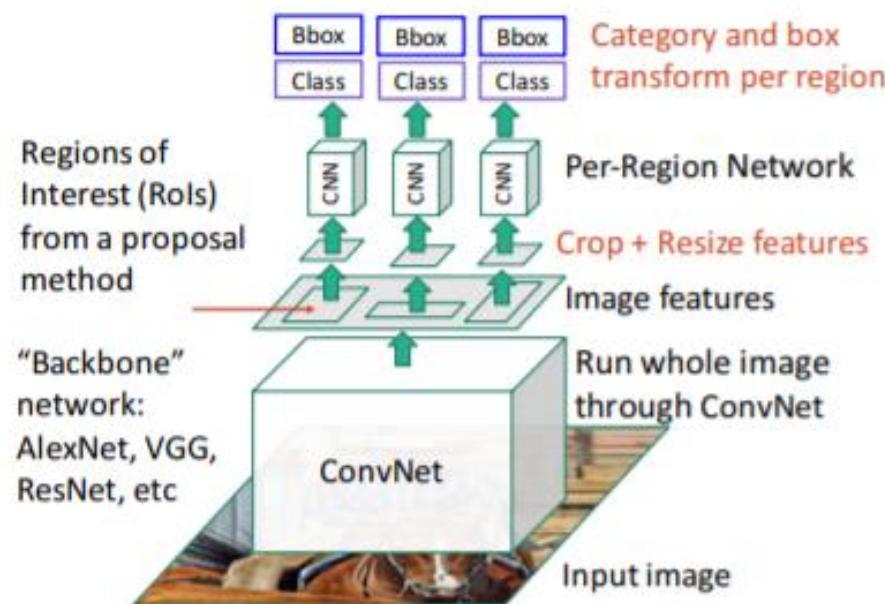


# Fast R-CNN

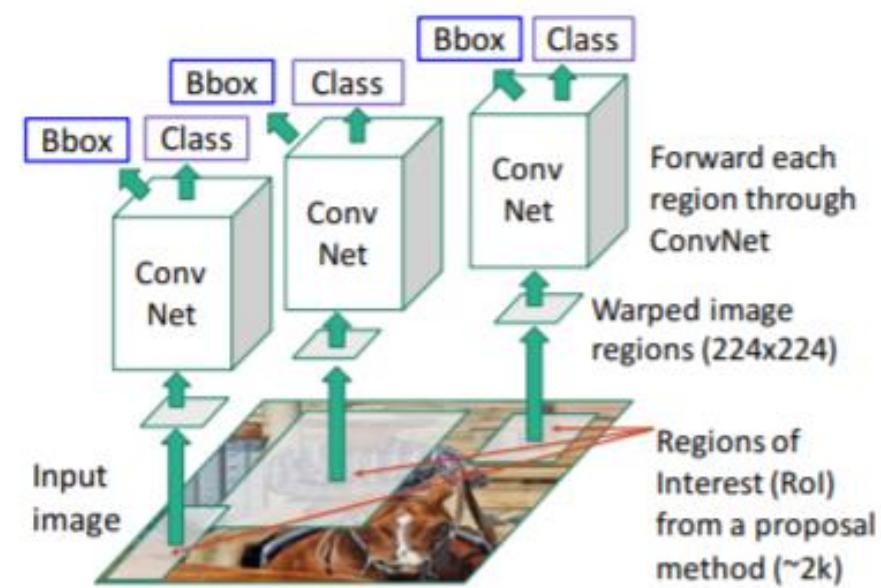


# R-CNN vs Fast R-CNN

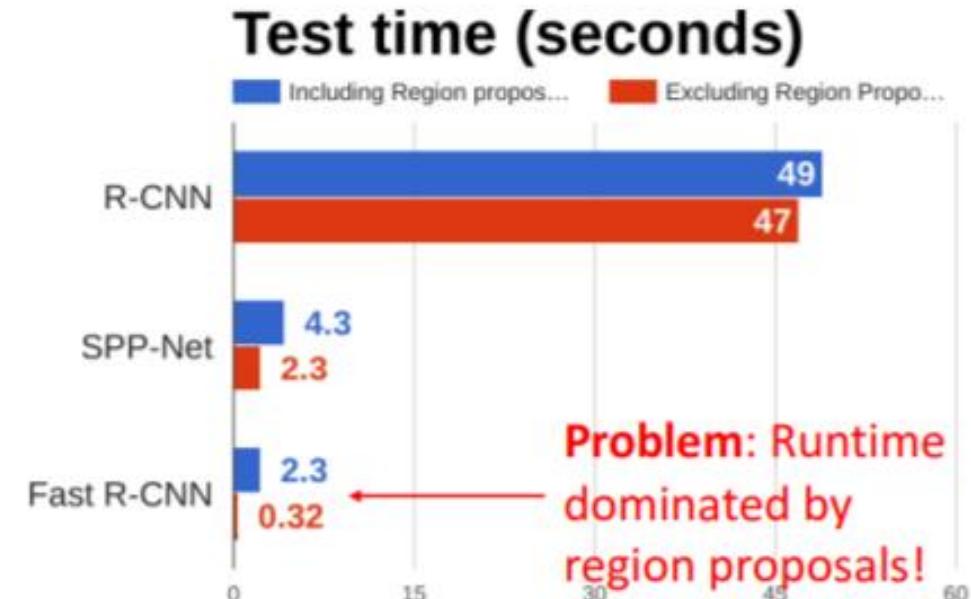
**Fast R-CNN:** Apply differentiable cropping to shared image features



**“Slow” R-CNN:** Apply differentiable cropping to shared image features



# R-CNN 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

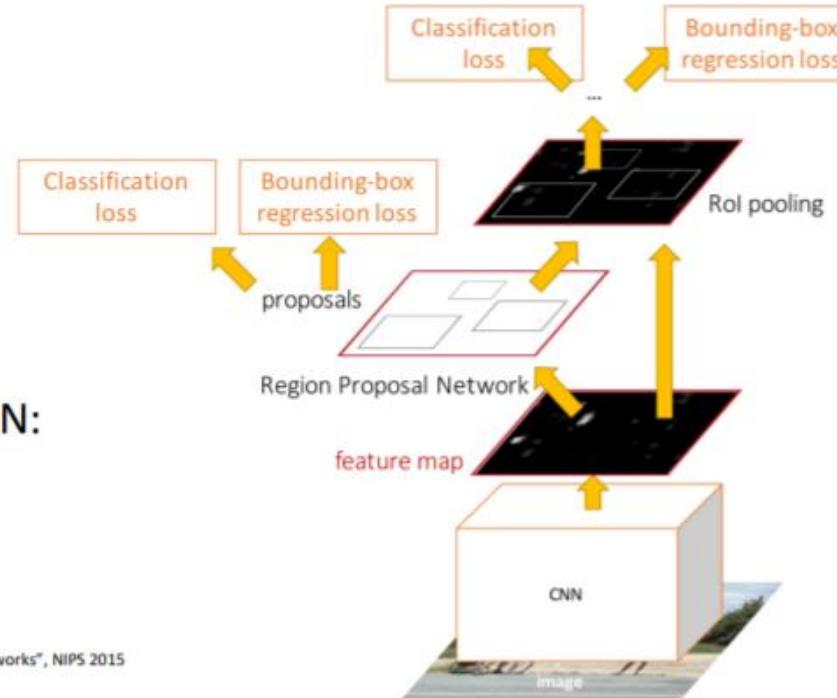
**Recall:** Region proposals computed by heuristic "Selective Search" algorithm on CPU -- let's learn them with a CNN instead!

# Faster R-CNN

Train CNN to predict region proposals on CNN without calculating region proposals by selective search.

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

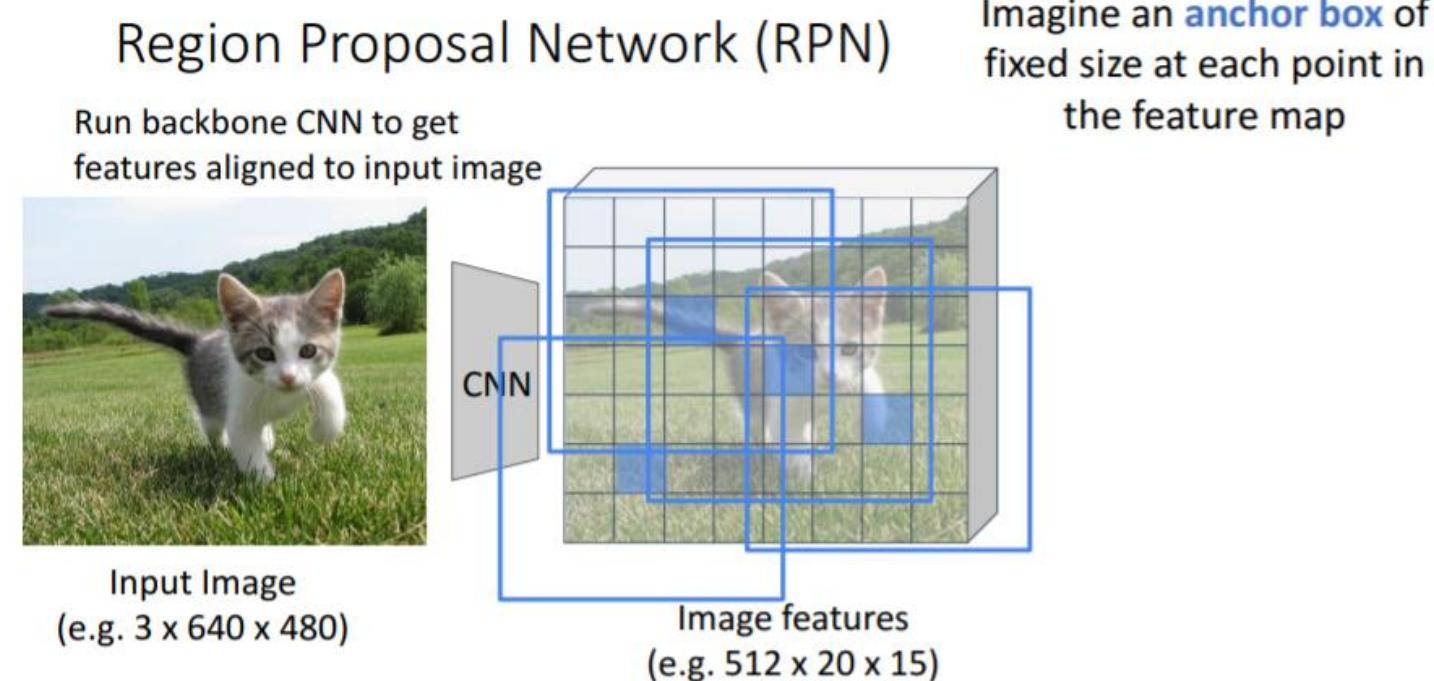
Otherwise same as Fast R-CNN:  
Crop features for each proposal, classify each one



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

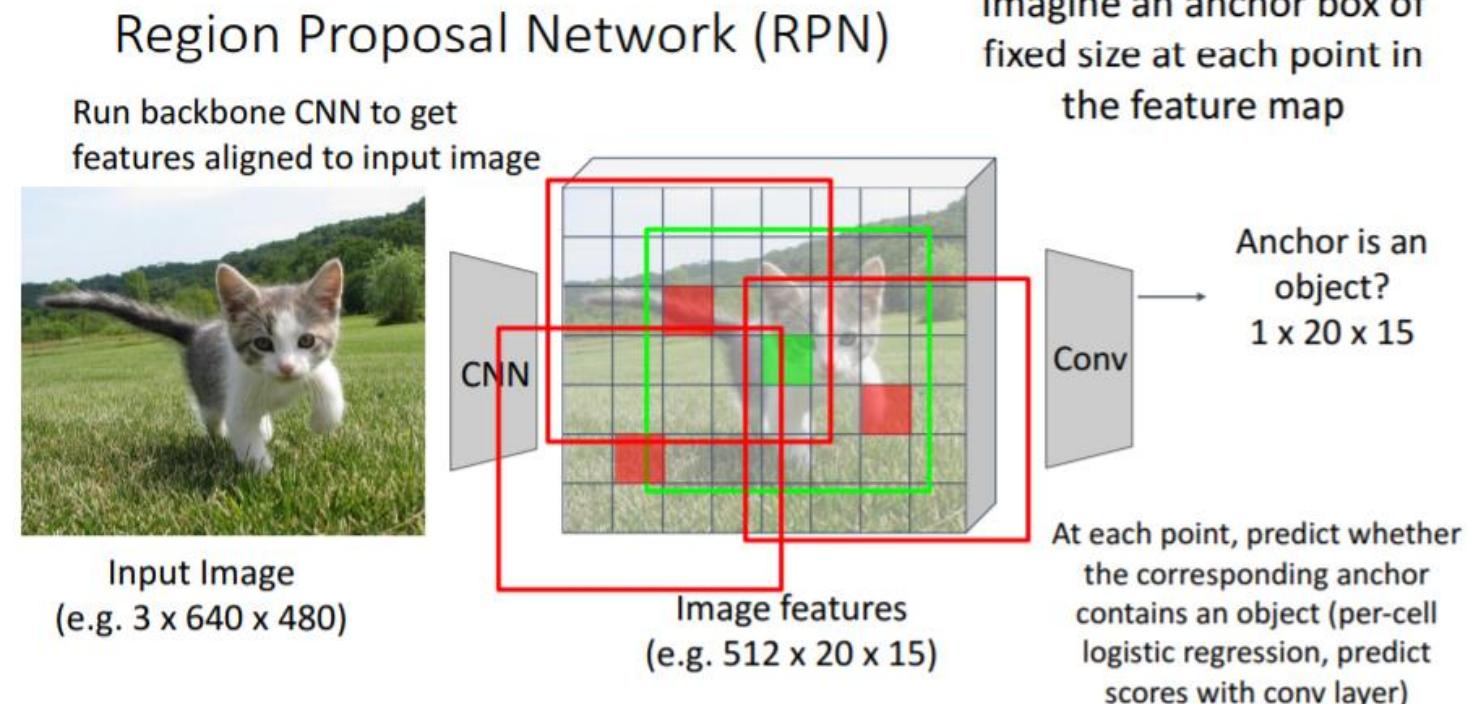
Output feature map passes through the RPN to predict the region proposals, and with the extracted region proposals, all process the rest of the process in the same way as in Fast R-CNN.

# Region Proposal Network (RPN)



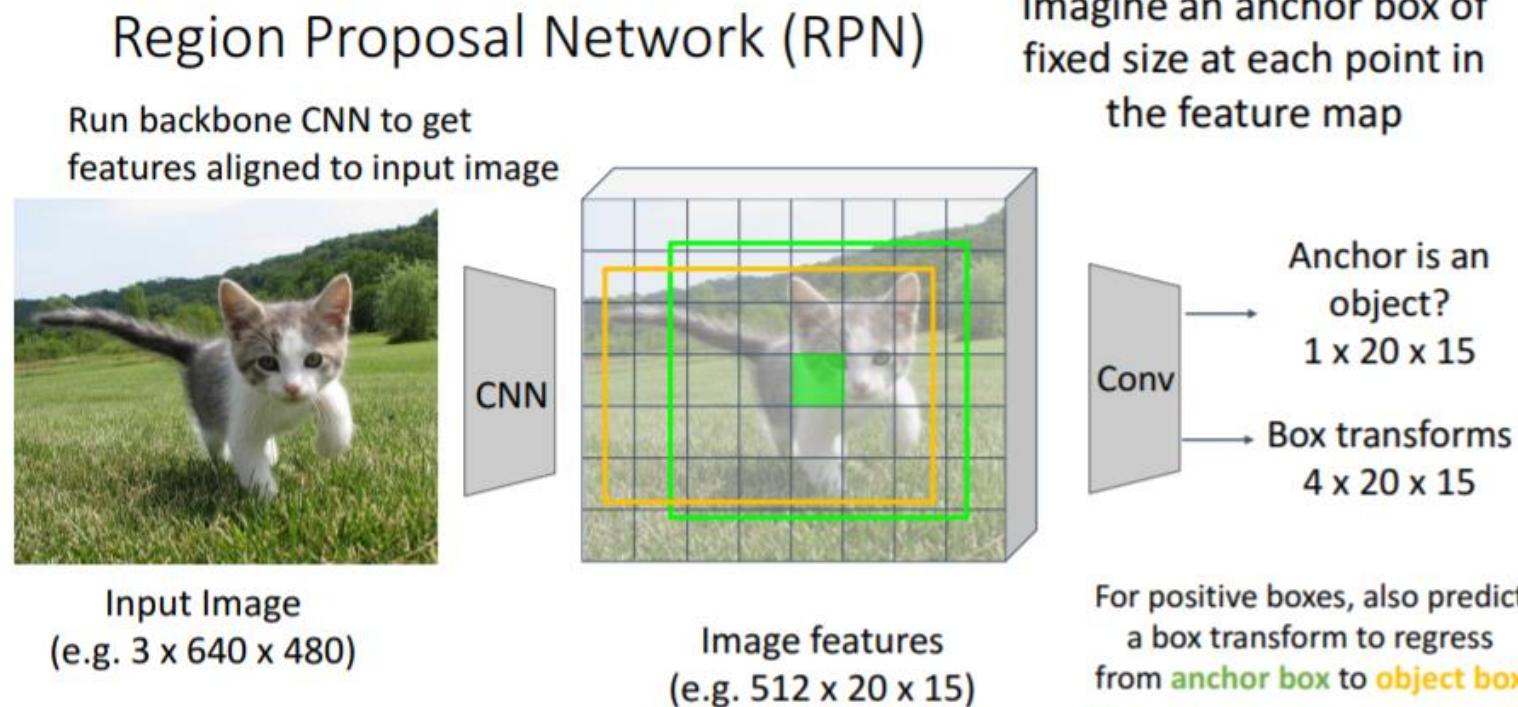
Fixed size of anchor box at each point in the feature map

# Region Proposal Network (RPN)



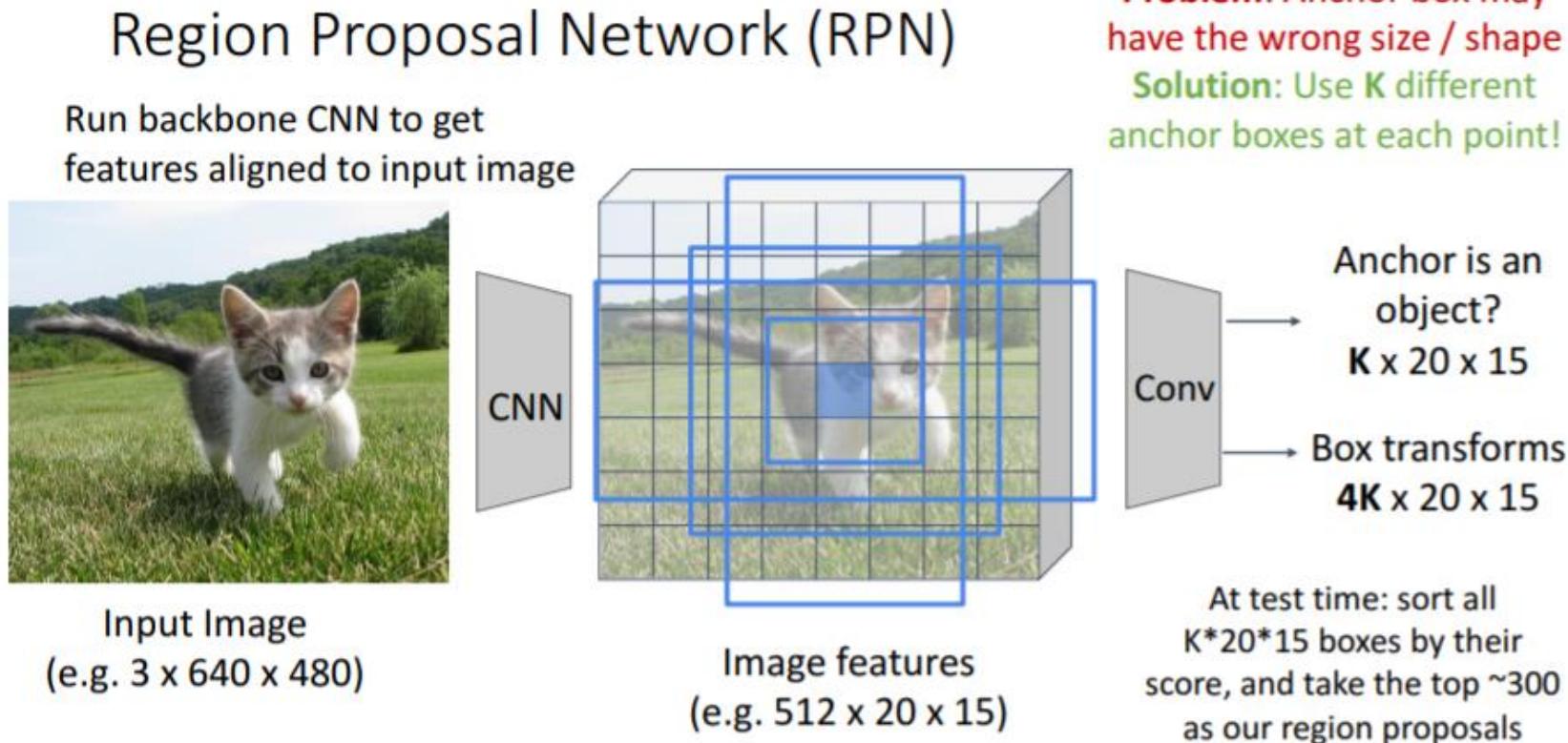
In each anchor box, **binary classify whether or not these anchor boxes contain objects through CNN.**

# Region Proposal Network (RPN)



The **box transform** is also predicted for **positive boxes**, and the **anchor box** is regressed to the **proposal box**.

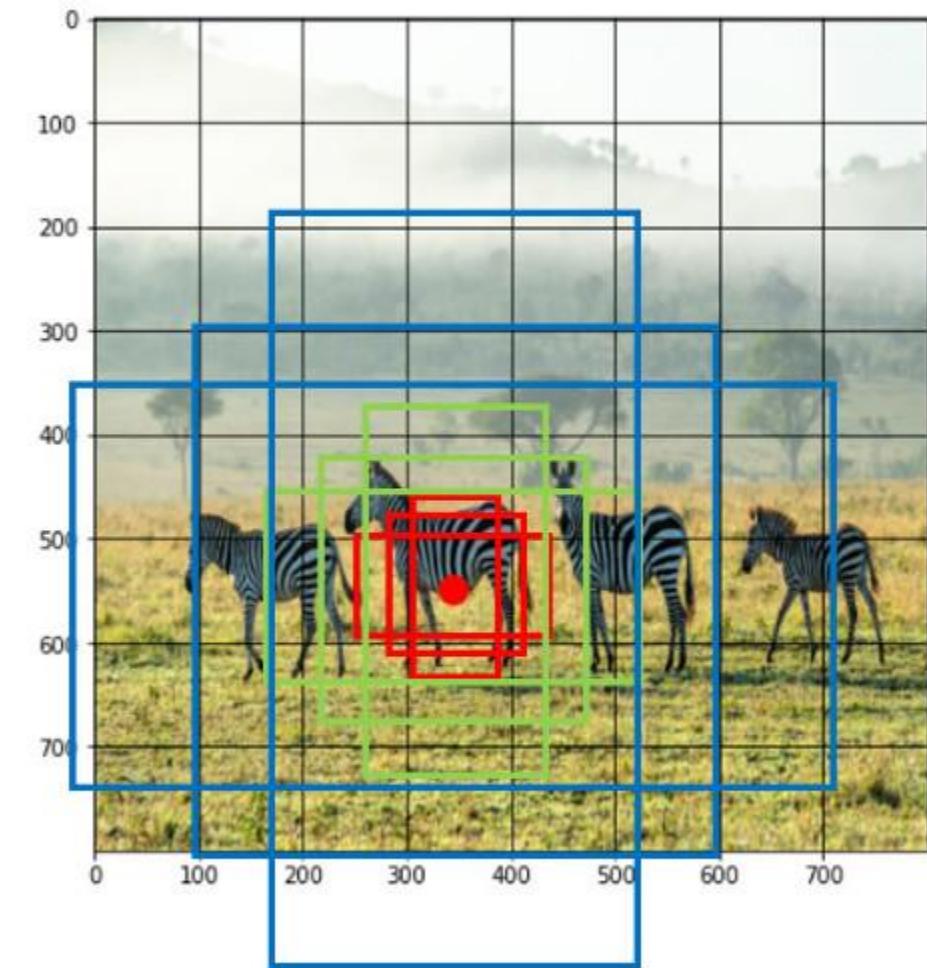
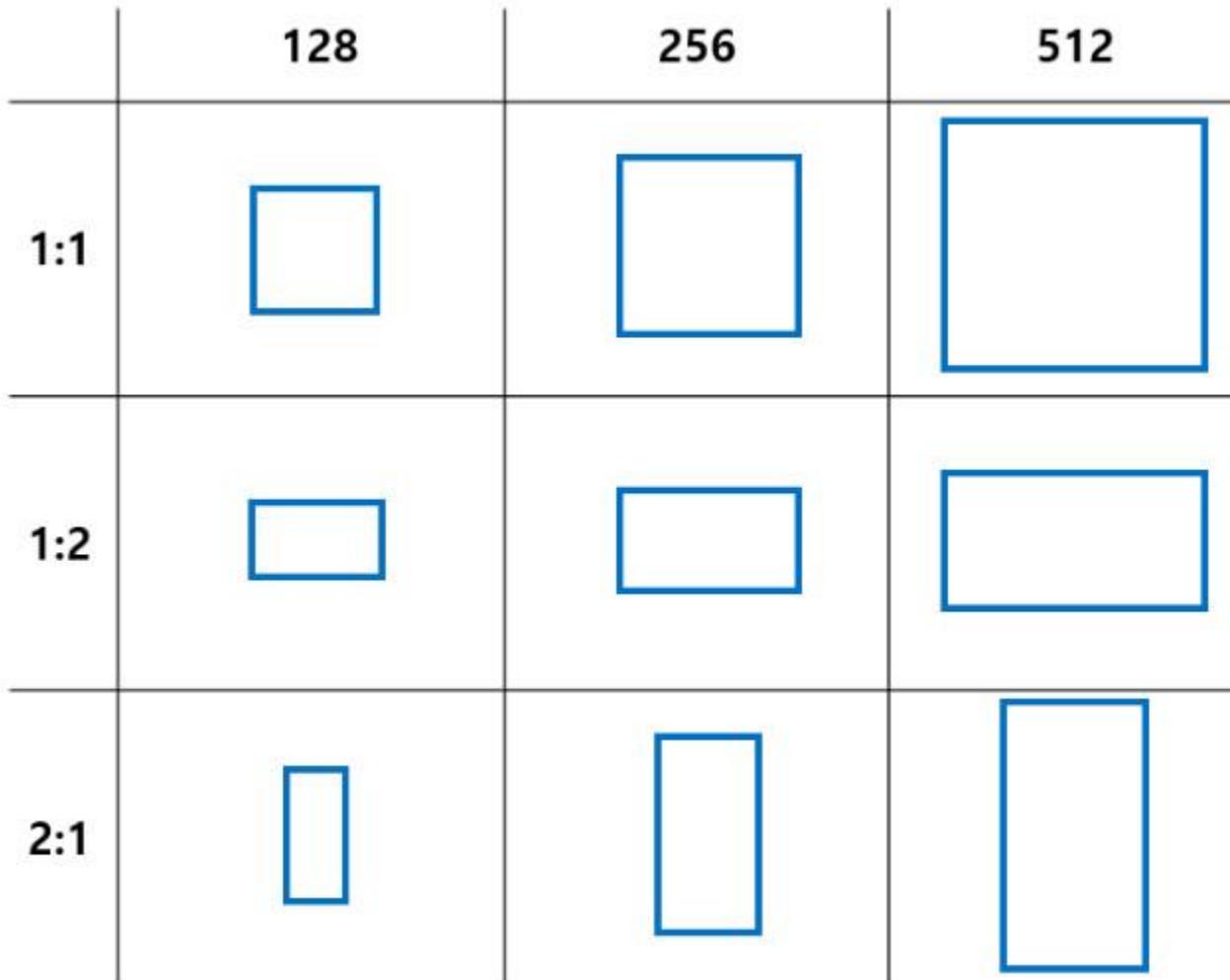
# Region Proposal Network (RPN)



Using only one fixed size anchor box is **not enough to fit well on all kinds of objects.**

Therefore, instead, **K different anchor boxes** (different sizes, different aspect ratios) are **used at each feature map point.**

# Region Proposal Network (RPN)

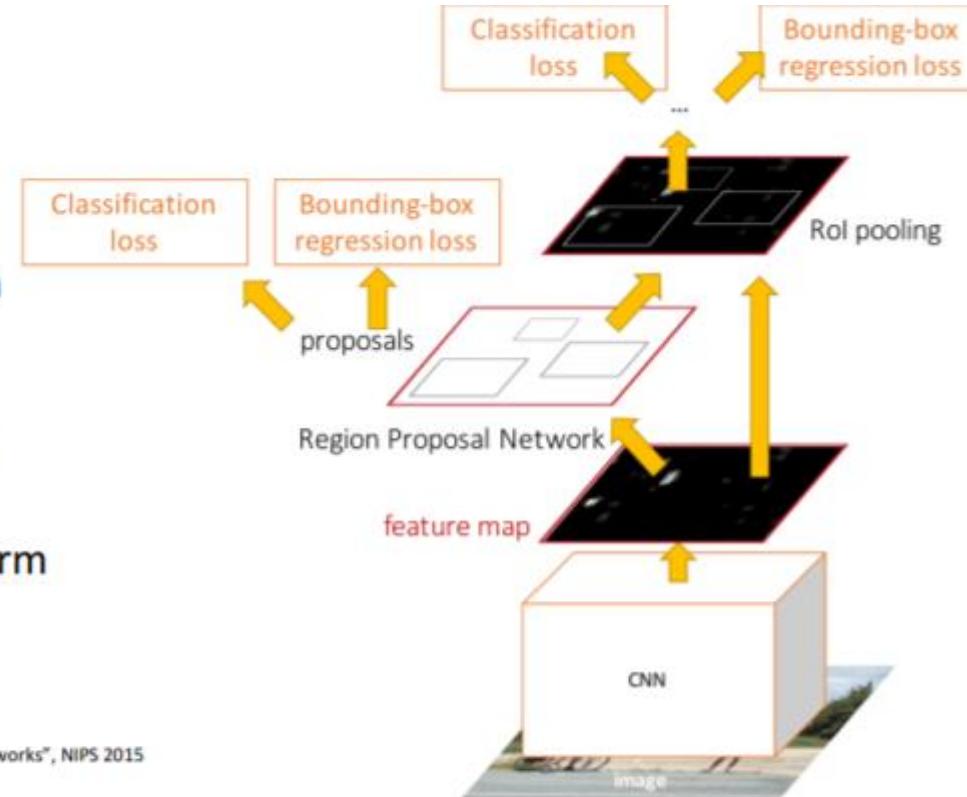


# Faster R-CNN: 4 Losses

---

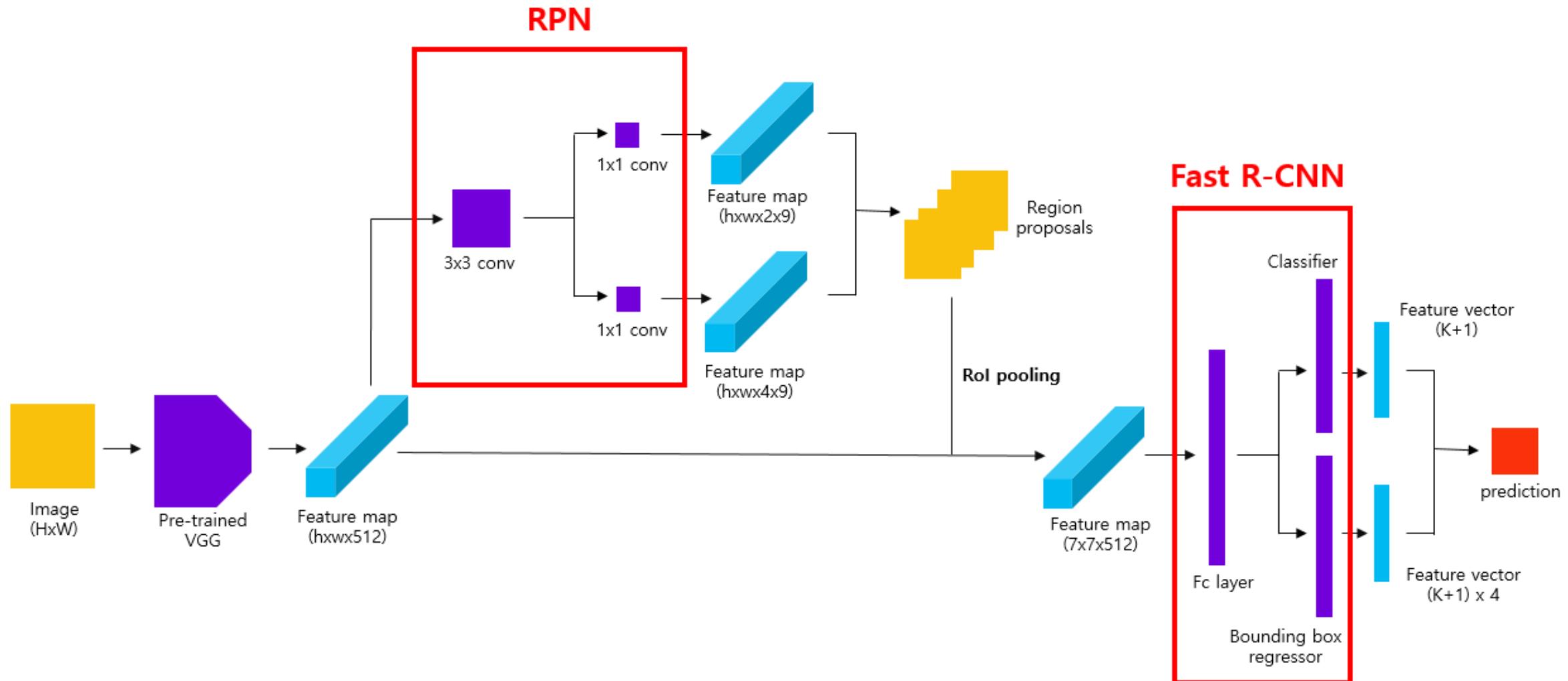
Jointly train with 4 losses:

1. **RPN classification**: anchor box is object / not an object
2. **RPN regression**: predict transform from anchor box to proposal box
3. **Object classification**: classify proposals as background / object class
4. **Object regression**: predict transform from proposal box to object box



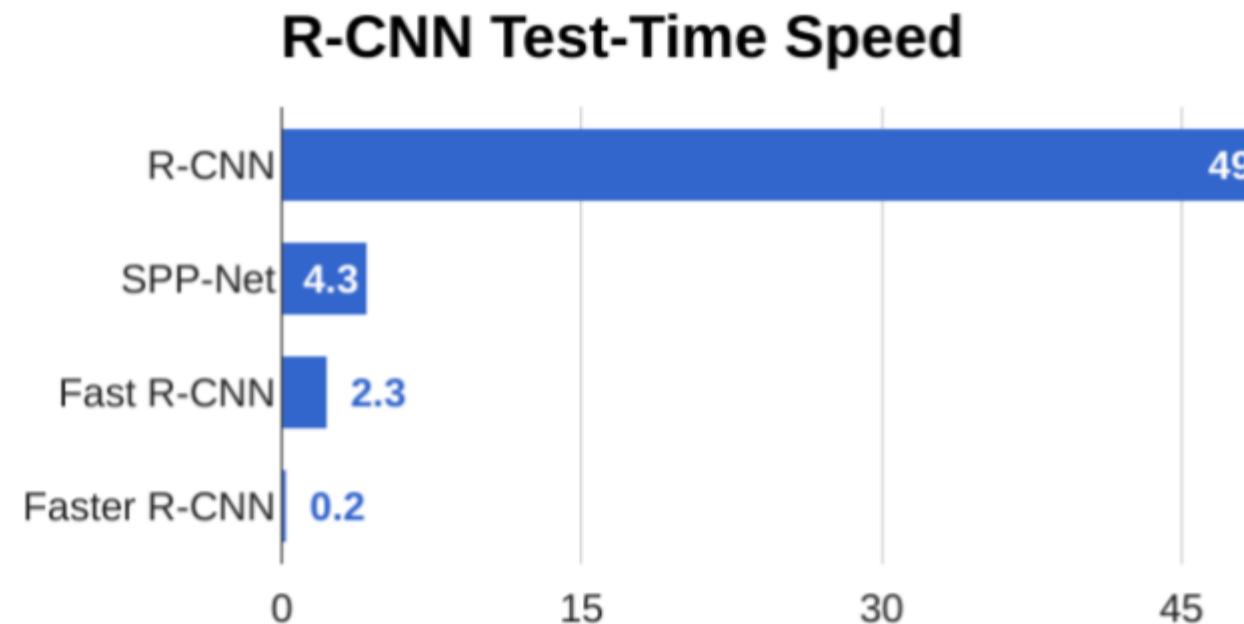
Ren et al., "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015  
Figure copyright 2015, Ross Girshick; reproduced with permission

# Faster R-CNN: 4 Losses



# Faster R-CNN

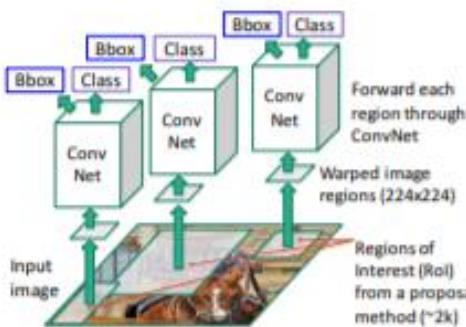
---



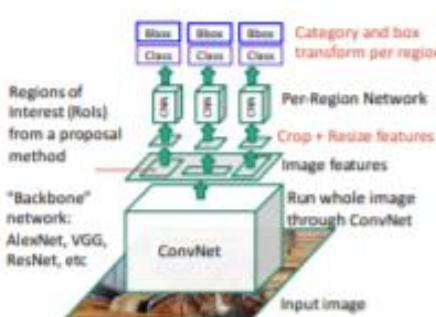
# Summary

---

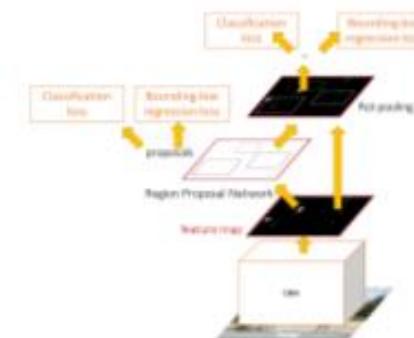
**“Slow” R-CNN:** Run CNN independently for each region



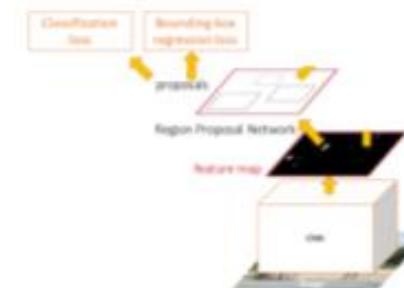
**Fast R-CNN:** Apply differentiable cropping to shared image features



**Faster R-CNN:** Compute proposals with CNN



**Single-Stage:** Fully convolutional detector



# Semantic Segmentation

Classification



CAT

No spatial extent

Semantic  
Segmentation



GRASS, CAT, TREE,  
SKY

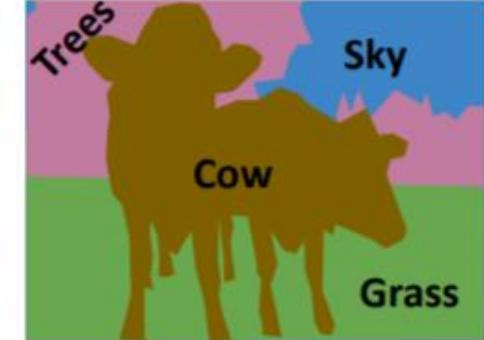
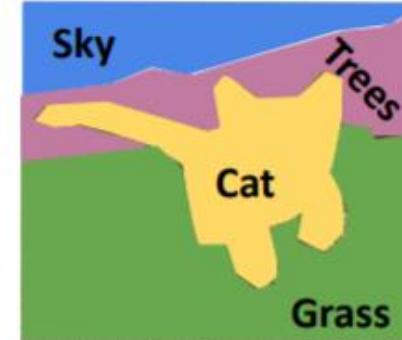
No objects, just pixels

Label each pixel in the image  
with a category label

Don't differentiate instances,  
only care about pixels

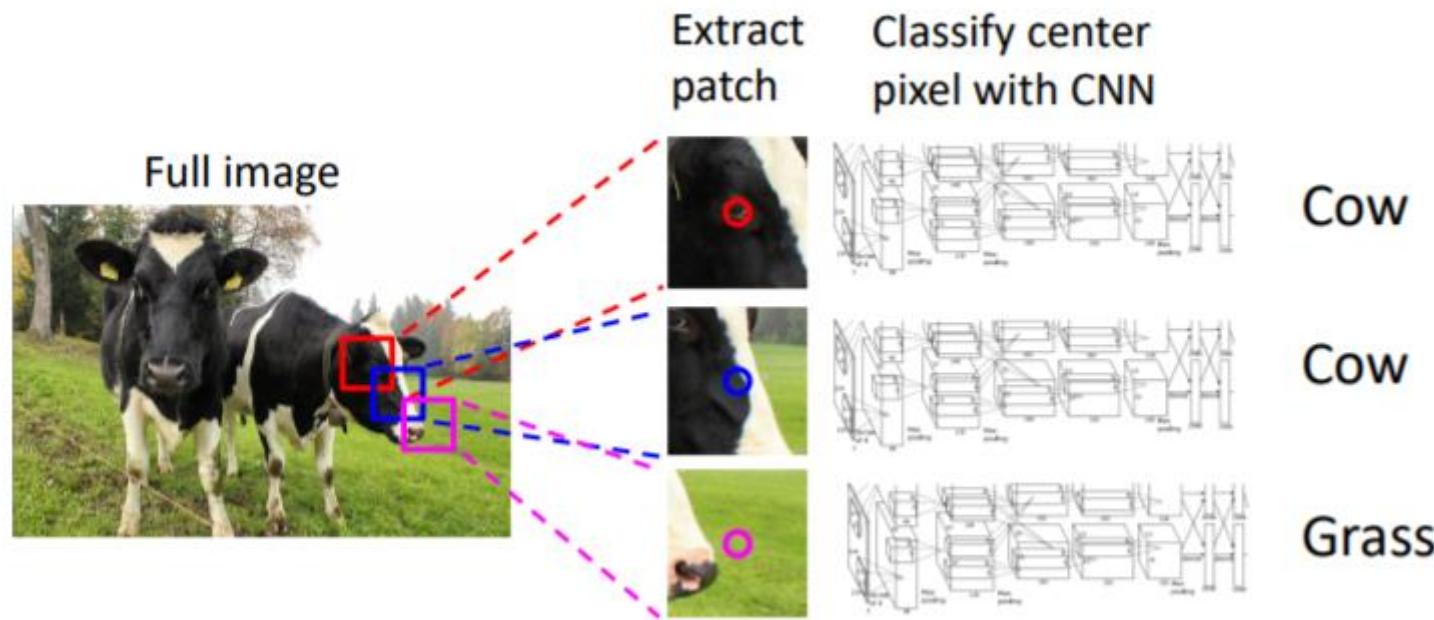


This image is CC0 public domain



- Label all the pixels in the image with category labels. It doesn't distinguish between instances, it just focuses on the pixel.
- For example, in the figure on the right, two cows are not detected separately.

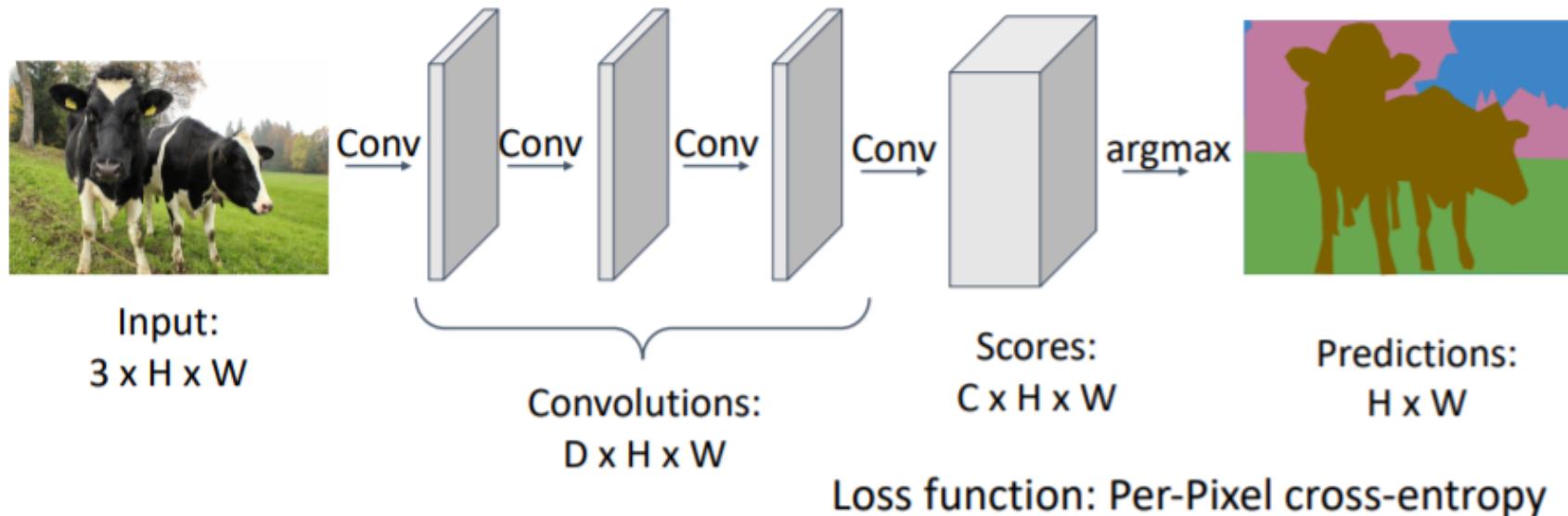
# Semantic Segmentation Idea: Sliding Window



- For each pixel of the image, a small patch including the periphery is extracted. We then put this patch into CNN to predict the category label. Repeat this.
- **Problem:** Very inefficient! Not reusing shared features between overlapping patches
- Therefore, this method is rarely used.

# Semantic Segmentation Idea: Fully Convolutional Network

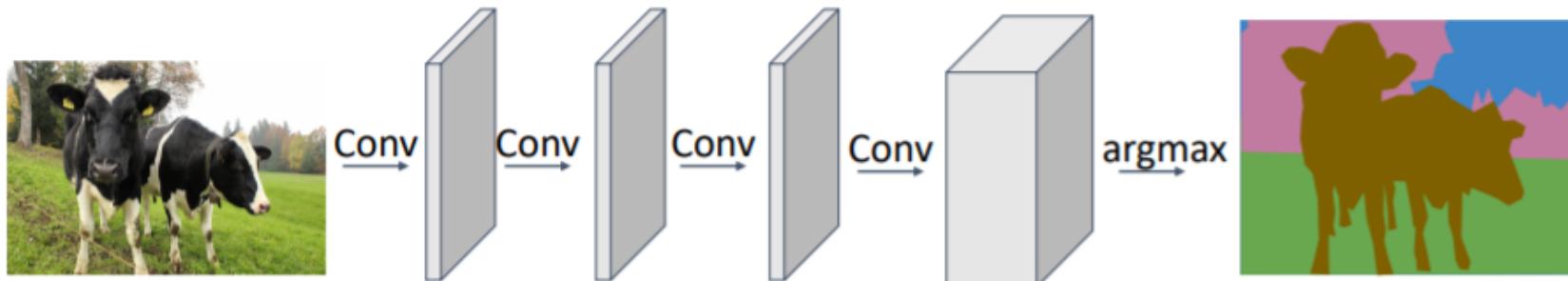
Design a network as a bunch of convolutional layers to make predictions for pixels all at once!



- It uses a fully convolutional network where **only conv layers are stacked** without pooling layers.
- When an input image of a certain size is inserted, the output that passes through the network is a divided image of the same size as the input.  $C$  is the number of category labels.
- And the last score layer has a category label score for each pixel. In each pixel, softmax is used to determine which label the pixel is.

# Semantic Segmentation Idea: Fully Convolutional Network

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



Input:  
 $3 \times H \times W$

**Problem #1:** Effective receptive field size is linear in number of conv layers: With  $L$   $3 \times 3$  conv layers, receptive field is  $1+2L$

**Problem #2:** Convolution on high res images is expensive!  
Recall ResNet stem aggressively downsamples

- **Problem**

- The size of the receptive field increases linearly to  $1+2L$  because it continues to stack using the conv layer of  $3 \times 3$ , pad 1, and stride 1. Therefore, the larger the input image, the deeper the conv layer is needed.
- Conv is very expensive in high-resolution images.

- **Downsampling!!**

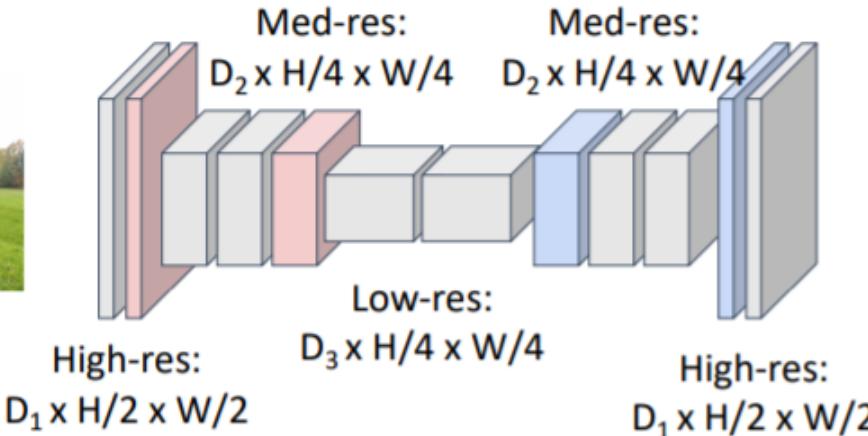
# Semantic Segmentation Idea: Fully Convolutional Network

**Downsampling:**  
Pooling, strided  
convolution



Input:  
 $3 \times H \times W$

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



**Upsampling:**  
???

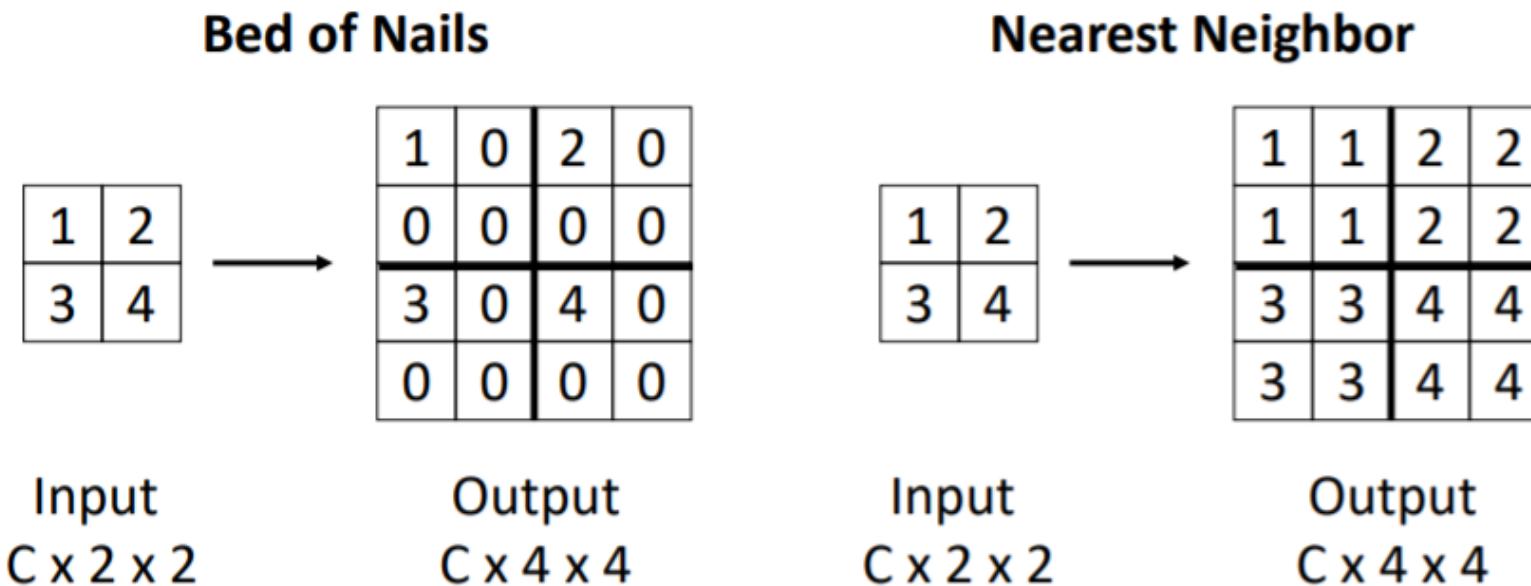


Predictions:  
 $H \times W$

- If **downsampling** is performed using **pooling and stride**, a larger receive field can be obtained more simply and the calculation process can be reduced.
- Then, how do you do **upsampling**?

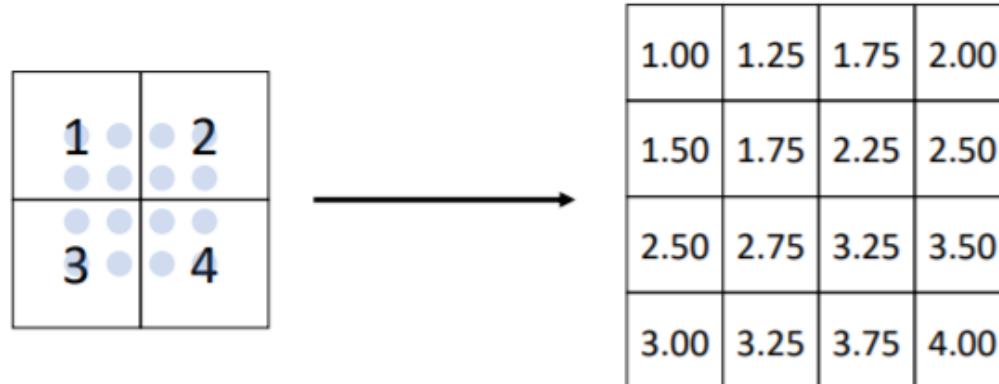
# In-Network Upsampling: “Unpooling”

---



- Since downsampling is mainly called **pooling**, upsampling is called **unpooling**.
- The **bed of nails** method is rarely used for **aliasing problems**.

# In-Network Upsampling: Bilinear Interpolation



$$f_{x,y} = \sum_{i,j} f_{i,j} \max(0, 1 - |x - i|) \max(0, 1 - |y - j|) \quad i \in \{\lfloor x \rfloor - 1, \dots, \lceil x \rceil + 1\}$$

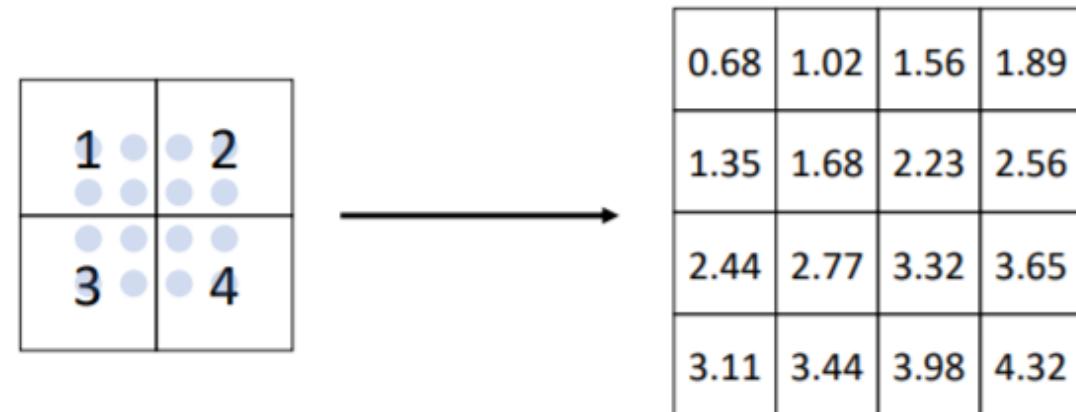
Use two closest neighbors in x and y  
to construct linear approximations

$$j \in \{\lfloor y \rfloor - 1, \dots, \lceil y \rceil + 1\}$$

- It also uses **bilinear interpolation** to do upsampling. This enables **smoother upsampling**.

# In-Network Upsampling: Bicubic Interpolation

---



Use **three** closest neighbors in x and y to  
construct **cubic** approximations  
(This is how we normally resize images!)

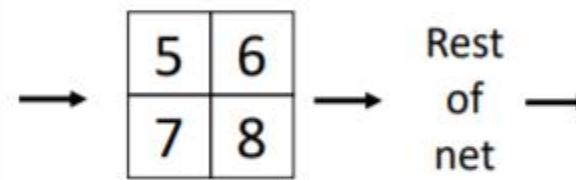
- Almost everyone uses this method.

<https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=dudcos101&logNo=221421989207>

# In-Network Upsampling: “Max Unpooling”

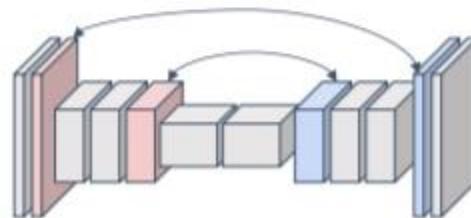
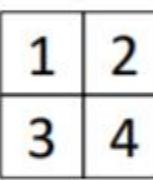
**Max Pooling:** Remember which position had the max

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



**Max Unpooling:** Place into remembered positions

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

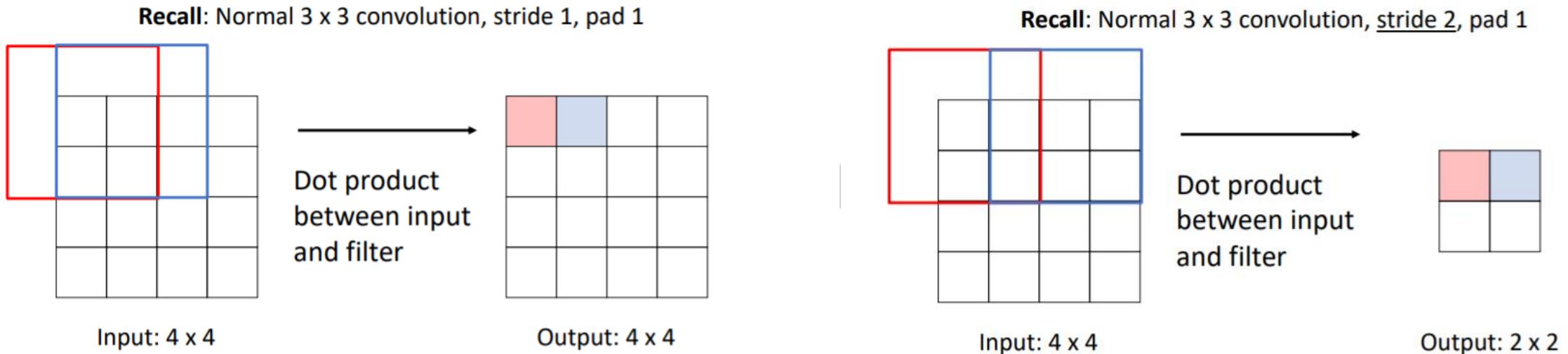


Pair each downsampling layer with an upsampling layer

Noh et al. "Learning Deconvolution Network for Semantic Segmentation". ICCV 2015

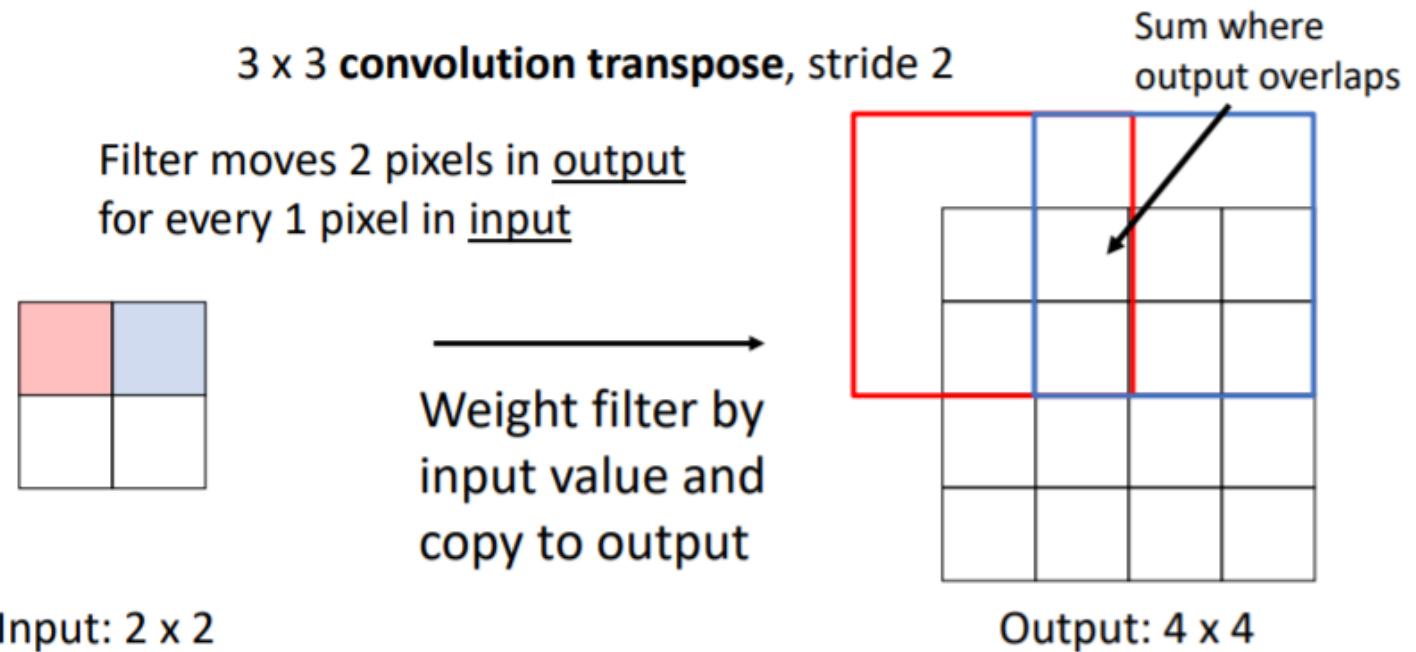
- Pair downsampling and upsampling to remember the coordinates that were downsampling, and put the value back in the place to upsampling. This is applicable when downsampled with max pooling.

# Learnable Upsampling: Transposed Convolution



- The upsampling methods described above are fixed functions and are not learnable. Transposed convolution is sometimes used as a learnable method.

# Learnable Upsampling: Transposed Convolution



- Slide the pixel one by one when the downsampled input is given. Let the 1 pixel value of this input be **input value**.
- In the above figure, we decided to use a filter with **3x3 convolution transpose, stride 2**, so multiply each of the 9 weights in this 3x3 filter by the **input value** and copy it to the appropriate position of the 4x4 output.
- The **overlap** between the filters is put as the sum of the overlapping values.

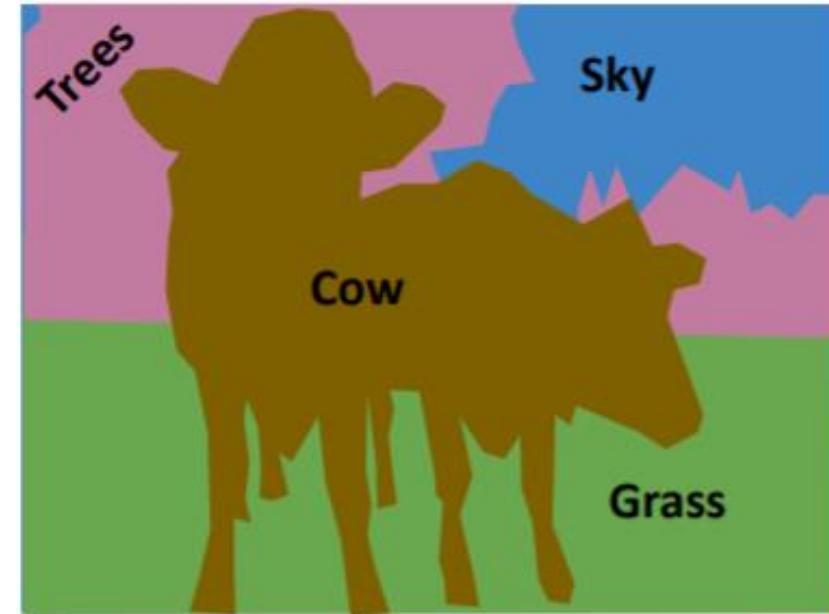
# Instance Segmentation

---

**Object Detection:** Detects individual object instances, but only gives box

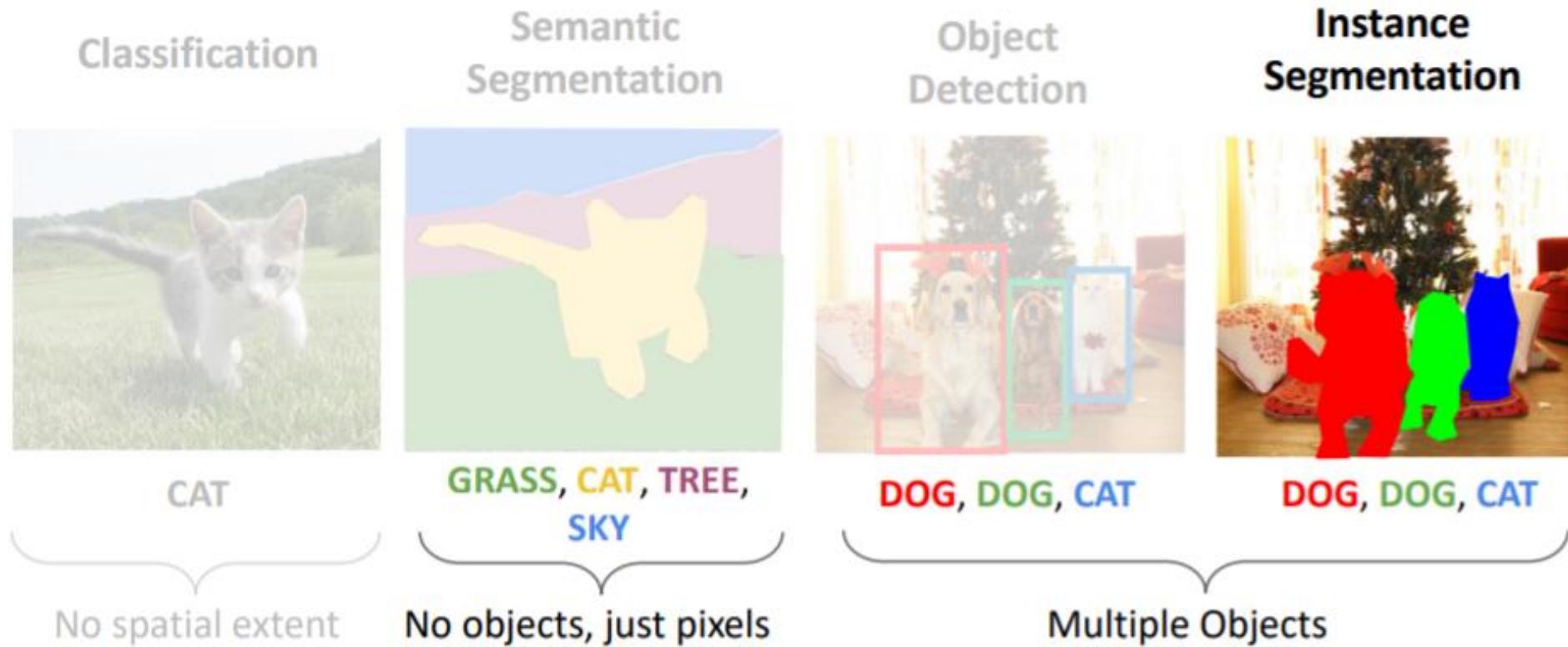


**Semantic Segmentation:** Gives per-pixel labels, but merges instances



# Instance Segmentation

---



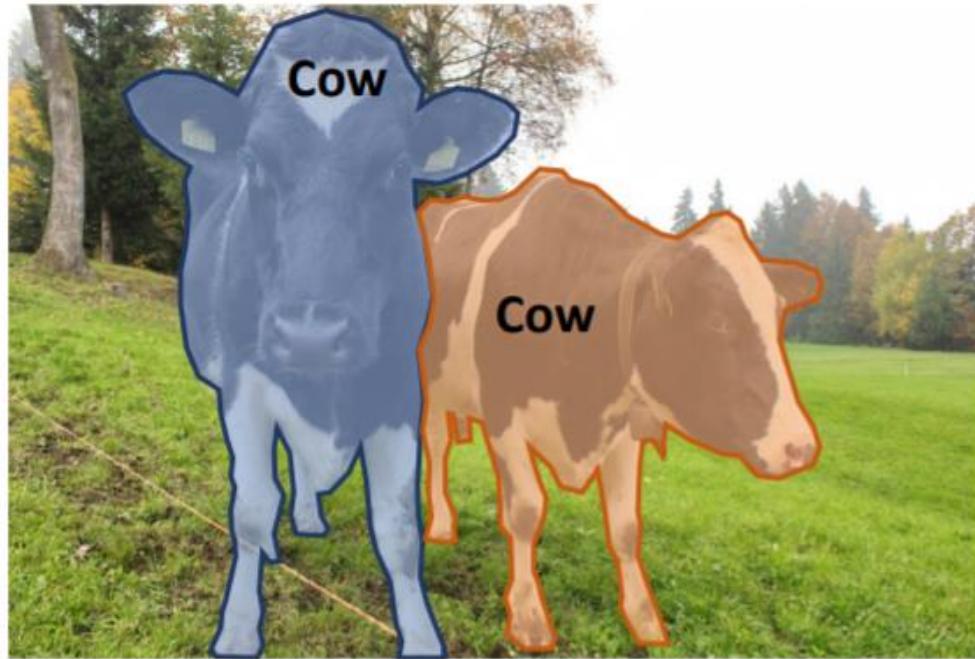
# Instance Segmentation

---

## **Instance Segmentation:**

Detect all objects in the image, and identify the pixels that belong to each object (Only things!)

**Approach:** Perform object detection, then predict a segmentation mask for each object!



- Instance Segmentation is a hybrid task that detects objects (only things) in the image and segments them by pixel.
- In other words, through this, we conduct object detection and semantic segmentation for detected objects.

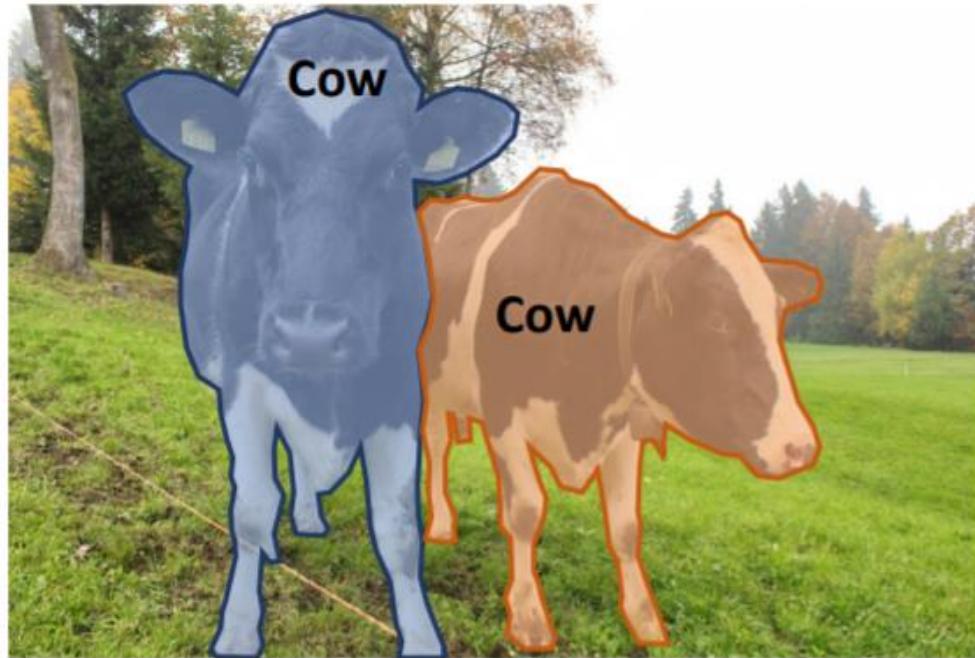
# Instance Segmentation

---

## **Instance Segmentation:**

Detect all objects in the image, and identify the pixels that belong to each object (Only things!)

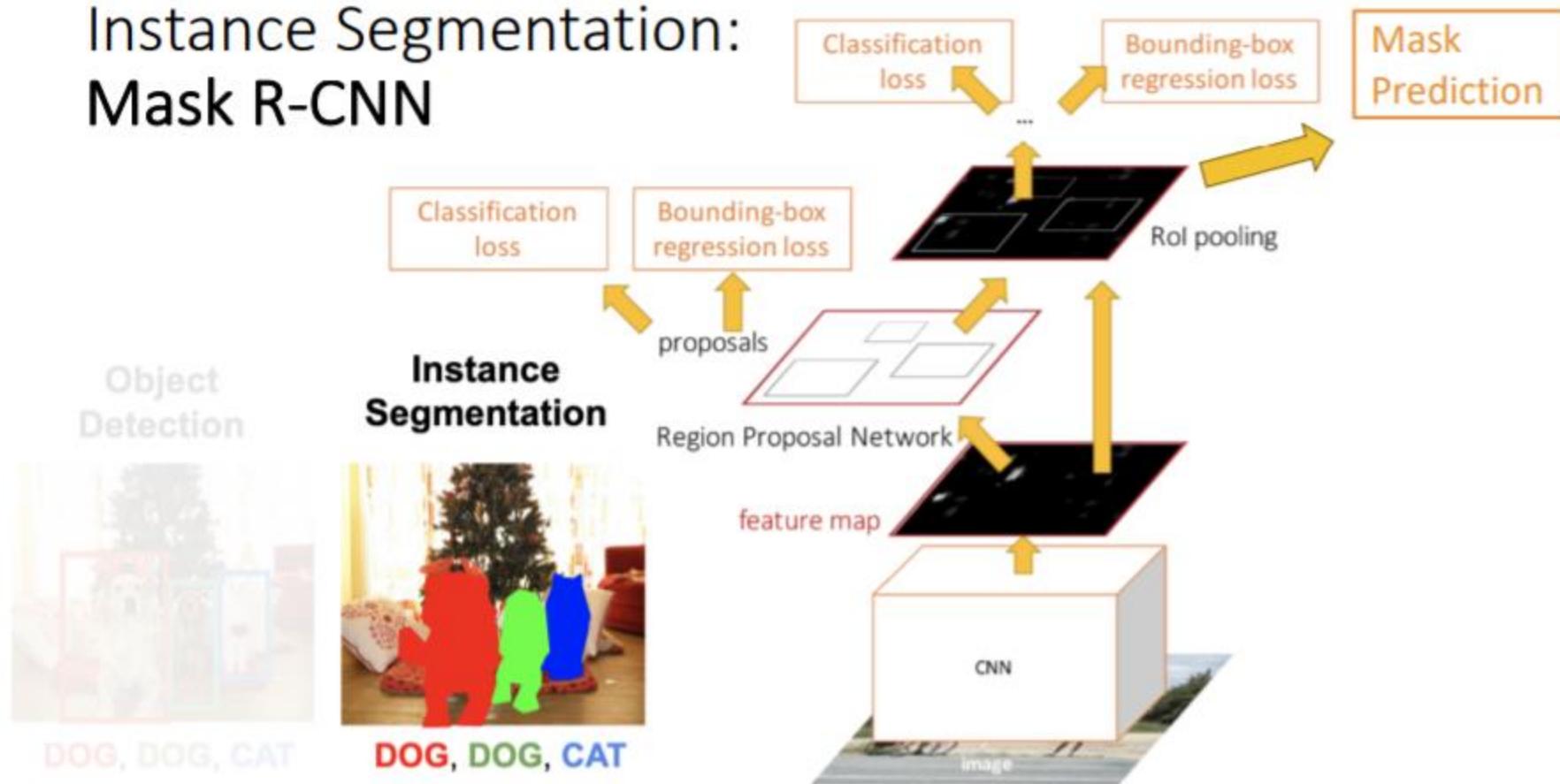
**Approach:** Perform object detection, then predict a segmentation mask for each object!



- Instance Segmentation is a hybrid task that detects objects (only things) in the image and segments them by pixel.
- In other words, through this, we conduct object detection and semantic segmentation for detected objects.

# Mask R-CNN

## Instance Segmentation: Mask R-CNN

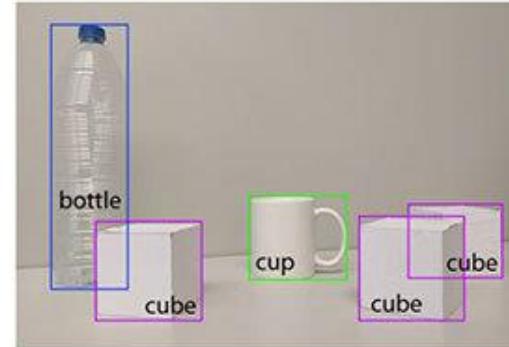


# Mask R-CNN

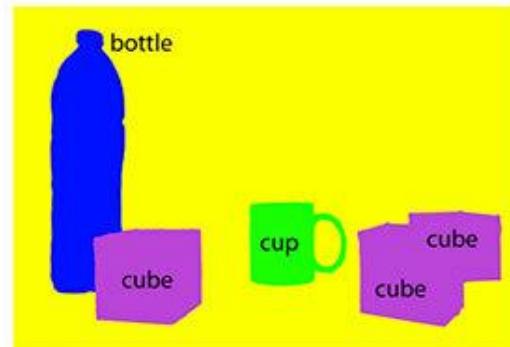
---



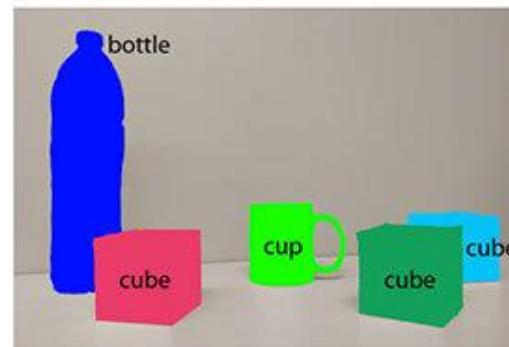
(a) Image classification



(b) Object localization



(c) Semantic segmentation

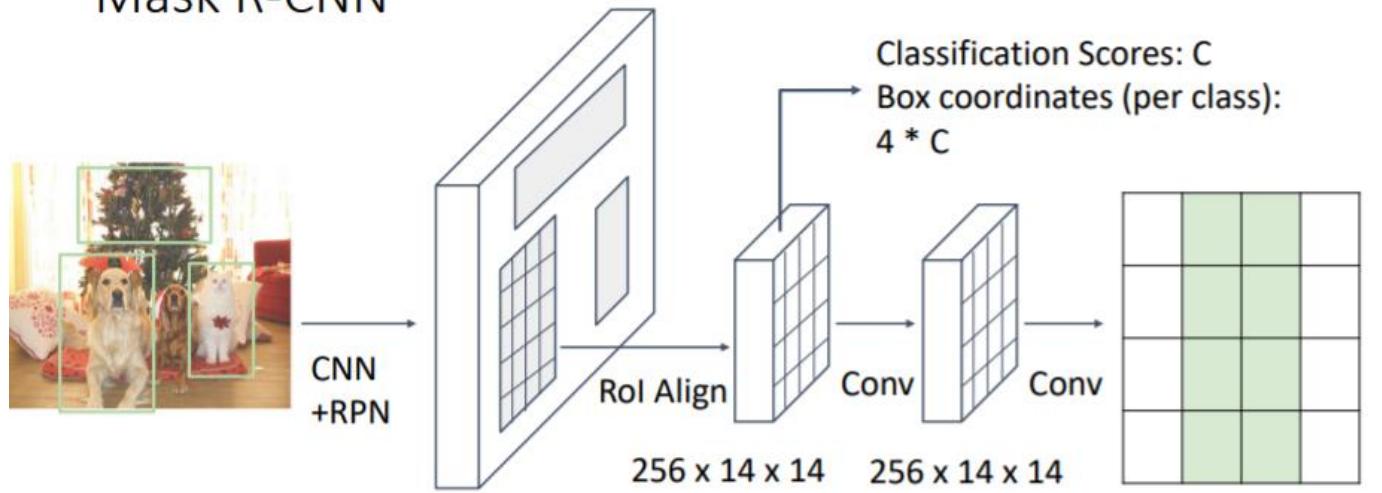


(d) Instance segmentation

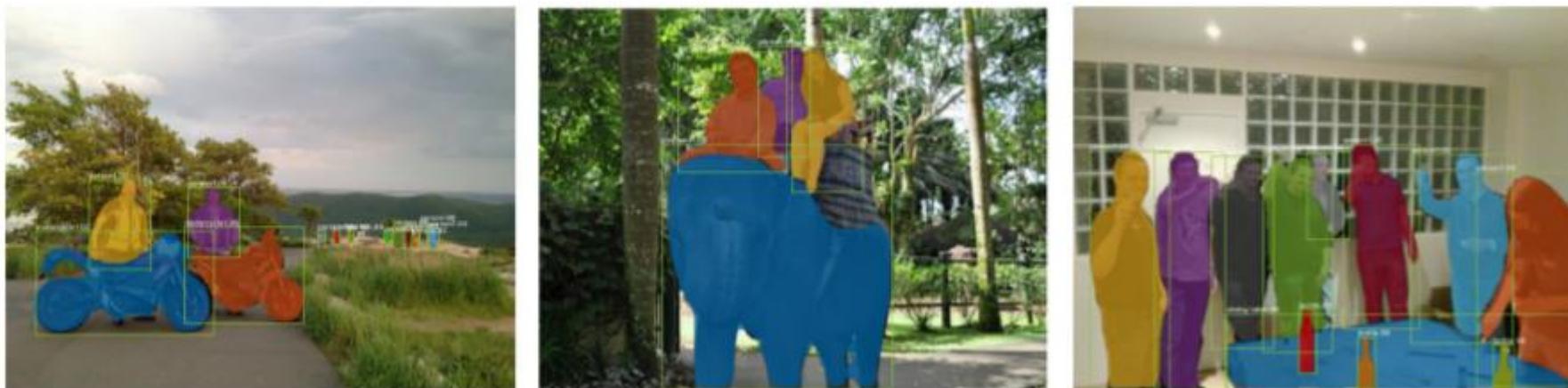
- **Instance segmentation** is a task that detects all objects present in an image while accurately classifying each instance in pixels.
- The difference is that **Semantic segmentation** does not distinguish instances of equally classified categories.
- This is a combination of object detection tasks that detect objects and semantic segmentation tasks that classify the categories of each pixel.

# Mask R-CNN

Mask R-CNN

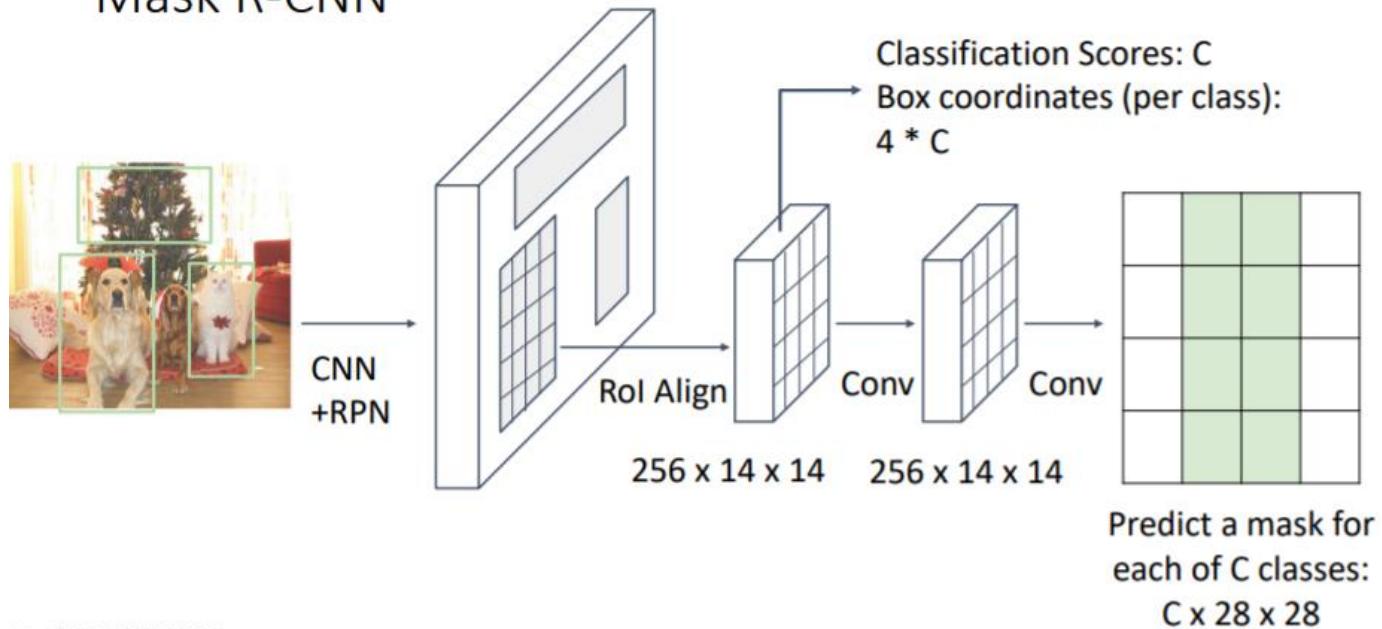


Predict a mask for  
each of  $C$  classes:  
 $C \times 28 \times 28$



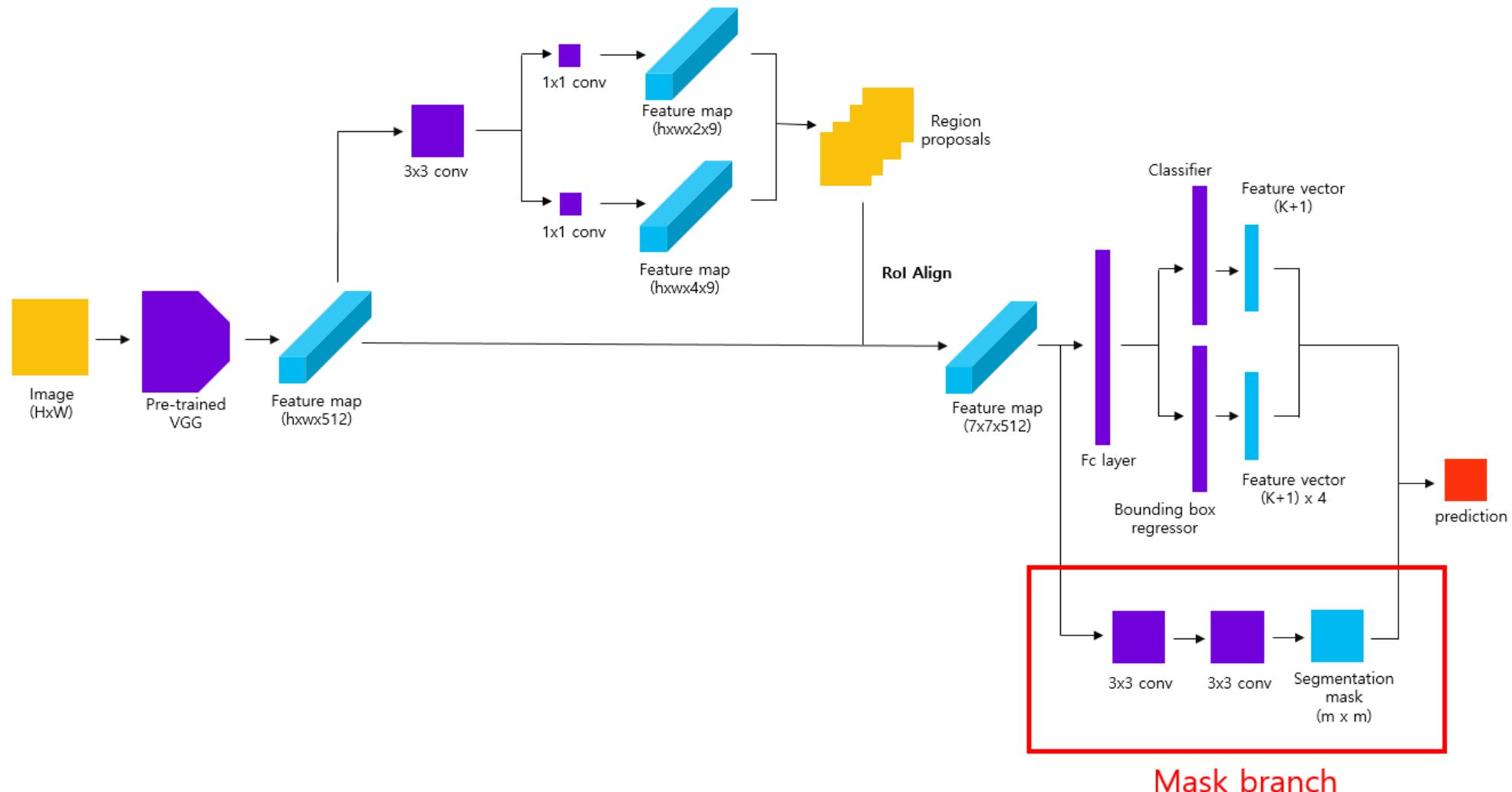
# Mask R-CNN

Mask R-CNN



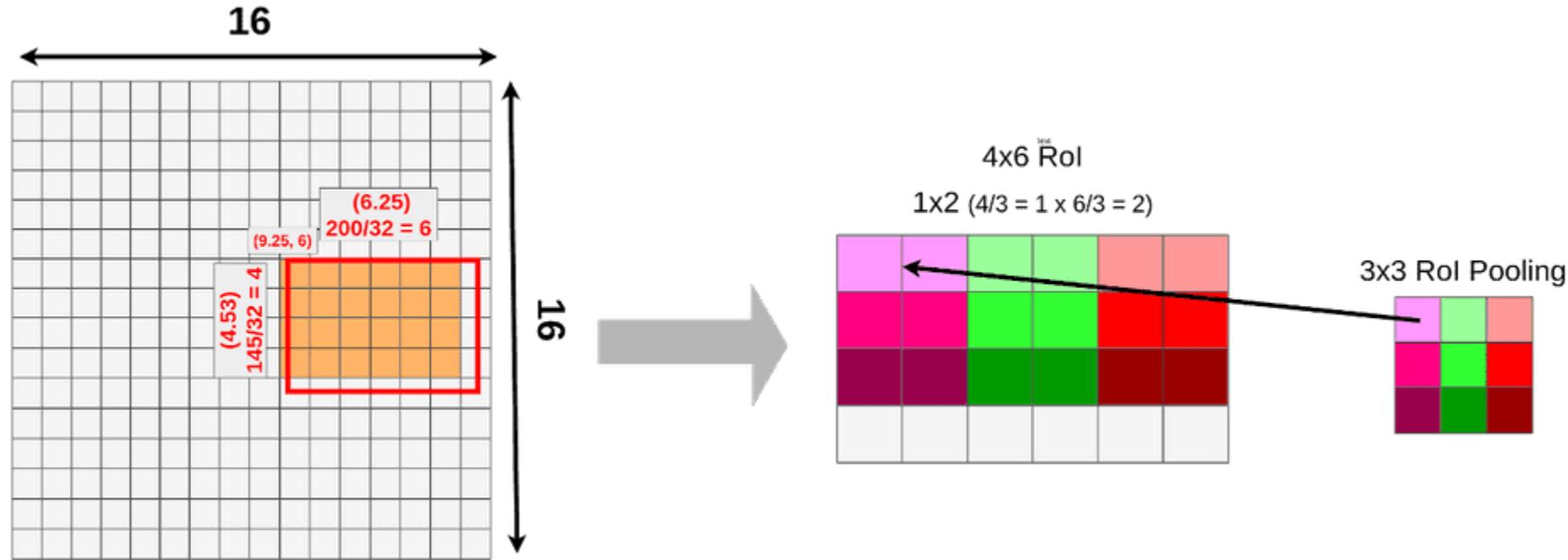
Mask R-CNN has a structure that adds a **classification branch** that predicts the class of objects, a **box regression branch** that performs box regression, and a **mask branch** that predicts the segmentation mask in parallel for the Region of Interest (RoI) obtained from Fast R-CNN's RPN. A **mask branch** is a small, fully convolutional network (FCN) added to each RoI. To perform segmentation tasks more effectively, the paper adds a **RoI Align** layer **that preserves the spatial location of objects**.

# Mask R-CNN



Mask branch

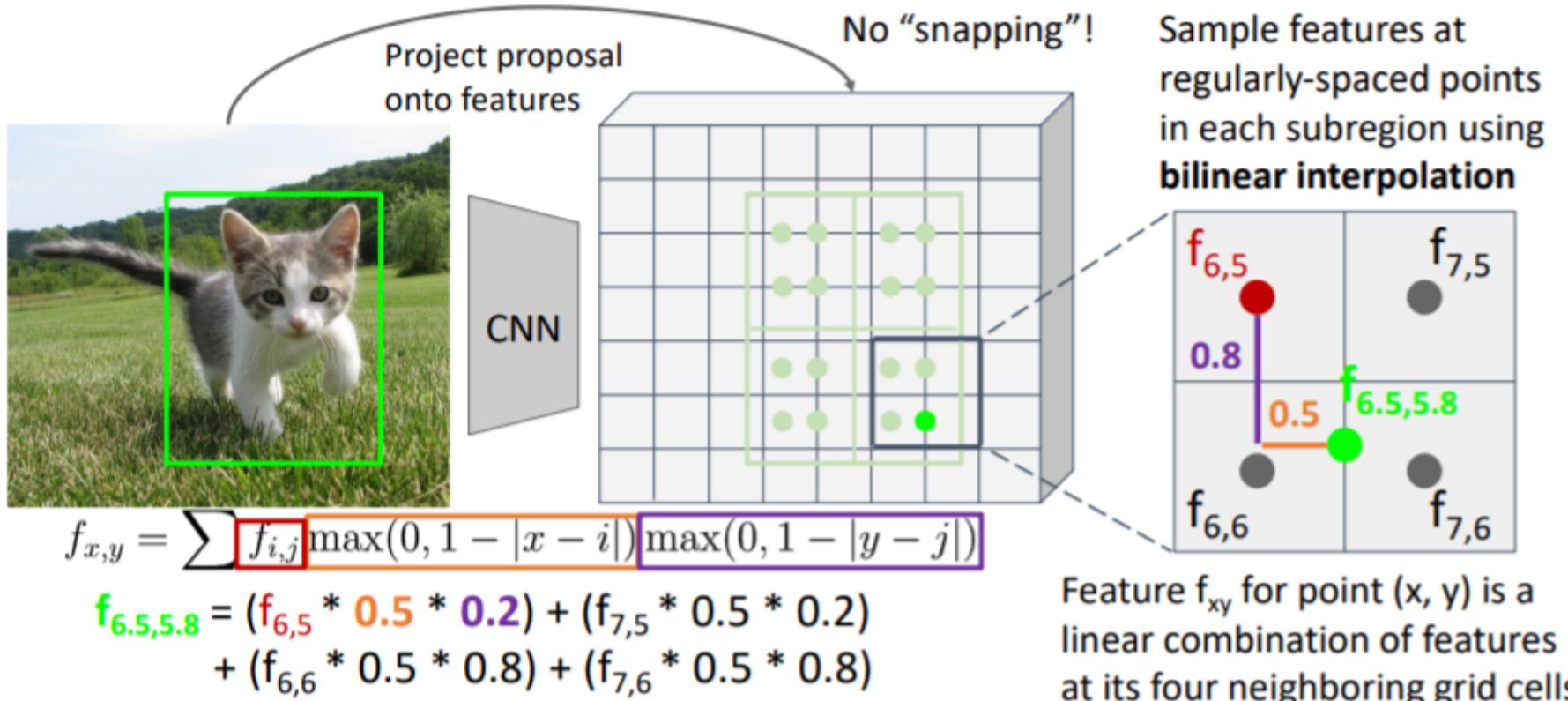
# Mask R-CNN: RoI Align



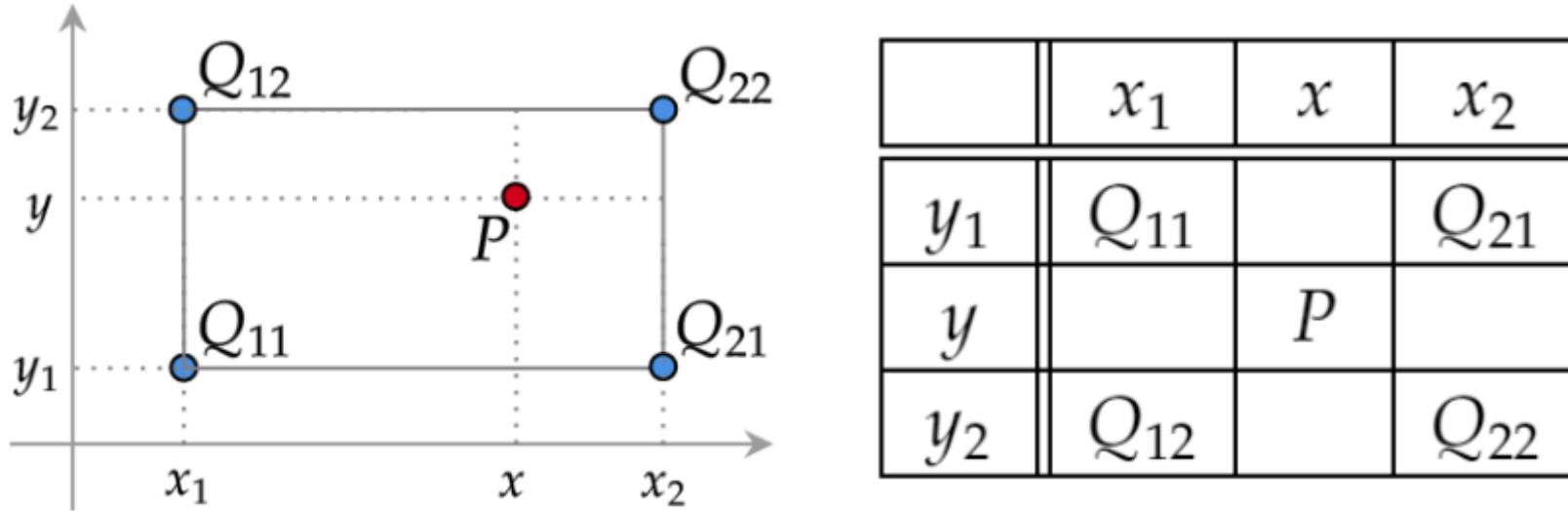
- Project RoI obtained through RPN to the size of the feature map extracted from the backbone network ( $4.53 \times 6.25 \rightarrow 4 \times 6$ )
- Misalignment !!

# Mask R-CNN: ROI Align

## Cropping Features: ROI Align

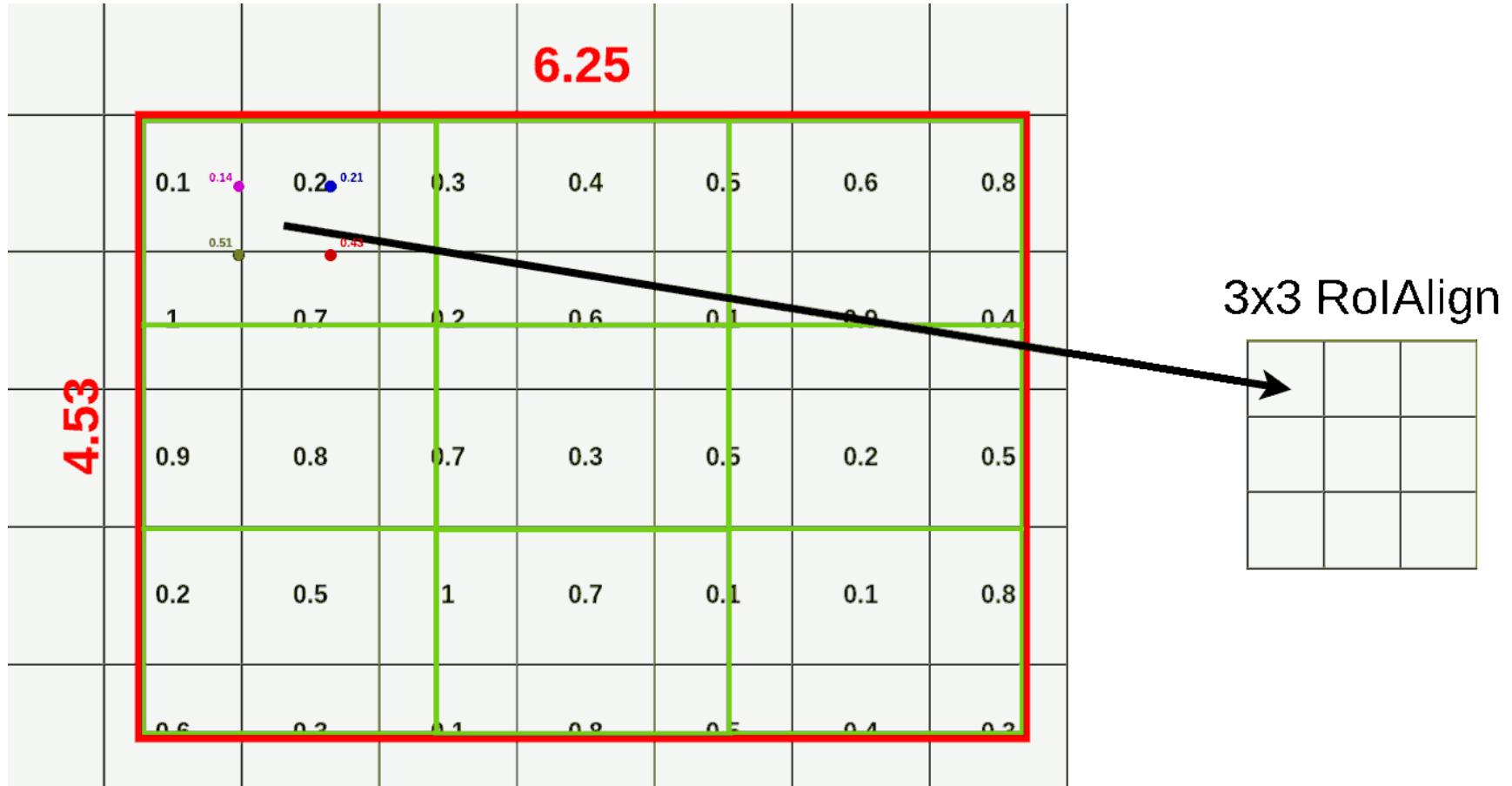


# Mask R-CNN: ROI Align



$$P \approx \frac{y_2 - y}{y_2 - y_1} \left( \frac{x_2 - x}{x_2 - x_1} Q_{11} + \frac{x - x_1}{x_2 - x_1} Q_{21} \right) + \frac{y - y_1}{y_2 - y_1} \left( \frac{x_2 - x}{x_2 - x_1} Q_{12} + \frac{x - x_1}{x_2 - x_1} Q_{22} \right)$$

# Mask R-CNN: RoI Align



# Mask R-CNN: Multi-task loss function

---

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

- $L_{mask}$ : binary cross entropy loss
  - Apply sigmoid function to each cell of  $K * m * m$  size feature map output from the mask branch and obtain loss.
  - Mask R-CNN separates the class branch and the mask branch, creates a mask for each class, and obtains the binary loss.

# Mask R-CNN: Training

## 1) Input image Pre-processing

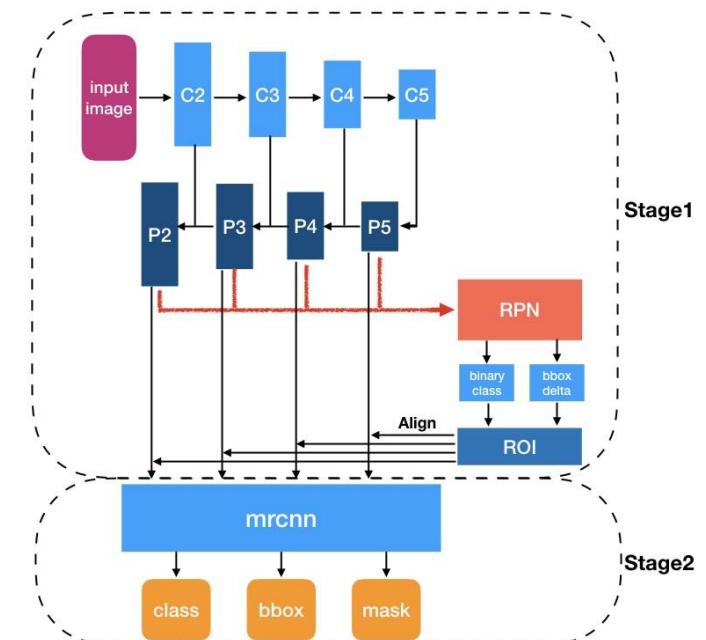
- The width or height of the original image, the shorter edge, is resized to the target size. And the longer edge is resized to preserve the aspect ratio. If the longer side exceeds the maximum size, it is resized to the maximum size and the shorter side is resized to the direction in which the aspect ratio is preserved. The default values for target size and maximum size are 800 and 1333, respectively.

## 2) Feature pyramid by backbone network

- Input : resized image
- Process : constructing feature pyramid
- Output : feature pyramid {P2, P3, P4, P5, P6}

## 3) Region proposal by RPN

- Input : feature pyramid {P2, P3, P4, P5, P6}
- Process : Region proposal
- Output : Region proposals with objectness score and bbox regressor per feature pyramid {P2, P3, P4, P5, P6}



# Mask R-CNN: Training

---

## 4) Select best RoI by Proposal layer

- Input : Region proposals
- Process : selecting top-N RoIs
- Output : top-N RoIs

## 5) feature map by RoI Align layer

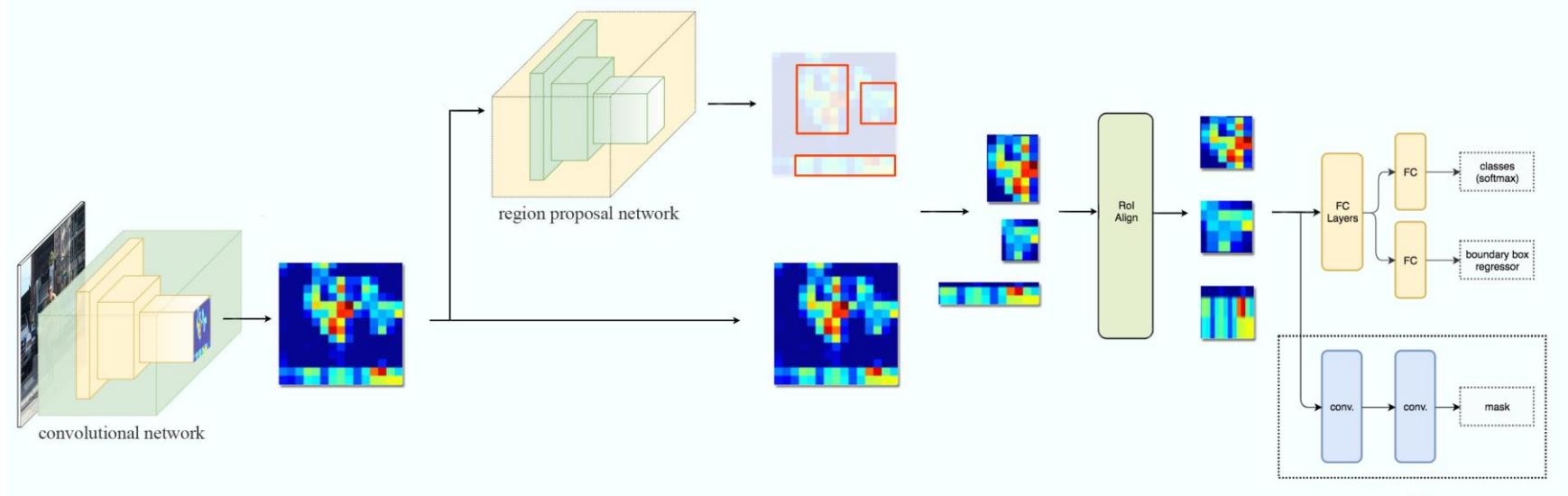
- $k = \left\lceil k_0 + \log_2 \left( \frac{\sqrt{wh}}{224} \right) \right\rceil$
- Input : feature pyramid and RoIs
- Process : RoIAlign
- Output : 7x7 sized feature map

## 6) Classification and Bounding box regression by Fast R-CNN

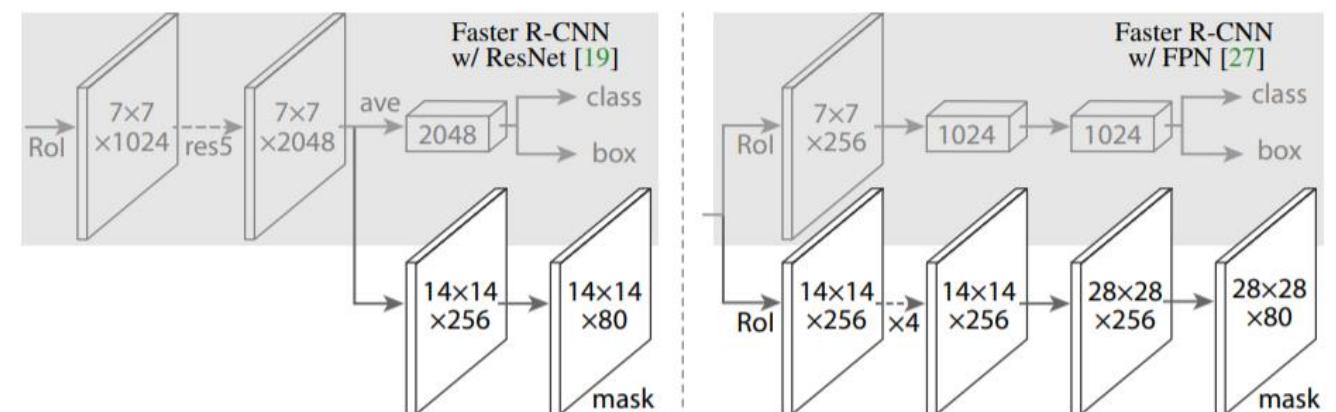
## 7) Mask segment by Mask branch

# Mask R-CNN: Training

## 6) Classification and Bounding box regression by Faster R-CNN



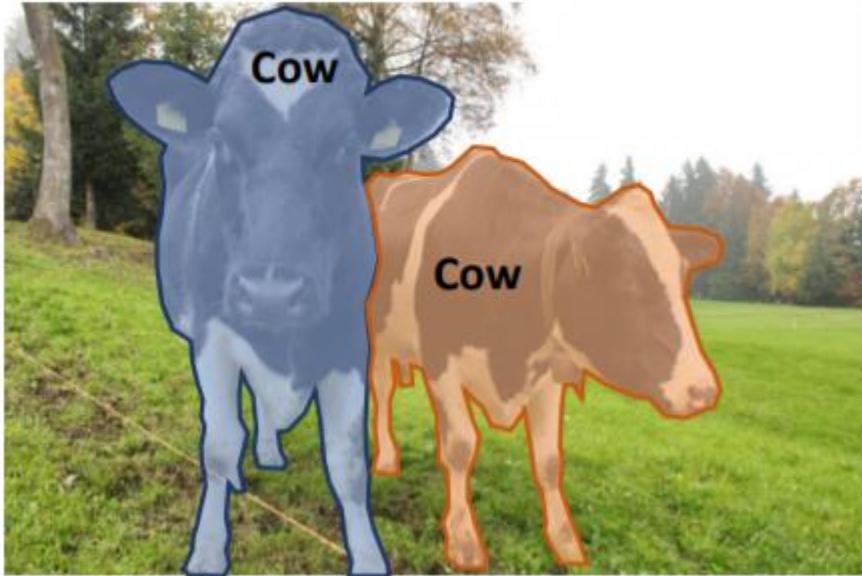
## 7) Mask segment by Mask branch



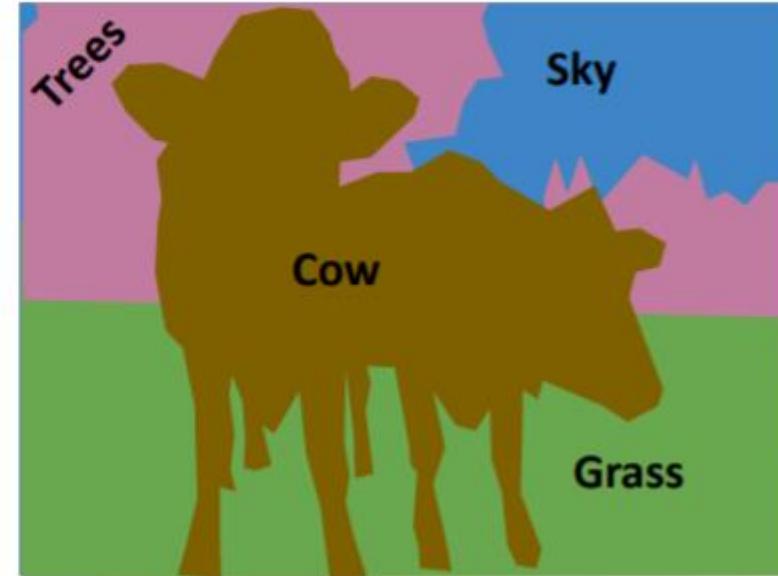
# Panoptic Segmentation

---

**Instance Segmentation:** Separate object instances, but only things



**Semantic Segmentation:** Identify both things and stuff, but doesn't separate instances



- Instance Segmentation distinguishes object instances, but **only deals with objects that correspond to things**.

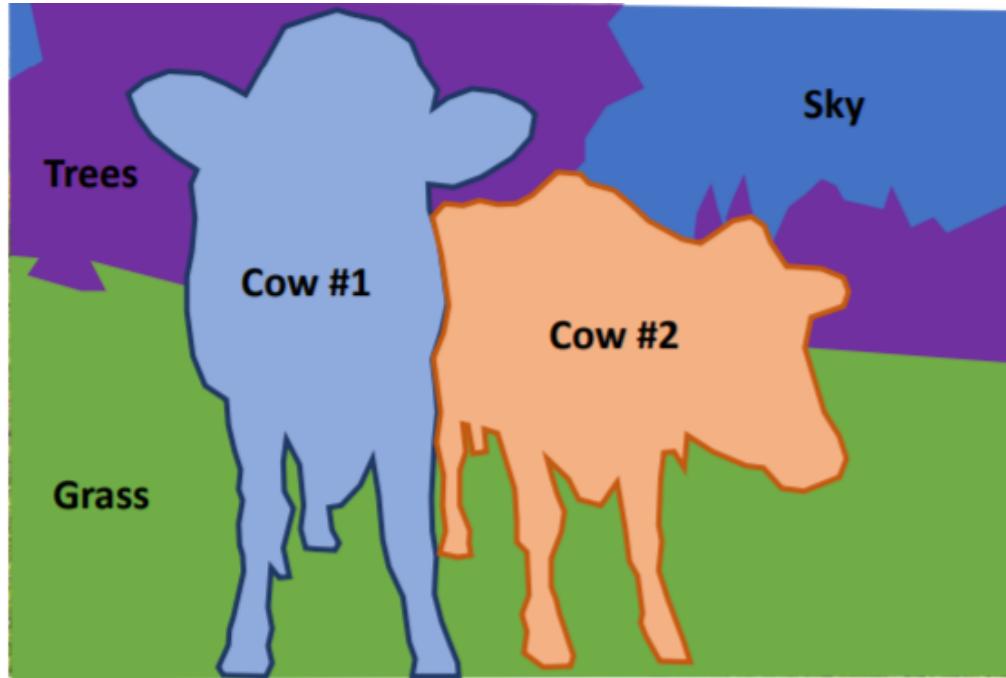
# Beyond Instance Segmentation: Panoptic Segmentation

---

Label all pixels in  
the image (both  
things and stuff)

For “thing”  
categories also  
separate into  
instances

Rilov et al. “Panoptic Segmentation”. CVPR 2019



- Label all objects for things and stuff by pixel. Things are construed by instances.

# Beyond Instance Segmentation: Panoptic Segmentation

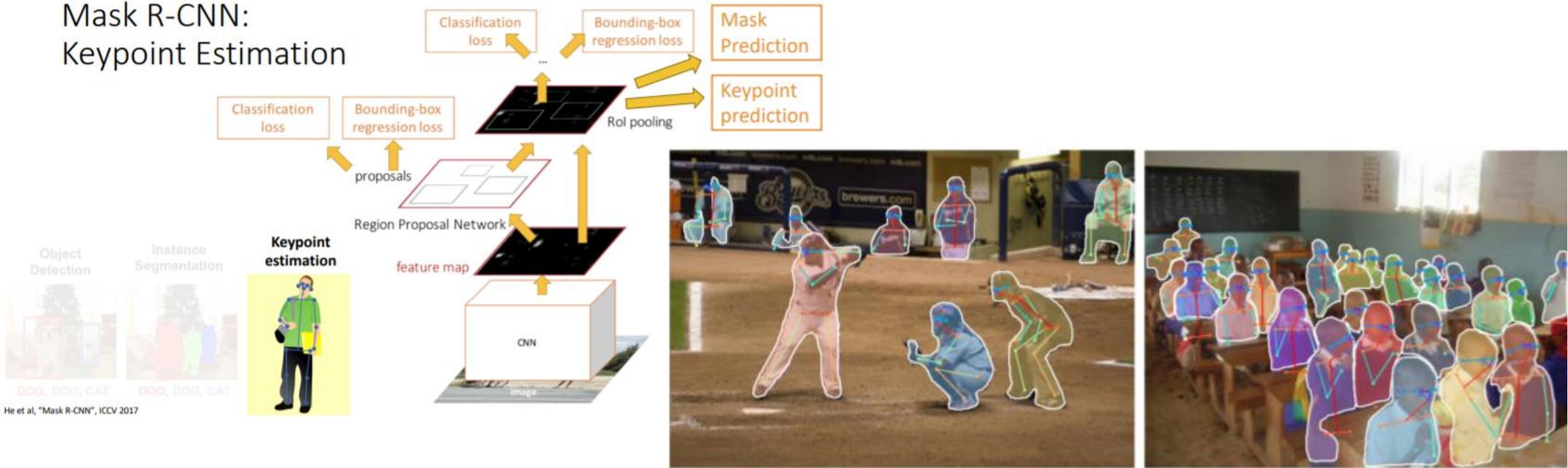
---



Kirillov et al., "Panoptic Feature Pyramid Networks", CVPR 2019

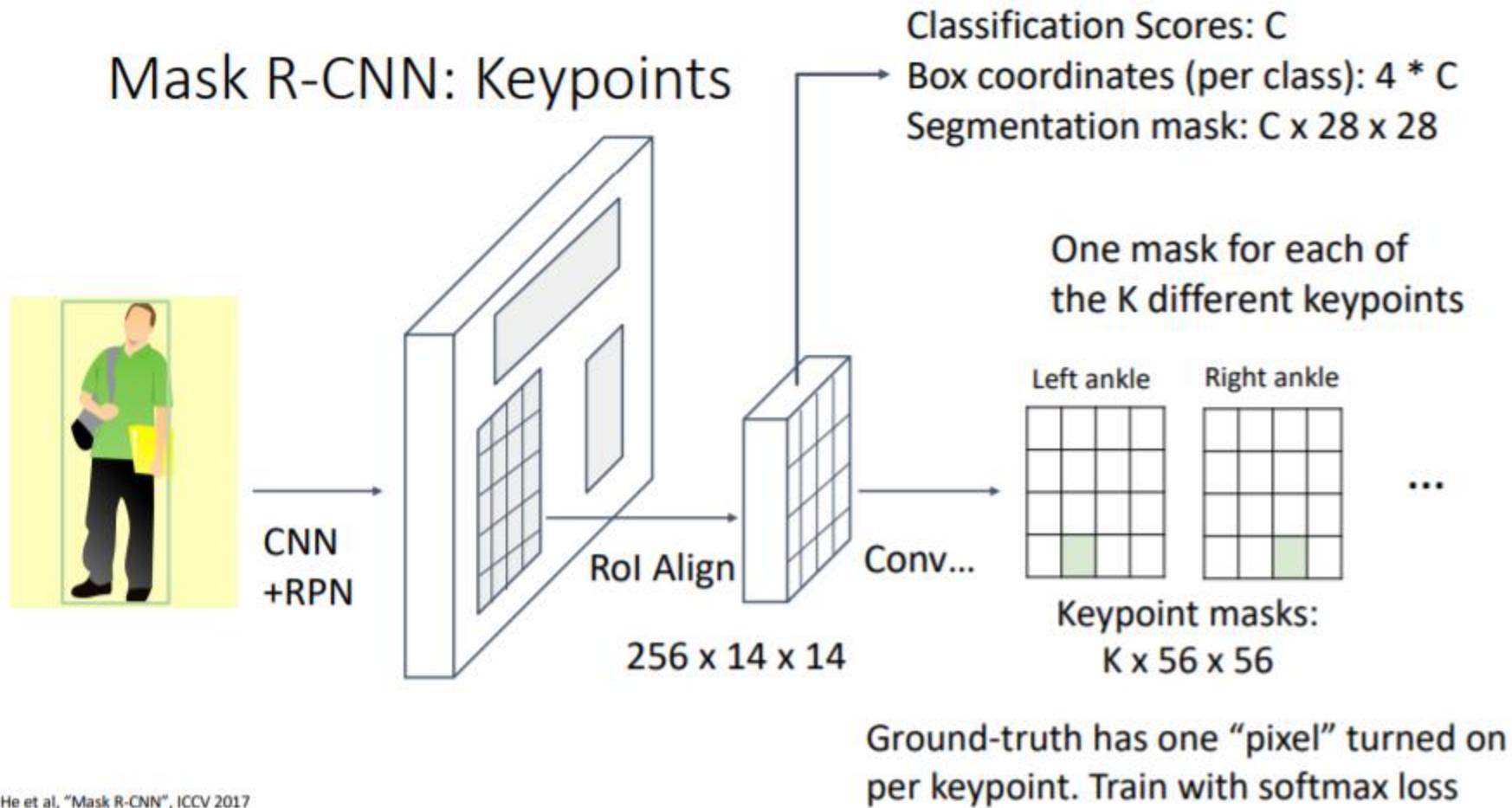
# Mask R-CNN: KeyPoints

Mask R-CNN:  
Keypoint Estimation



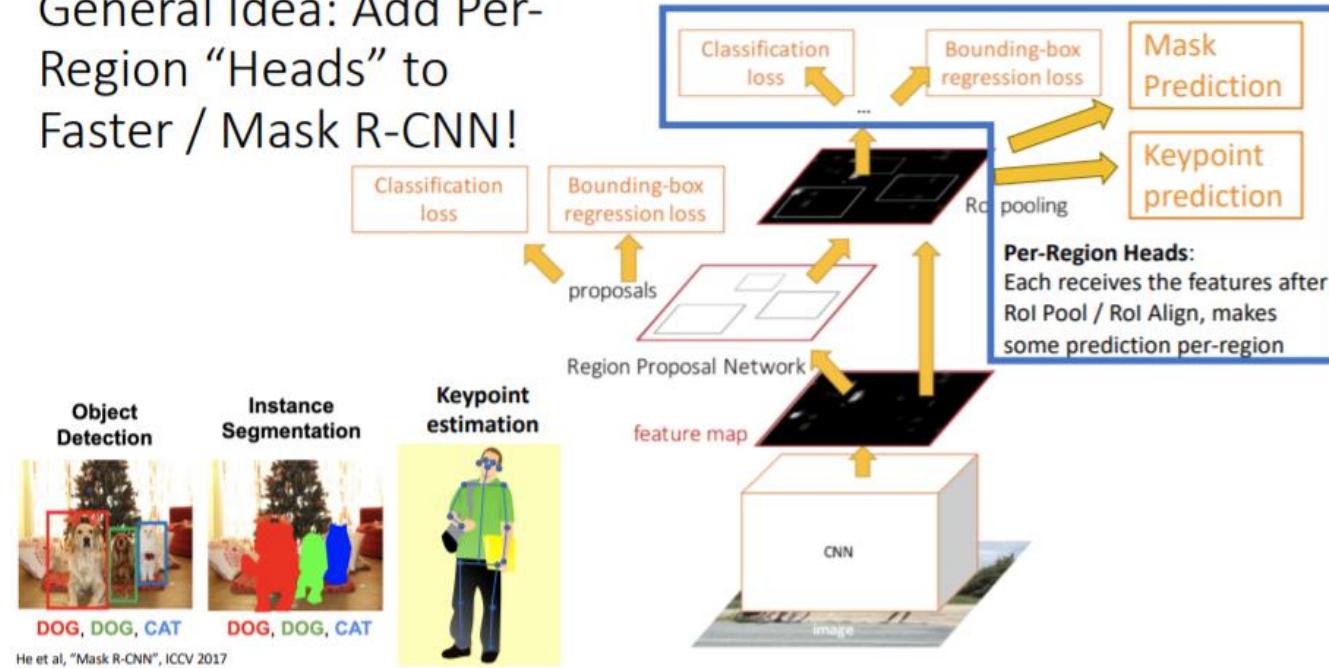
In the **keypoint prediction branch**, **K** different **keypoint masks** are applied (a mask consisting of one pixel)

# Mask R-CNN: KeyPoints



# General Idea: Add PerRegion “Heads” to Faster / Mask R-CNN!

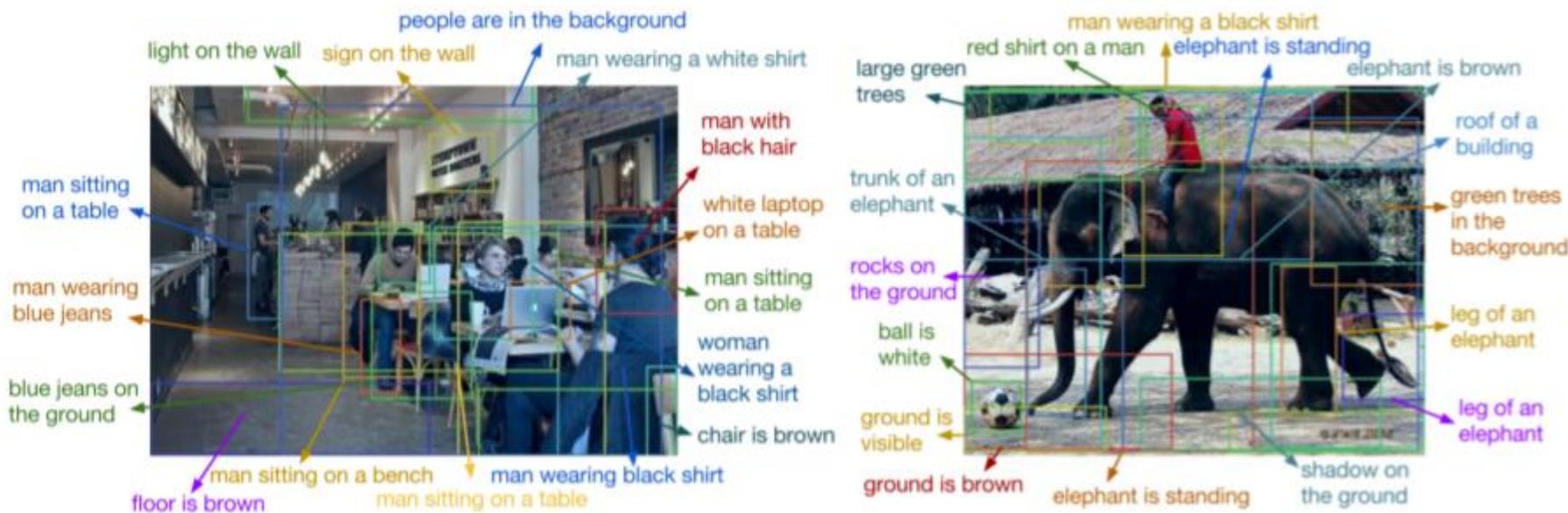
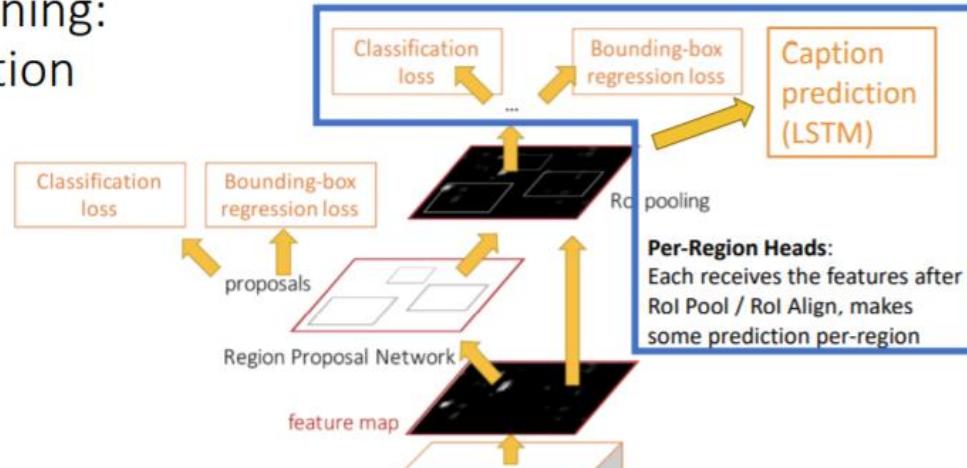
General Idea: Add Per-Region “Heads” to Faster / Mask R-CNN!



It can be applied to a **new computer vision task** using the method of **adding branches**.

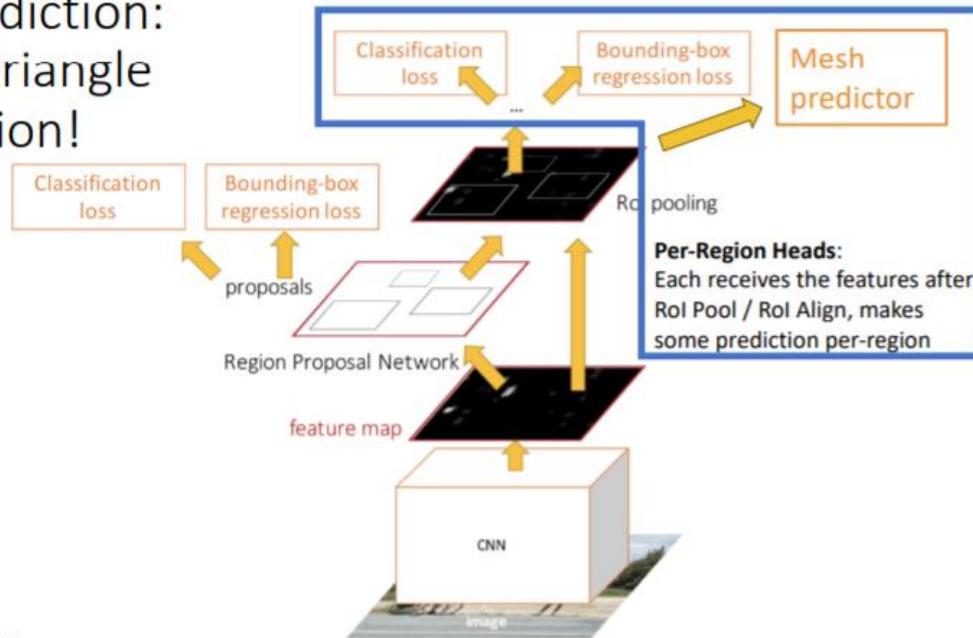
# Dense Captioning

Dense Captioning:  
Predict a caption  
per region!



# Mesh R-CNN

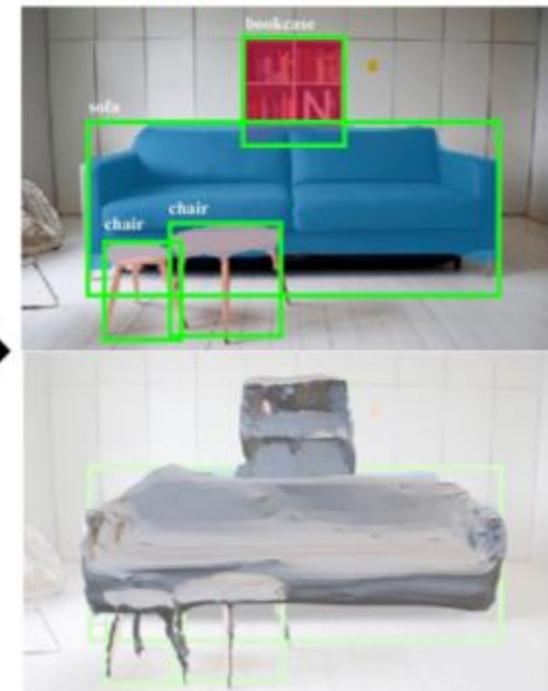
3D Shape Prediction:  
Predict a 3D triangle  
mesh per region!



Mask R-CNN:  
2D Image -> 2D shapes



**Mesh R-CNN:**  
2D Image -> 3D shapes

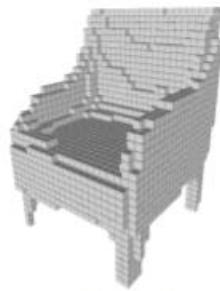


# 3D Shape Representations

Predicting 3D Shapes  
from single image

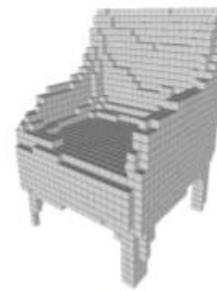


Input Image

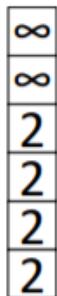


3D Shape

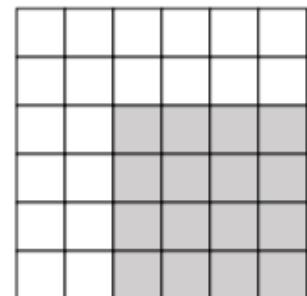
Processing 3D  
input data



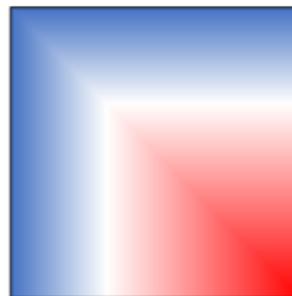
Chair



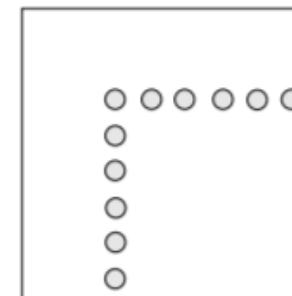
Depth  
Map



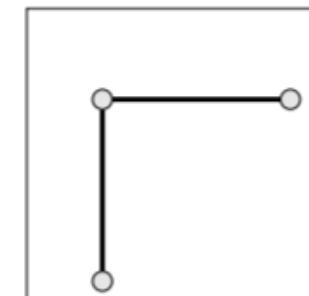
Voxel  
Grid



Implicit  
Surface



Pointcloud



Mesh

# 3D Shape Representations: Triangle Mesh

Represent a 3D shape as a set of triangles

**Vertices:** Set of  $V$  points in 3D space

**Faces:** Set of triangles over the vertices

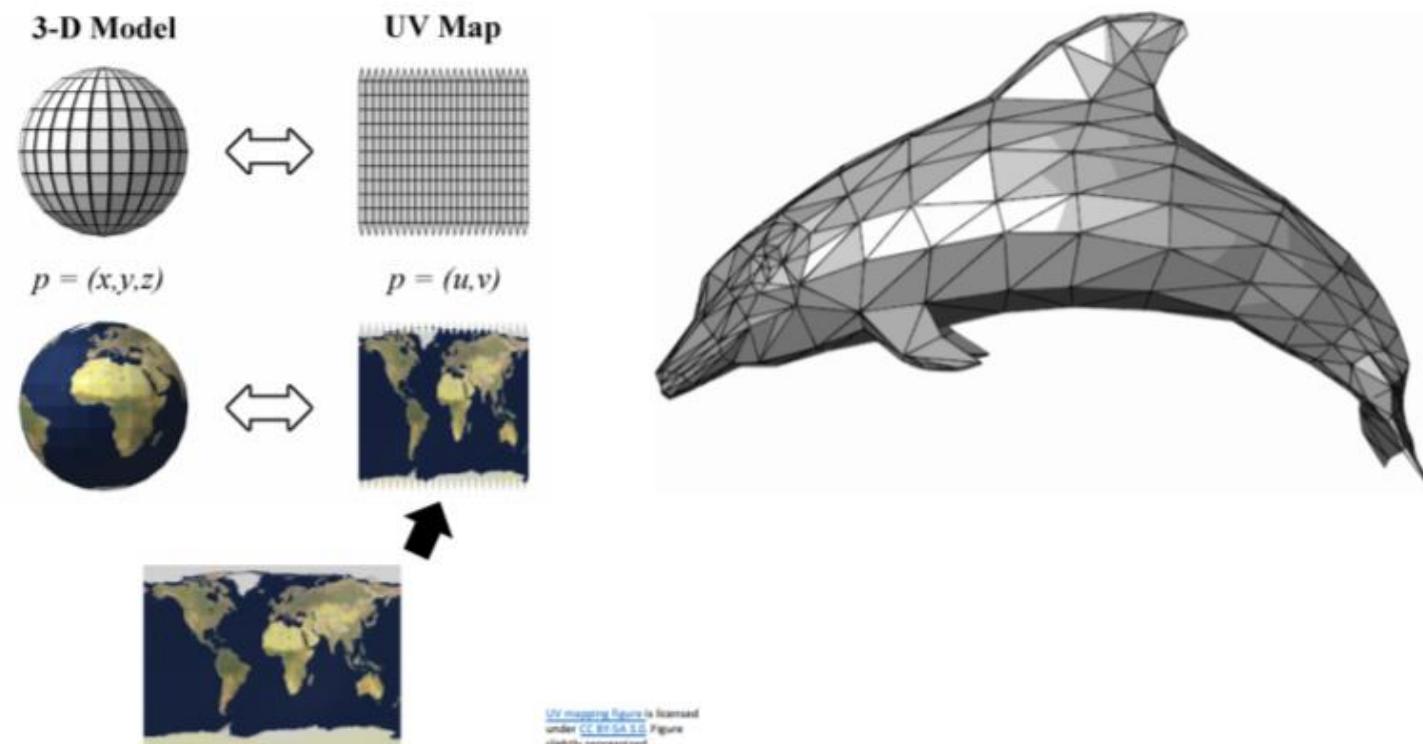
(+) Standard representation for graphics

(+) Explicitly represents 3D shapes

(+) Adaptive: Can represent flat surfaces very efficiently, can allocate more faces to areas with fine detail

(+) Can attach data on verts and interpolate over the whole surface: RGB colors, texture coordinates, normal vectors, etc.

(-) Nontrivial to process with neural nets!



- Effective when **representing a flat surface**, and multiple details can be expressed by **adjusting the size of the triangular faces**.
- Vertex information can be put into Faces (the entire surface) using **coordinate interpolation**, etc.
- However, it is **not easy for the neural nets to perform**

# Mesh R-CNN

---

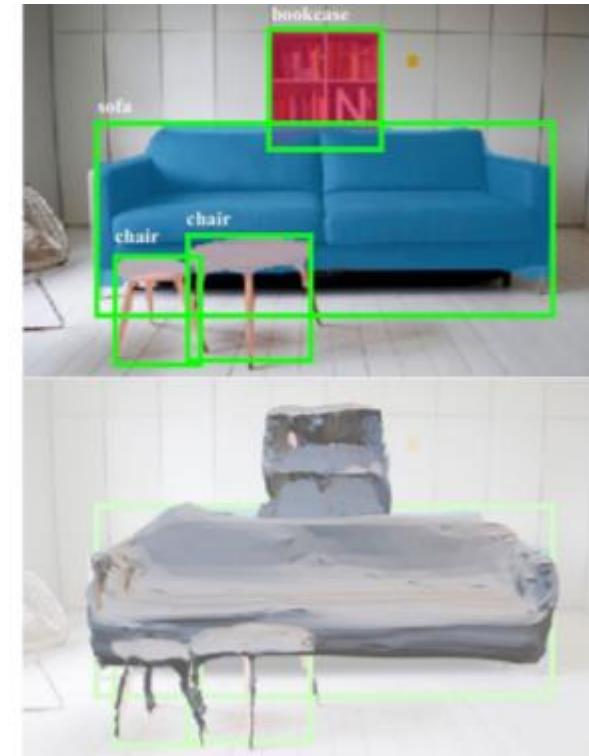
**Input:** Single RGB image

**Output:**

Mask R-CNN

Mesh head

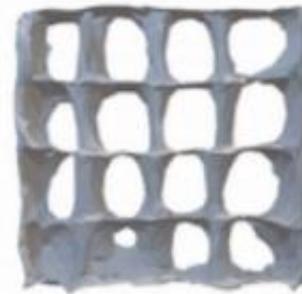
A set of detected objects  
For each object:  
- Bounding box  
- Category label  
- Instance segmentation  
- 3D triangle mesh



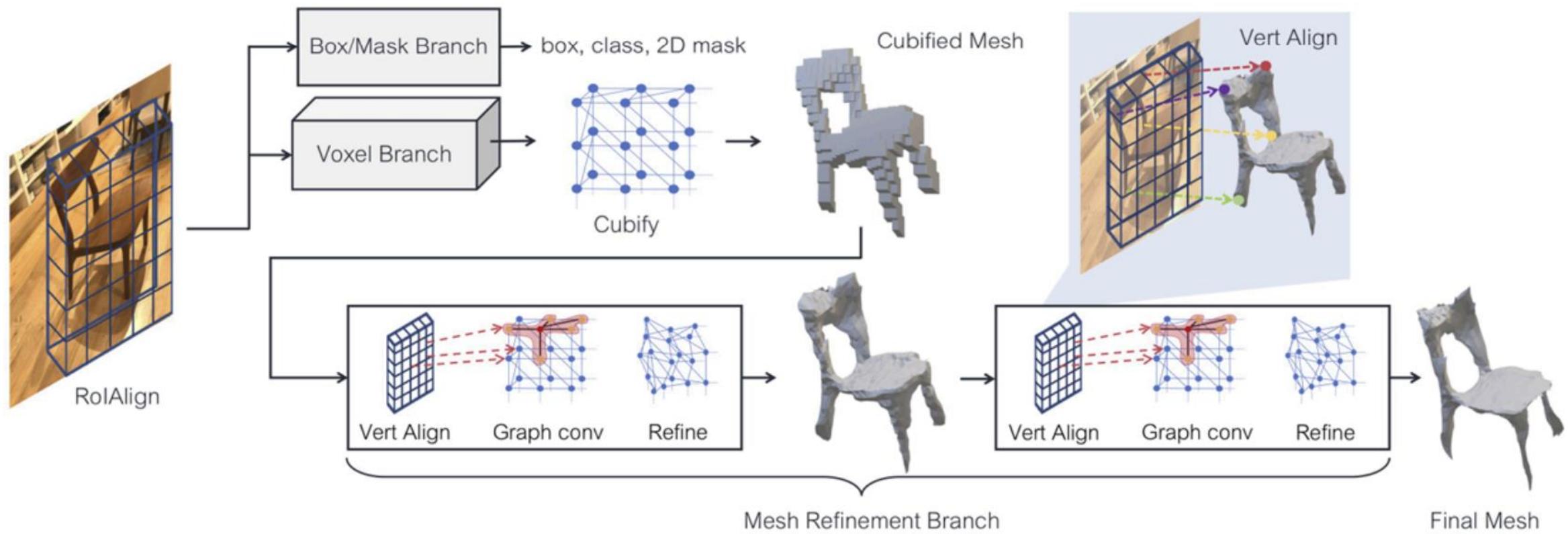
# Mesh R-CNN

---

Mesh R-CNN: Pix3D Results



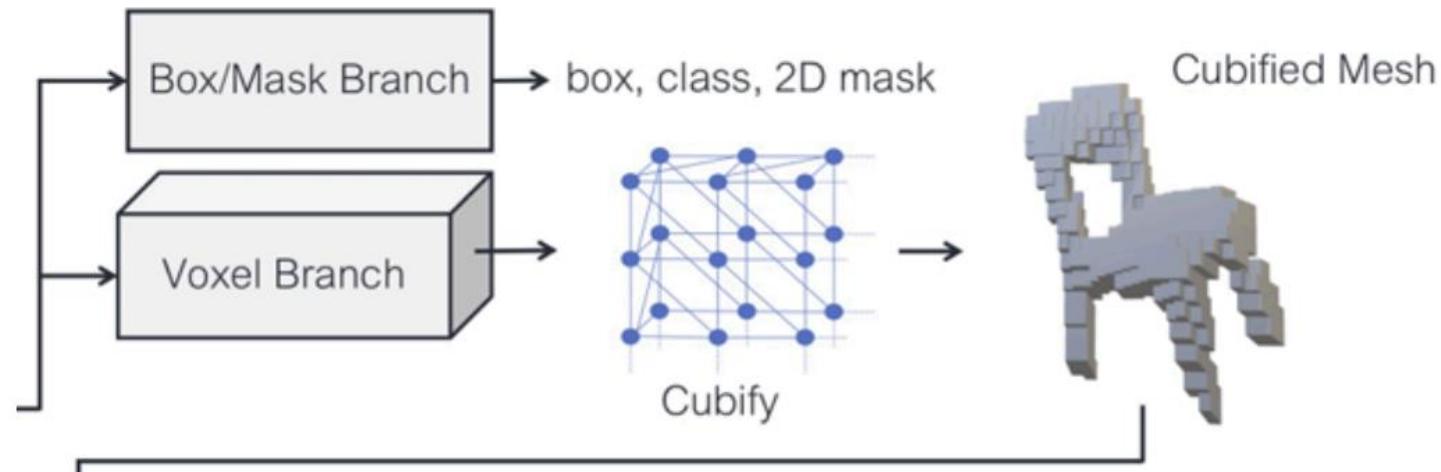
# Mesh R-CNN: Method



The **voxel branch** predicts a **coarse shape** for each detected object which is further deformed with a sequence of refinement stages in the **mesh refinement branch**

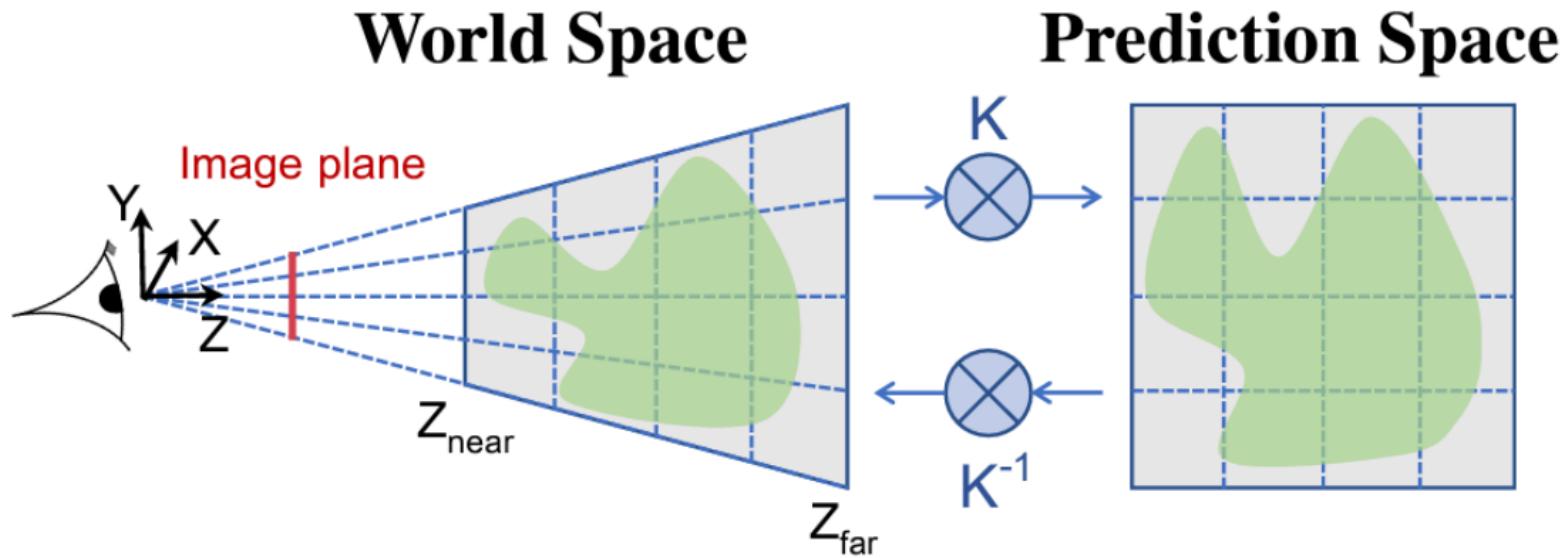
# Mesh R-CNN: Voxel Branch

---



- The **voxel branch** predicts a **grid of voxel occupancy probabilities**.
- Rather than predicting a  $M \times M$  grid giving the object's shape in the image plane, we instead **predict a  $G \times G \times G$  grid giving the object's full 3D shape**.
- Each occupied voxel is **replaced with a cubified triangle mesh** with 8 vertices, 18 edges, and 12 faces.

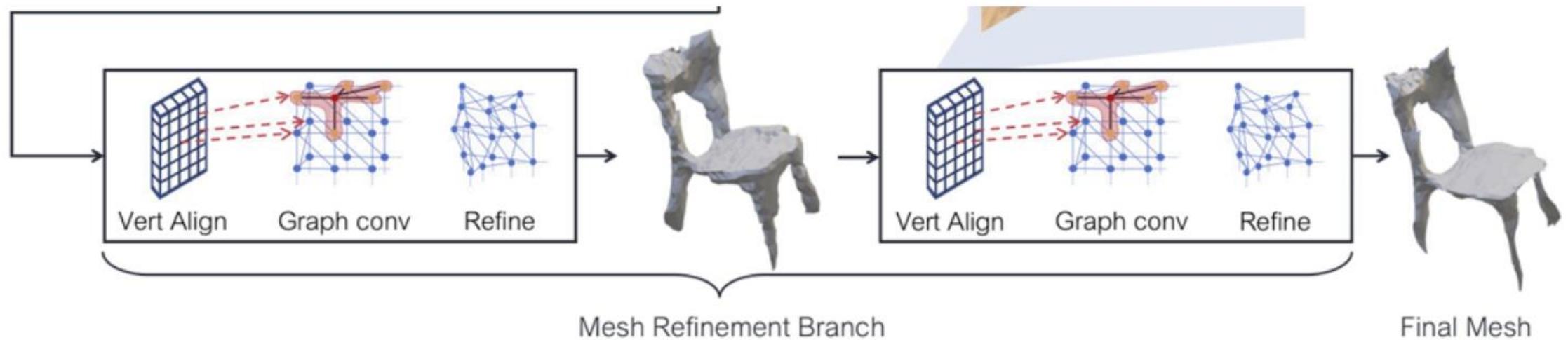
# Mesh R-CNN: Voxel Branch



Objects become smaller as they recede from the camera.

Achieved this effect by **making voxel predictions in a space that is transformed by the camera's (known) intrinsic matrix  $K$** .

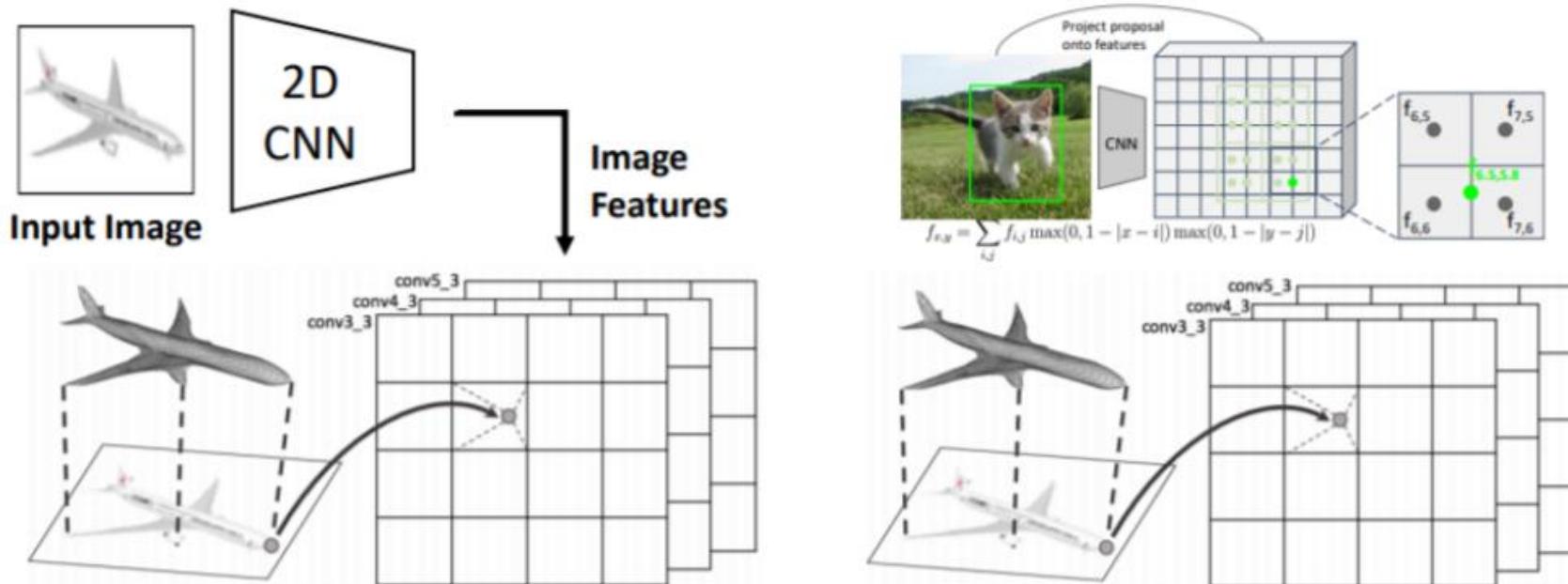
# Mesh R-CNN: Mesh Refinement Branch



The mesh refinement branch processes this initial cubified mesh, refining its vertex positions with a sequence of refinement stages

- **Vertex alignment** : extracts image features for vertices
- **Graph convolution** : propagates information along mesh edges
- **Vertex refinement** : updates vertex positions

# Mesh R-CNN: Vertex Alignment



- Project mesh's verticies to image plane (Image Feature map)
- Bilinear Interpolation -> exact image feature vector

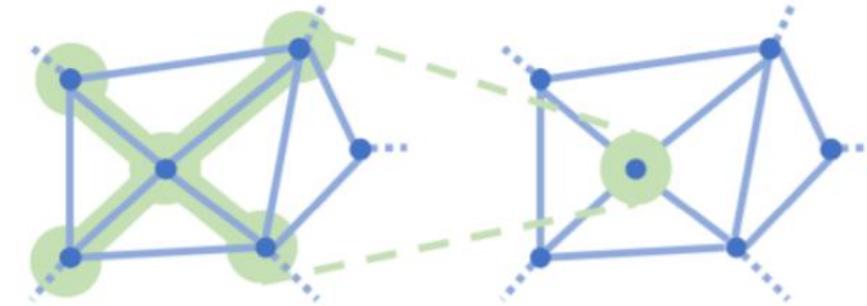
# Mesh R-CNN: Vertex Graph Convolution

$$f'_i = W_0 f_i + \sum_{j \in N(i)} W_1 f_j$$

Vertex  $v_i$  has feature  $f_i$

New feature  $f'_i$  for vertex  $v_i$  depends on feature of neighboring vertices  $N(i)$

Use same weights  $W_0$  and  $W_1$  to compute all outputs



**Input:** Graph with a feature vector at each vertex

**Output:** New feature vector for each vertex

- **Input:** Graph that contains feature vector  $f_i$  for each vertex.
- **Output:** new feature vector  $f'_i$  for each vertex

# Mesh R-CNN: Vertex Refinement

---

$$v'_i = v_i + \tanh(W_{vert} [f_i; v_i])$$

Vertex Refinement **computes updated vertex positions** where  $W_{vert}$  is a learned weight matrix. This updates the mesh geometry, keeping its topology fixed.

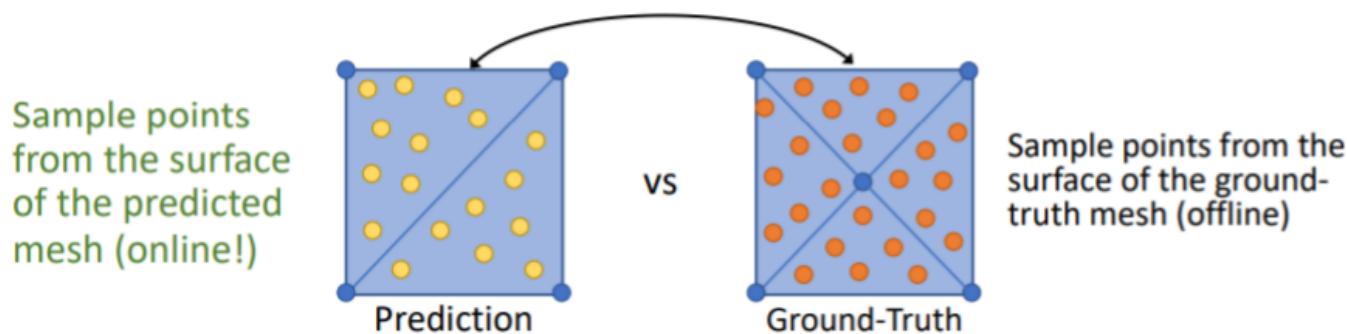
# Mesh R-CNN: Loss

## Method - Mesh Predictor

### Mesh Losses

Defining losses that operate natively on triangle meshes is challenging, so we instead use **loss functions defined over a finite set of points**. We represent a **mesh with a pointcloud by densely sampling its surface**. Consequently, a **pointcloud loss approximates a loss over shapes**.

Loss = Chamfer distance between **predicted samples** and **ground-truth samples**



$$\mathcal{L}_{\text{cham}}(P, Q) = |P|^{-1} \sum_{(p,q) \in \Lambda_{P,Q}} \|p - q\|^2 + |Q|^{-1} \sum_{(q,p) \in \Lambda_{Q,P}} \|q - p\|^2$$

$$\mathcal{L}_{\text{norm}}(P, Q) = -|P|^{-1} \sum_{(p,q) \in \Lambda_{P,Q}} |u_p \cdot u_q| - |Q|^{-1} \sum_{(q,p) \in \Lambda_{Q,P}} |u_q \cdot u_p|$$

# Mesh R-CNN: Loss

## Method - Mesh Predictor

### Edge loss

The **chamfer** and **normal distances** penalize mismatched positions and normals between two pointclouds, but minimizing these distances alone results in **degenerate meshes**.

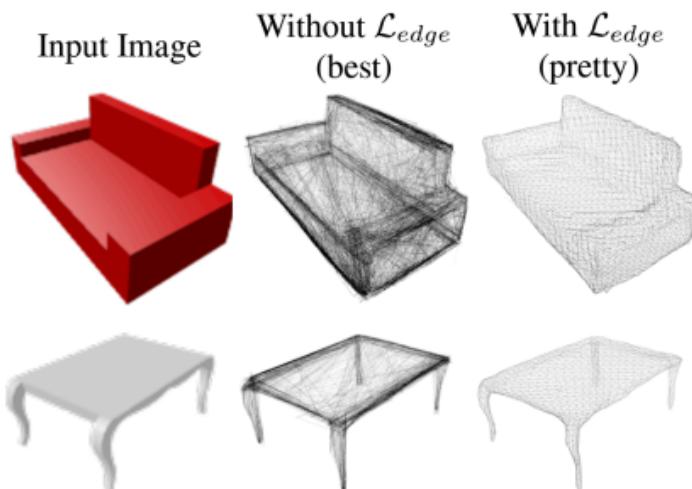


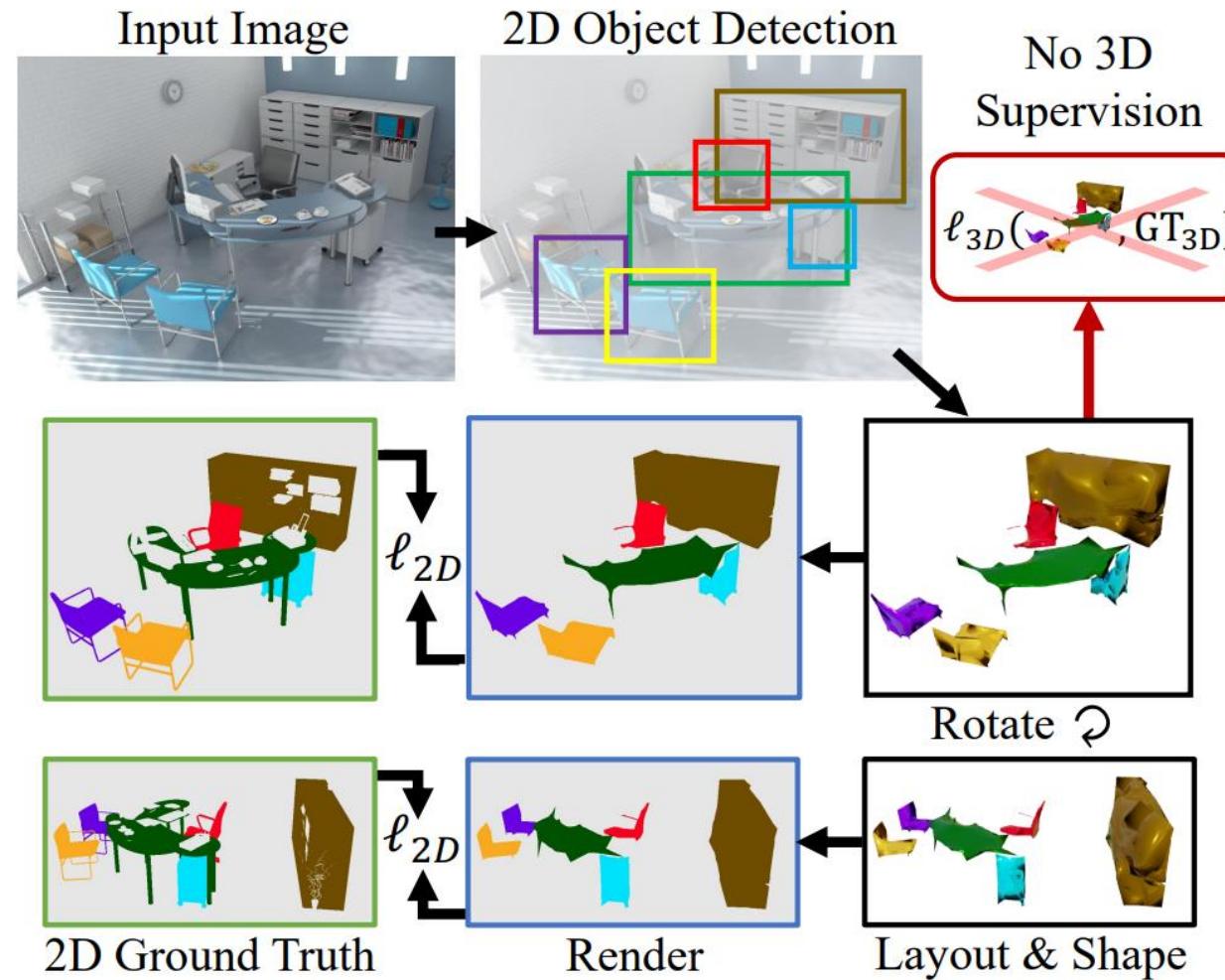
Figure 5. Training without the edge length regularizer  $\mathcal{L}_{edge}$  results in degenerate predicted meshes that have many overlapping faces. Adding  $\mathcal{L}_{edge}$  eliminates this degeneracy but results in worse agreement with the ground-truth as measured by standard metrics such as Chamfer distance.

High-quality mesh predictions require additional **shape regularizers**

$$\mathcal{L}_{edge}(V, E) = \frac{1}{|E|} \sum_{(v, v') \in E} \|v - v'\|^2$$

The mesh loss of the  $i$ -th stage is a weighted sum of  $\mathcal{L}_{\text{cham}}(P^i, P^{gt})$ ,  $\mathcal{L}_{\text{norm}}(P^i, P^{gt})$  and  $\mathcal{L}_{\text{edge}}(V^i, E^i)$

# Learning 3D Object Shape and Layout without 3D Supervision



# Learning 3D Object Shape and Layout without 3D Supervision

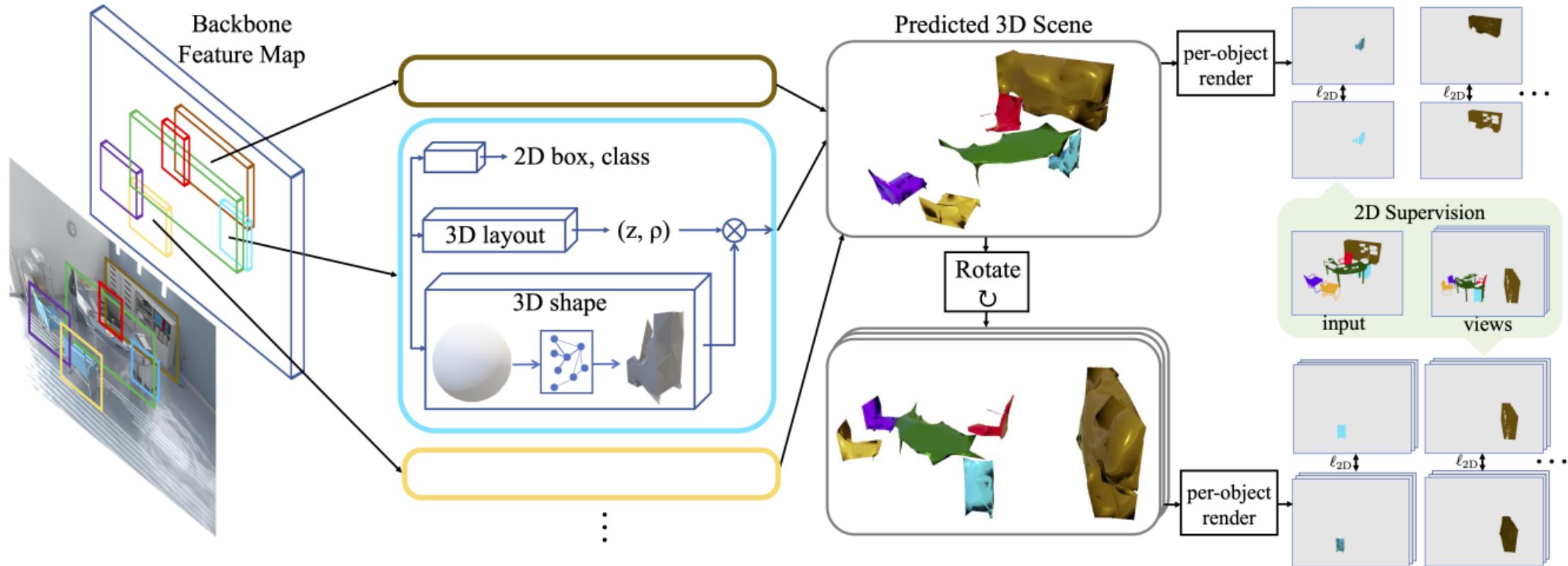


Figure 2. Our model takes as input an RGB image, detects all objects in 2D and predicts their 3D location and shape via *layout* and *shape* heads, respectively. The output is a scene composed of all detected 3D objects. During training, the scene is differentiably rendered from other views and compared with the 2D ground truth. We use no 3D shape or layout supervision.

# PyTorch

---

- **FineTuning from a pretrained model**

[https://pytorch.org/tutorials/intermediate/torchvision\\_tutorial.html](https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html)

```
● ● ●

import torchvision
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor

# load a model pre-trained on COCO
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(weights="DEFAULT")

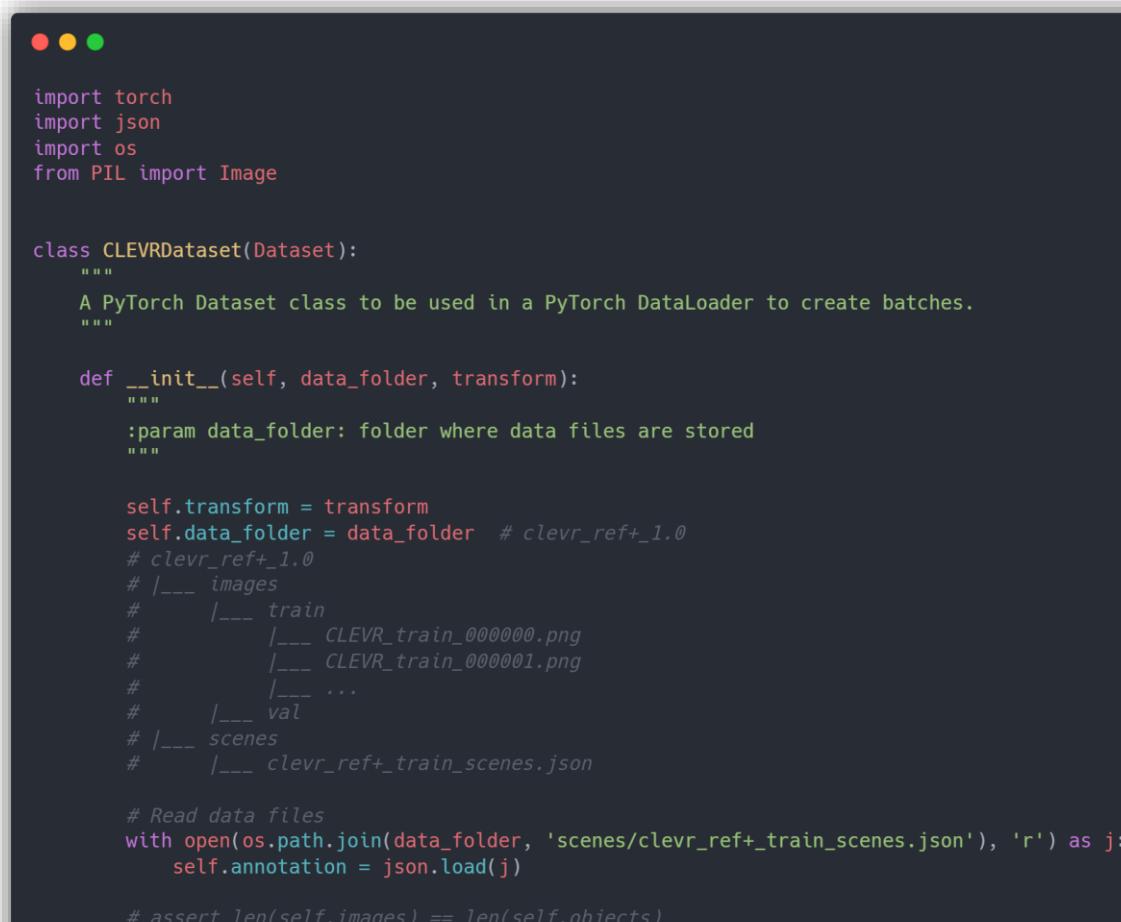
# replace the classifier with a new one, that has
# num_classes which is user-defined
num_classes = 2 # 1 class (person) + background
# get number of input features for the classifier
in_features = model.roi_heads.box_predictor.cls_score.in_features
# replace the pre-trained head with a new one
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)
```

# PyTorch

---

- Custom dataset `__init__`

[https://pytorch.org/tutorials/intermediate/torchvision\\_tutorial.html](https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html)



```
import torch
import json
import os
from PIL import Image

class CLEVRDataset(Dataset):
    """
    A PyTorch Dataset class to be used in a PyTorch DataLoader to create batches.
    """

    def __init__(self, data_folder, transform):
        """
        :param data_folder: folder where data files are stored
        """

        self.transform = transform
        self.data_folder = data_folder # clevr_ref+_1.0
        # clevr_ref+_1.0
        # /__ images
        #     /__ train
        #         /__ CLEVR_train_000000.png
        #         /__ CLEVR_train_000001.png
        #         /__ ...
        #     /__ val
        # /__ scenes
        #     /__ clevr_ref+_train_scenes.json

        # Read data files
        with open(os.path.join(data_folder, 'scenes/clevr_ref+_train_scenes.json'), 'r') as j:
            self.annotation = json.load(j)

        # assert len(self.images) == len(self.objects)
```

# PyTorch

---

- Custom dataset `__getitem__`

```
def __getitem__(self, idx):
    ##### images #####
    image = os.path.join(self.data_folder, 'images/train', self.annotation['scenes'][idx]
['image_filename'])
    image = Image.open(image, mode='r')
    image = image.convert('RGB')
    if self.transform:
        image = self.transform(image)

    ##### boxes #####
    boxes = list(dict(sorted(self.annotation['scenes'][idx]['obj_bbox'].items(), key=lambda x:
int(x[0])).values())) # (n_objects, 4)
    boxes = torch.FloatTensor(boxes)
    boxes[:, 2] = boxes[:, 0] + boxes[:, 2]
    boxes[:, 3] = boxes[:, 1] + boxes[:, 3]

    ##### labels #####
    labels = [self.to_label(label['shape']) for label in self.annotation['scenes'][idx]['objects']]
# (n_objects)
    labels = torch.tensor(labels, dtype=torch.int64)

    ##### masks #####
    obj_masks = list(dict(sorted(images['scenes'][idx]['obj_mask'].items(), key=lambda x:
int(x[0])).values())) # sorted masks

    masks = []
    for one_obj in obj_masks:
        gt_mask |= str_to_biimg(one_obj)
        masks.append(gt_mask)

    masks = torch.FloatTensor(masks)

    target = {
        'boxes': boxes,
        'labels': labels,
        'masks': masks,
    }

    return image, target
```

# PyTorch

---

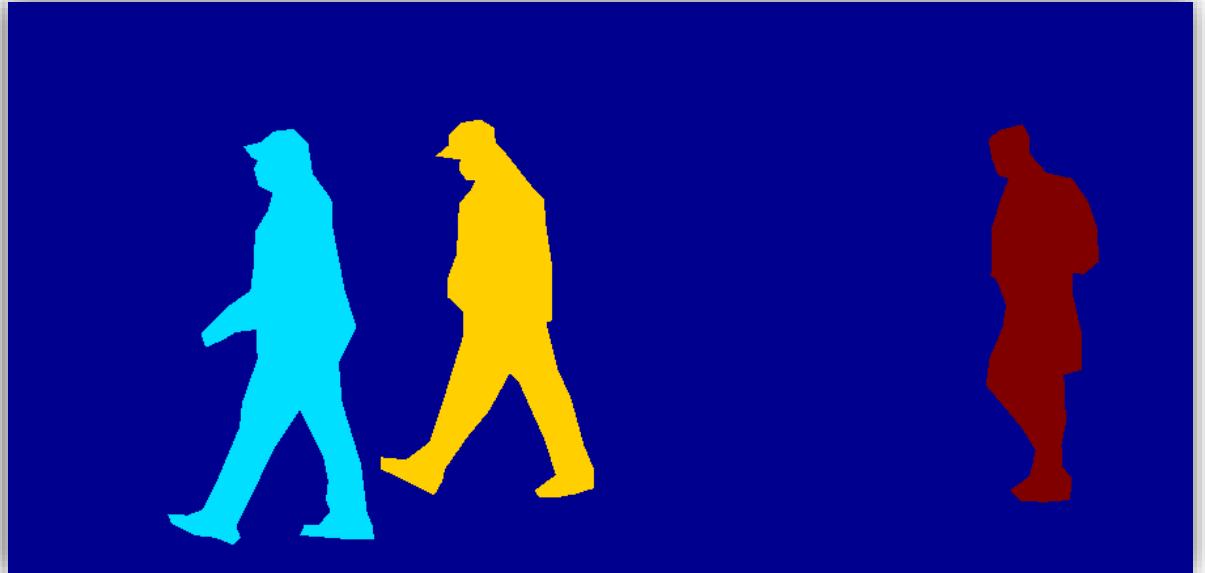
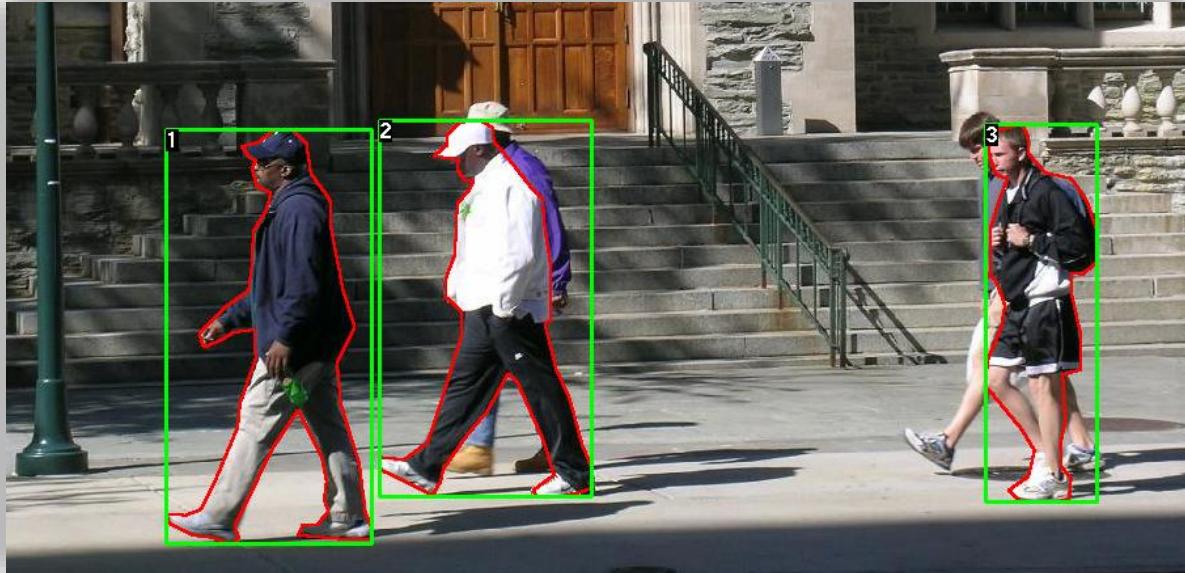
- Custom dataset `__len__`



```
def __len__(self):  
    return len(self.annotation['scenes'])
```

# Assignment #3

- FineTuning from a pretrained Faster RCNN
- You will use custom datasets ( <https://url.kr/u2hbln> )  다운로드
- You should implement **your own dataset class and dataloader.**



# Assignment #3

---

- **Conda create –n pytorch python=3.9**
- **Other libraries should be downloaded**
  - pip install tqdm
  - Pip install beautifulsoup4
  - ...

# Assignment #3

---

- You will use mask wearing dataset.
- Each image, there is an **annotation file (xml)**. You should close look at those annotation files and figure out what is **bounding boxes, lables, idx**.
- You should fill all the **TODOs** in the code. (Custum Dataset and Model)



```
# get bounding box coordinates for each mask
boxes = []
labels = []
#####
# TODO: for each image, get the bounding box coordinates and labels
# boxes and labels are lists
# boxes: [[xmin, ymin, xmax, ymax], ...]
# labels: [label, ...]
# label: 1 for with_mask, 2 for mask_weared_incorrect, 3 for without_mask
#####
pass
```

# Assignment #3

- Due Date: 12/4
- If you put appropriate codes, you will see those outputs.
- Lectures, Exercises, and Assignments will be uploaded in Github ([https://github.com/ONground-Korea/2023-AIKU\\_DeepLearning-Bootcamp/tree/main/Assignments/3.%20Object%20Detection%20and%20Segmentation](https://github.com/ONground-Korea/2023-AIKU_DeepLearning-Bootcamp/tree/main/Assignments/3.%20Object%20Detection%20and%20Segmentation) )



---

**Thank you!**  
**Q & A**