Object Detection

Jisang Han

onground@korea.ac.kr

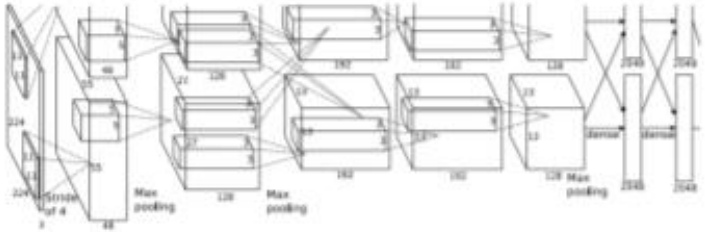
KUGODS

Department of Computer Science and Engineering, Korea University

Korea University GOogle Developer Students
KUGODS

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 x W?

Consider a box of size h x 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**

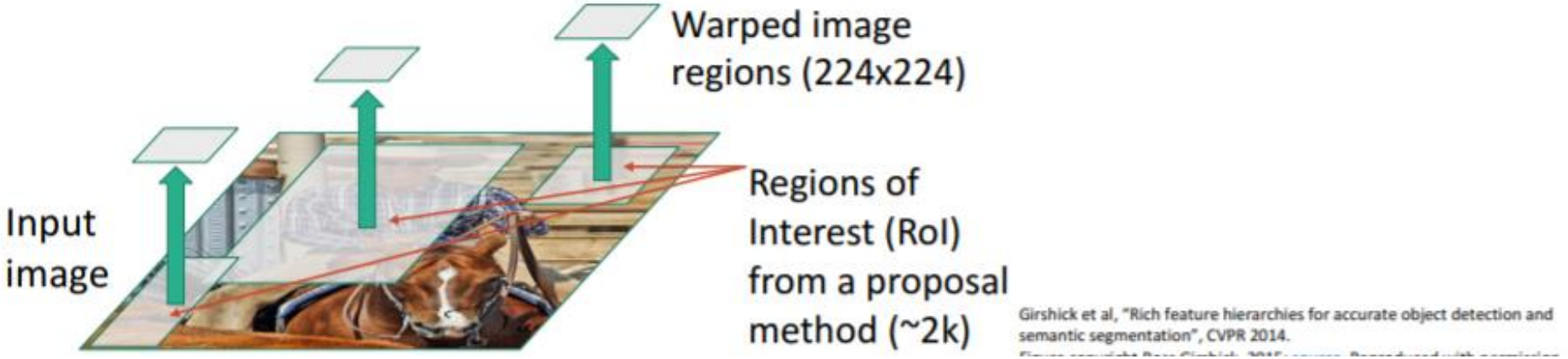
R-CNN: Region-Based CNN

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.

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

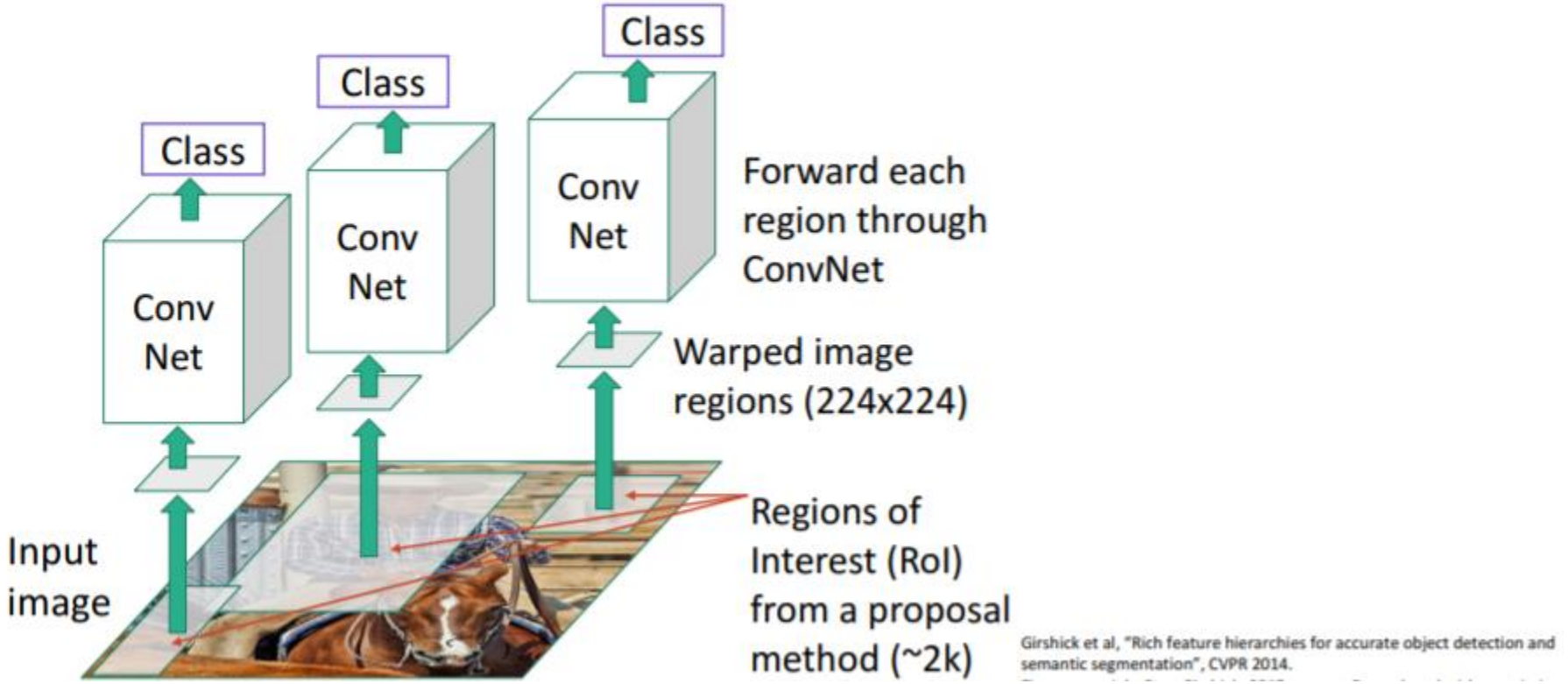
R-CNN: Region-Based CNN



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

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

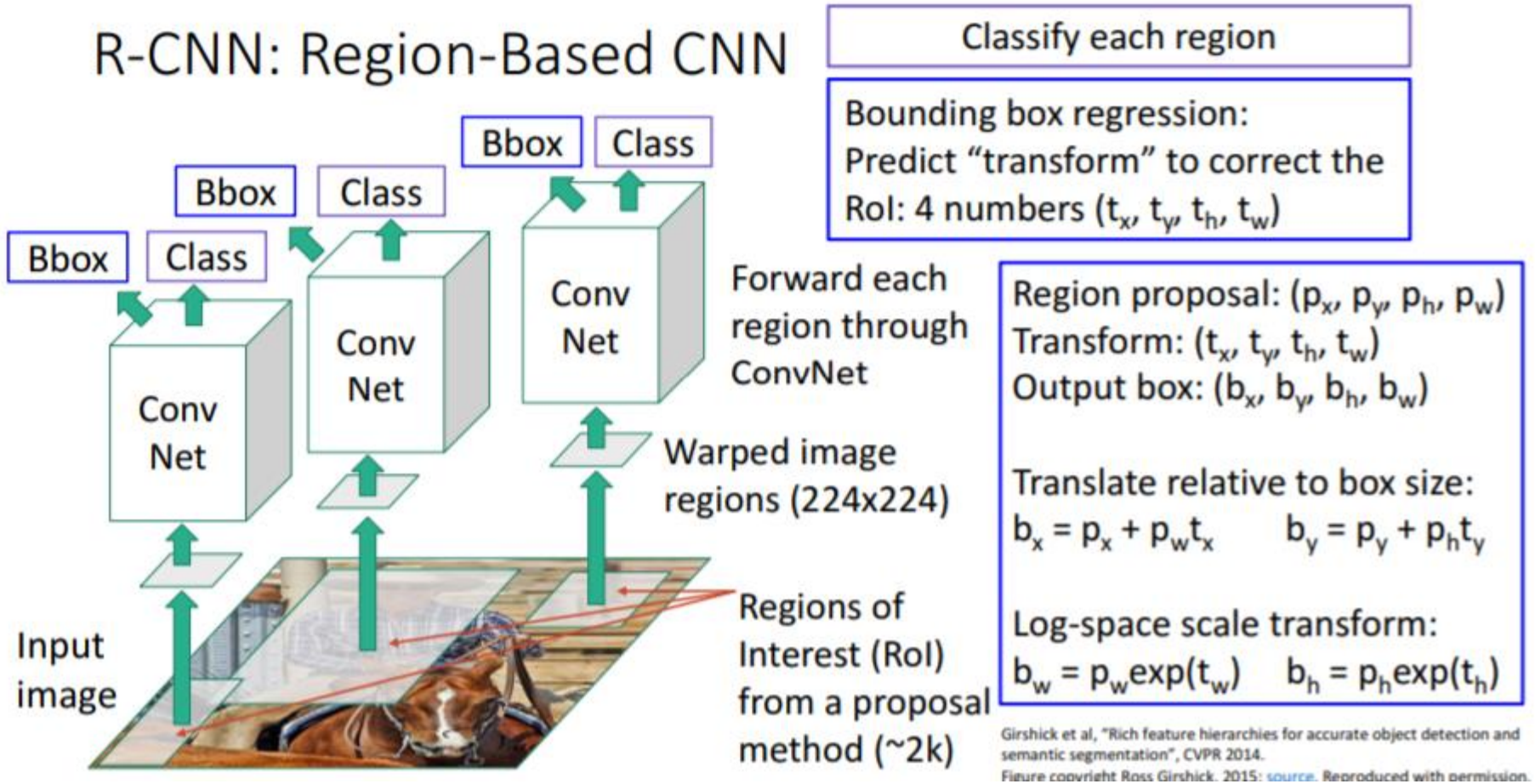
R-CNN: Region-Based CNN



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** $(p_x,\ p_y,\ p_h,\ p_w)$

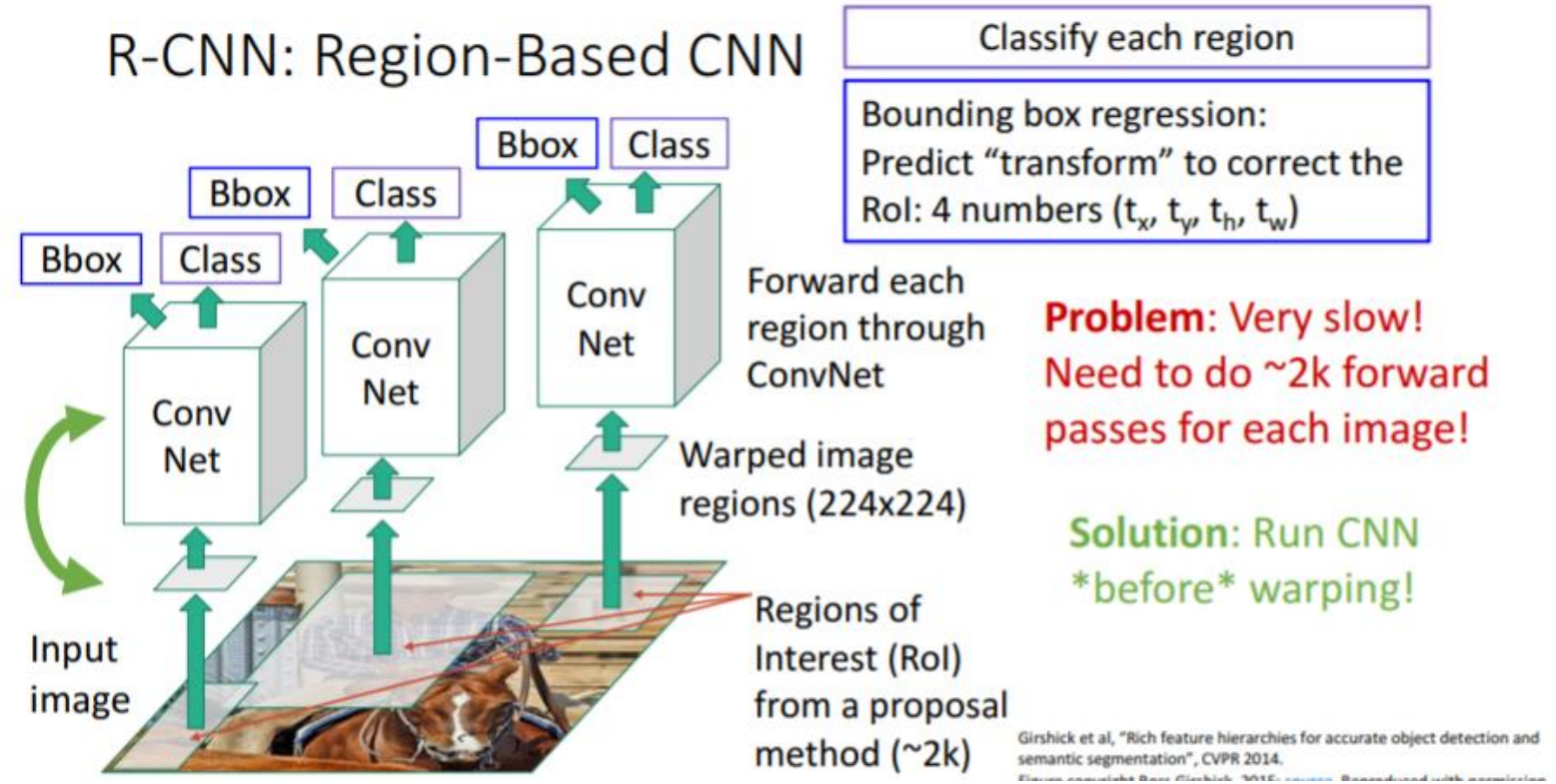
$$b_x = p_x + p_w t_w$$

$$b_y = p_y + p_h t_y$$

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

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

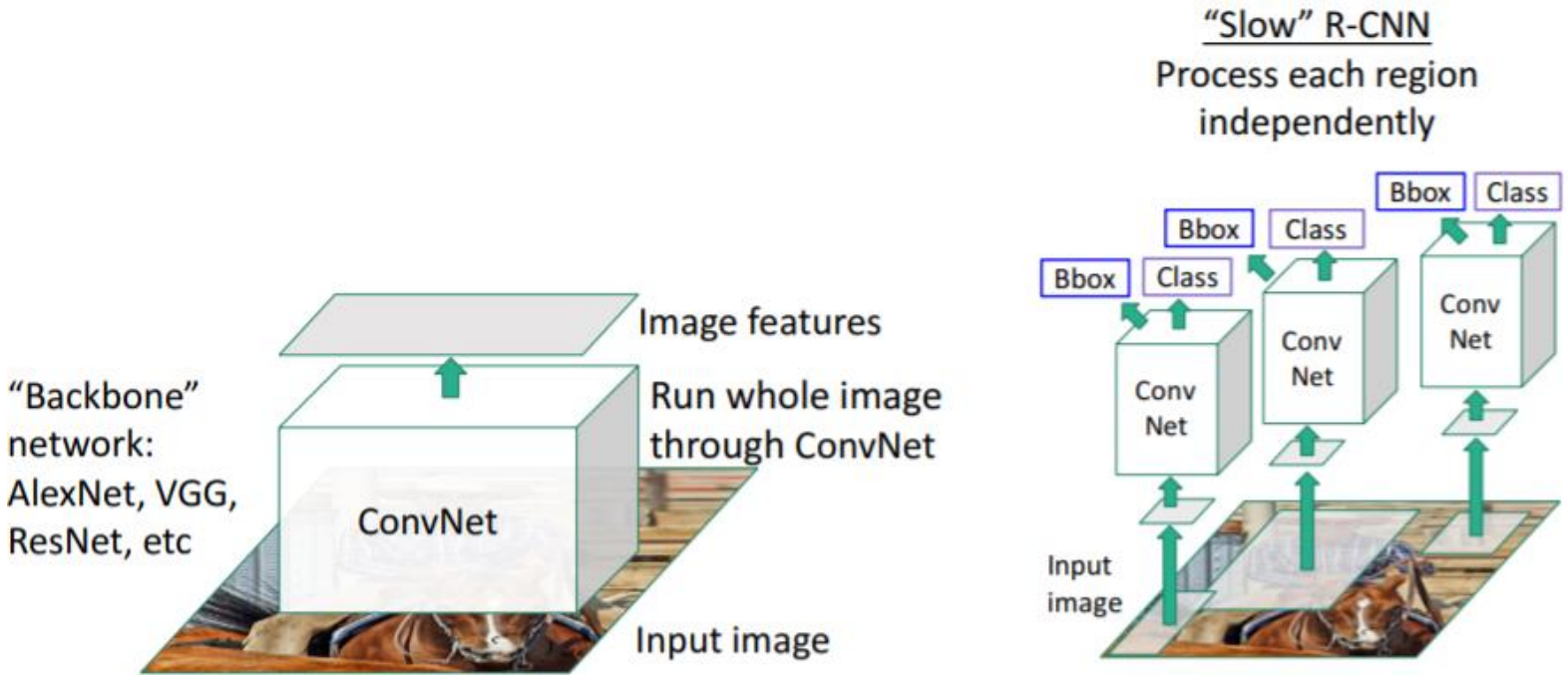
Fast R-CNN



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

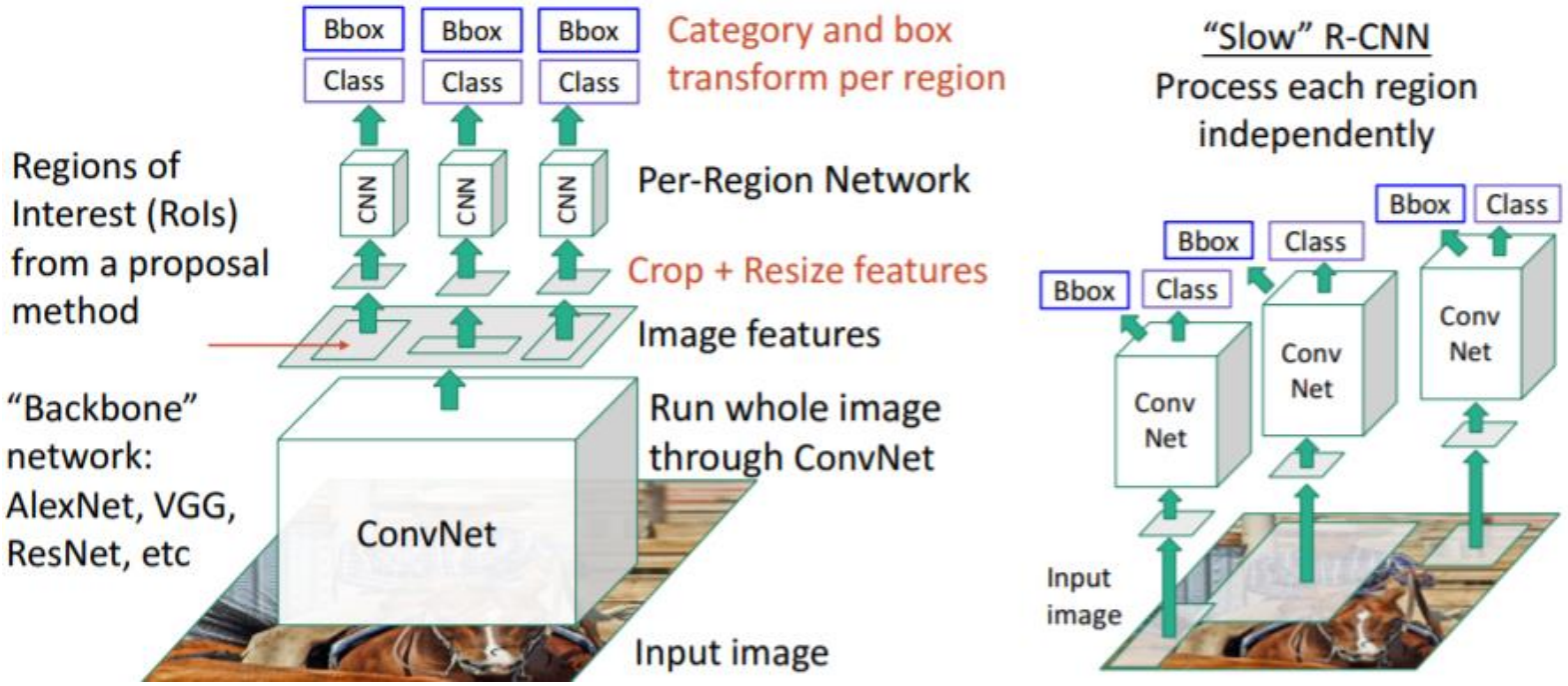
To solve this problem, they **changed the order of CNN and warping.**

Fast R-CNN



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**.

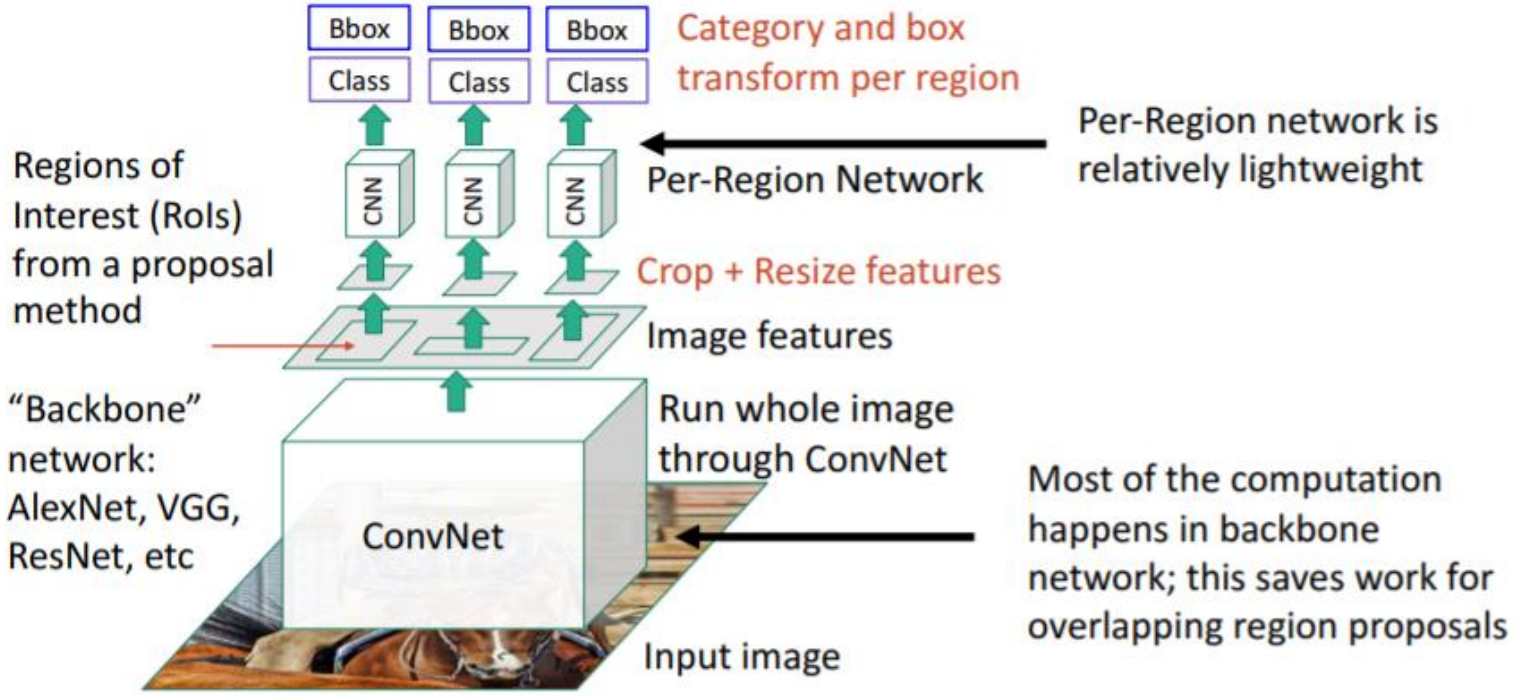
Fast R-CNN



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.

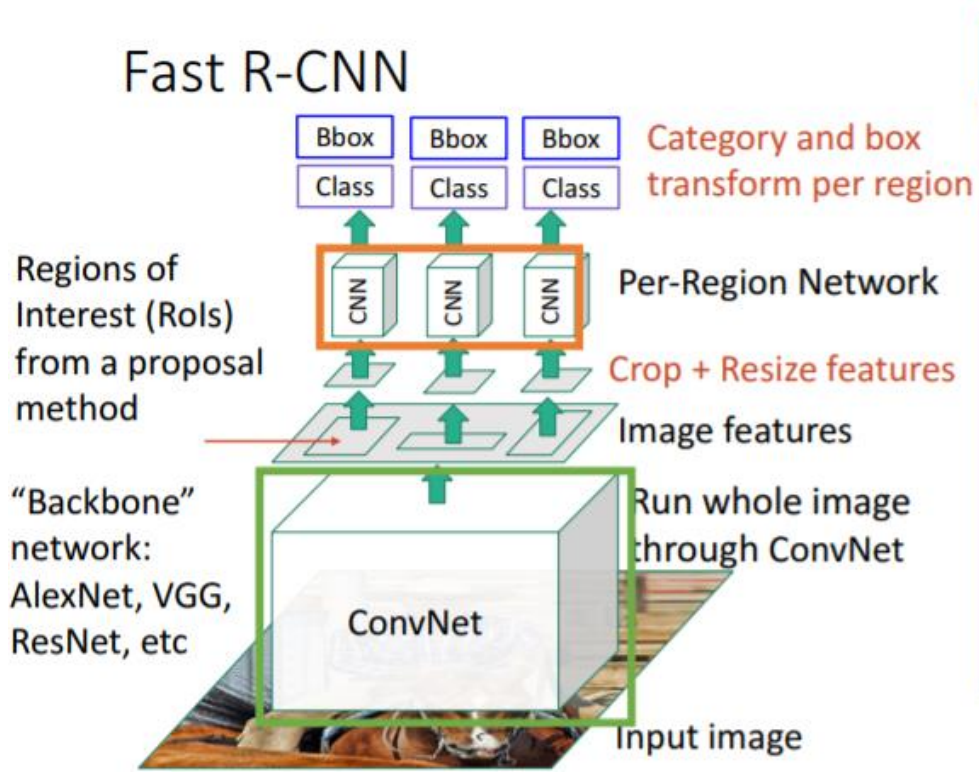
Fast R-CNN

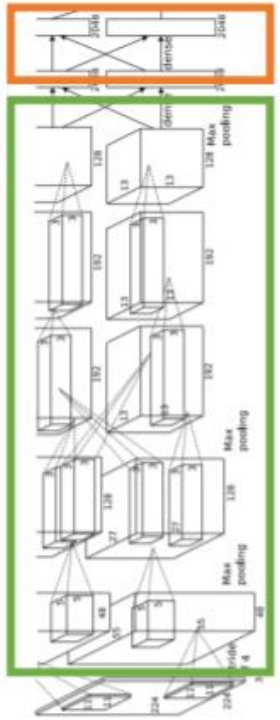


The reason **why this method is fast** is that **most of the calculations take place on the Backbone network** below, which **ends with a single calculation** without having to be repeated for overlapping region proposals.

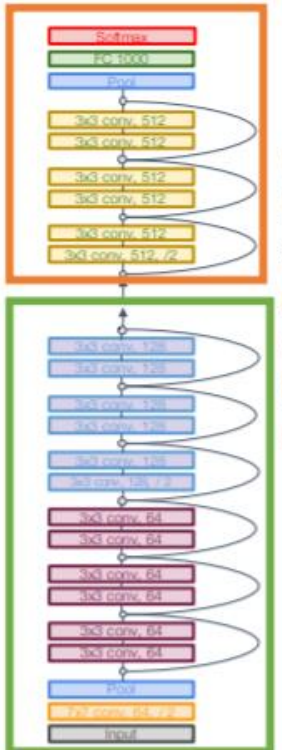
And the **CNN above**, per-region network, is **relatively very light and small**.

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



Example:
For ResNet, last stage is used as per-region network; the rest of the network is used as backbone

Fast R-CNN: Test Time

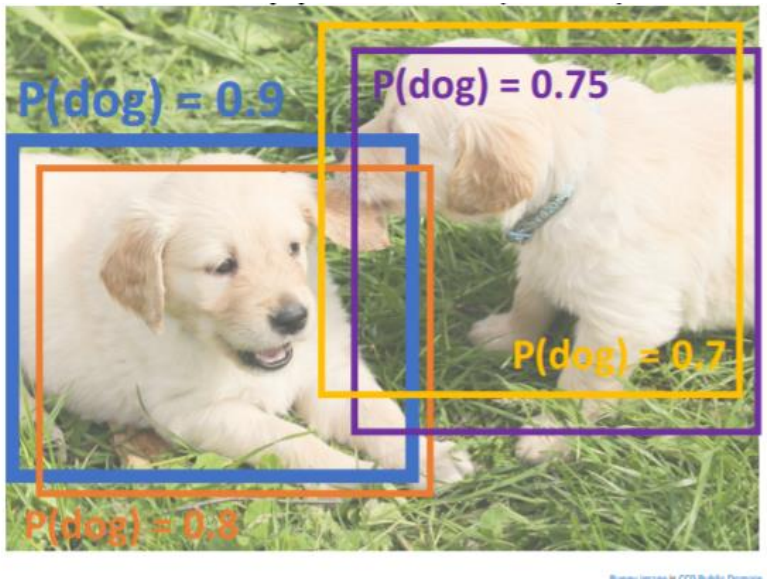
Non-Max Suppression (NMS)

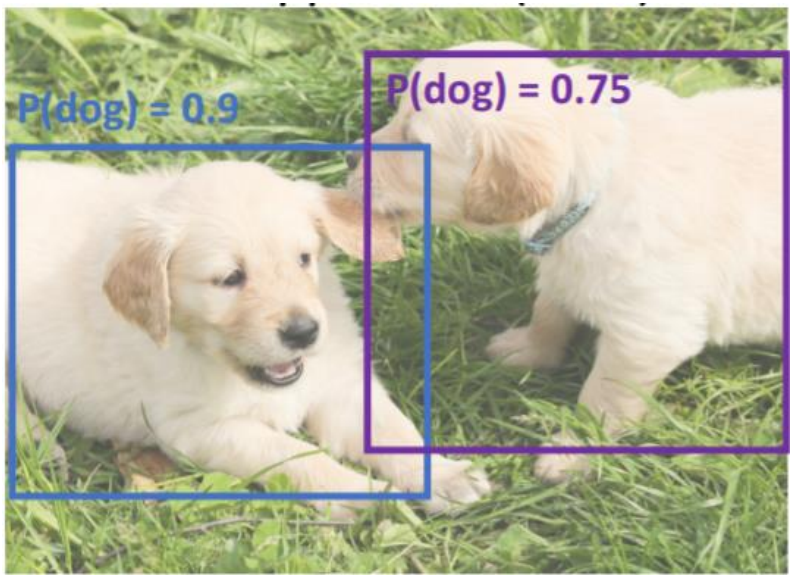
Problem: Object detectors often output many overlapping detections:

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

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

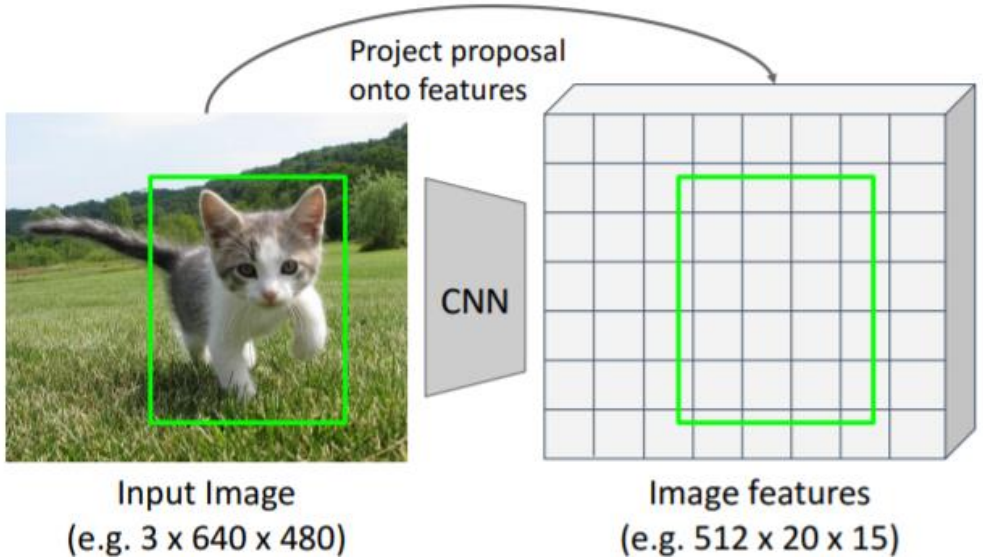
IoU(,) = **0.78**
IoU(,) = 0.05
IoU(,) = 0.07

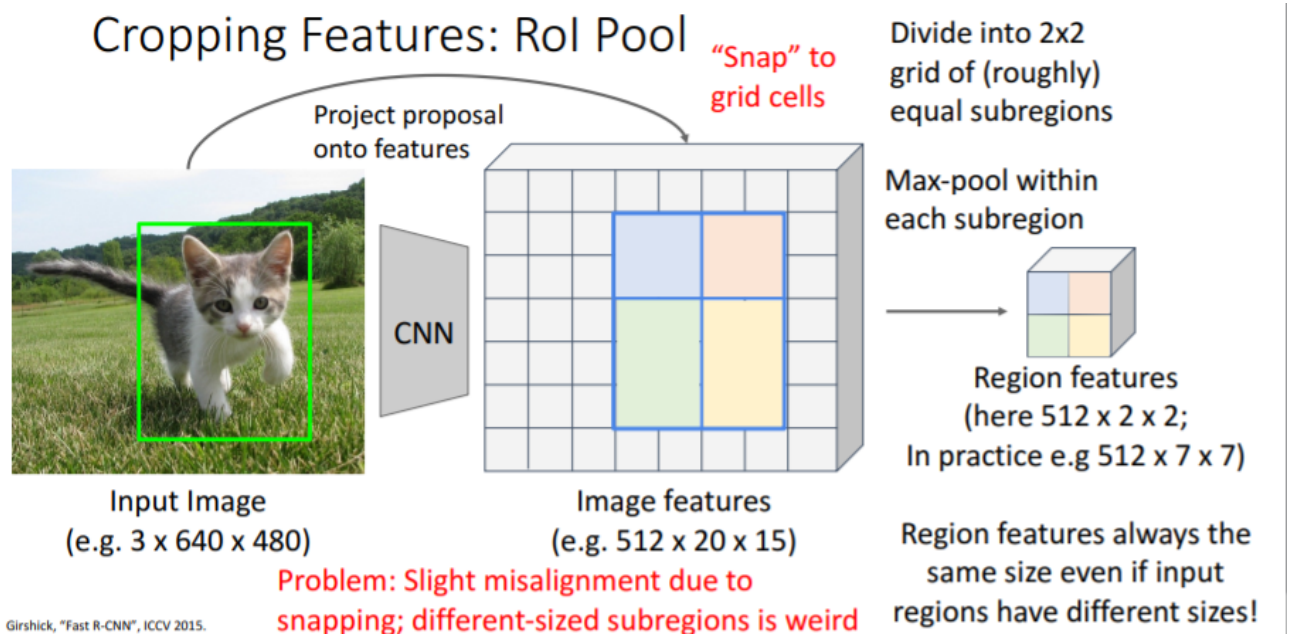




1. We get the image classification score for each bounding box. **Select next highest-scoring box**
2. For every other boxes, if IoU > threshold -> Eliminate
3. If any boxes remain, GOTO 1.

Fast R-CNN: RoI Pooling



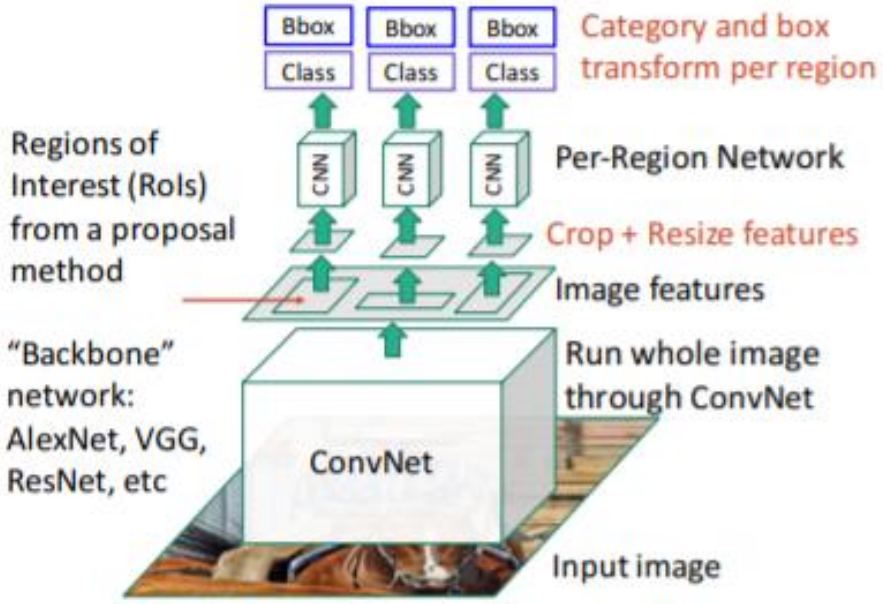


Problems

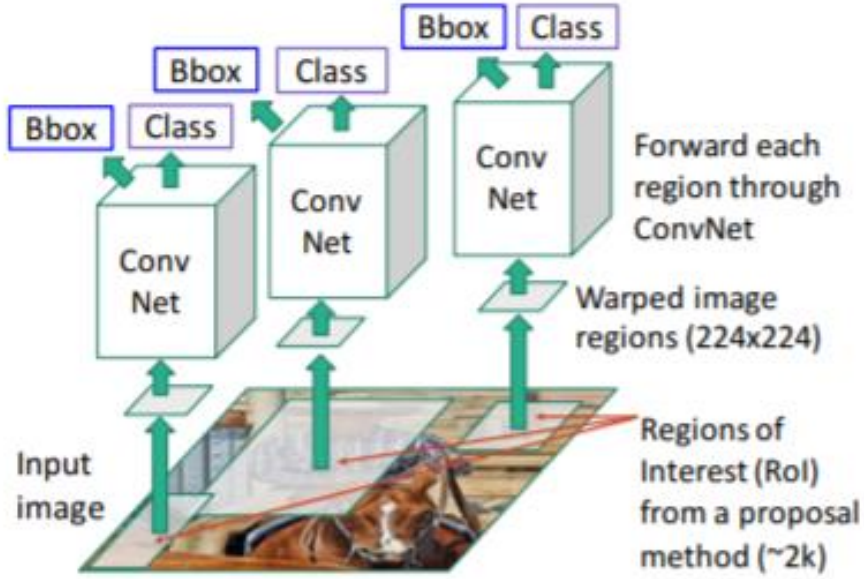
- Snapping cause misalignment
- Sub-regions are different-sized
- Cannot backpropagate to the original bounding box coordinates (snapped)

R-CNN vs Fast R-CNN

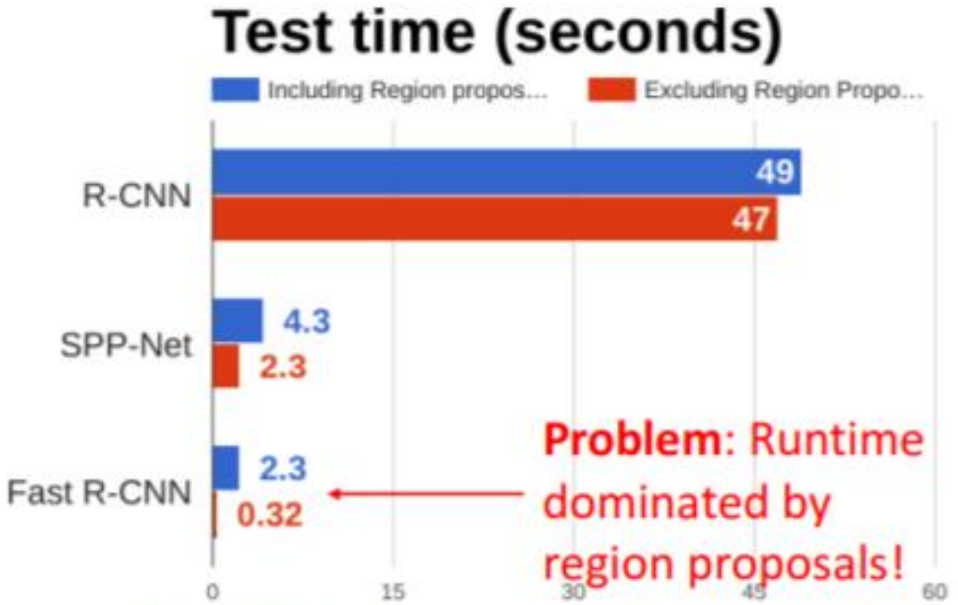
Fast R-CNN: Apply differentiable cropping to shared image features



"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

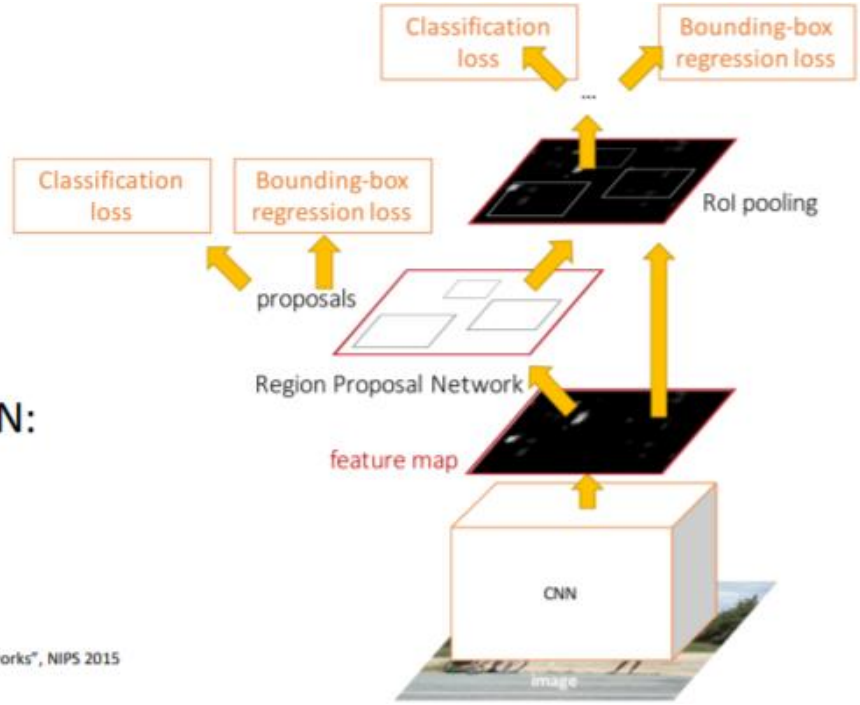
Recall: Region proposals computed by heuristic "Selective Search" algorithm on CPU -- let's learn them with a CNN instead!

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

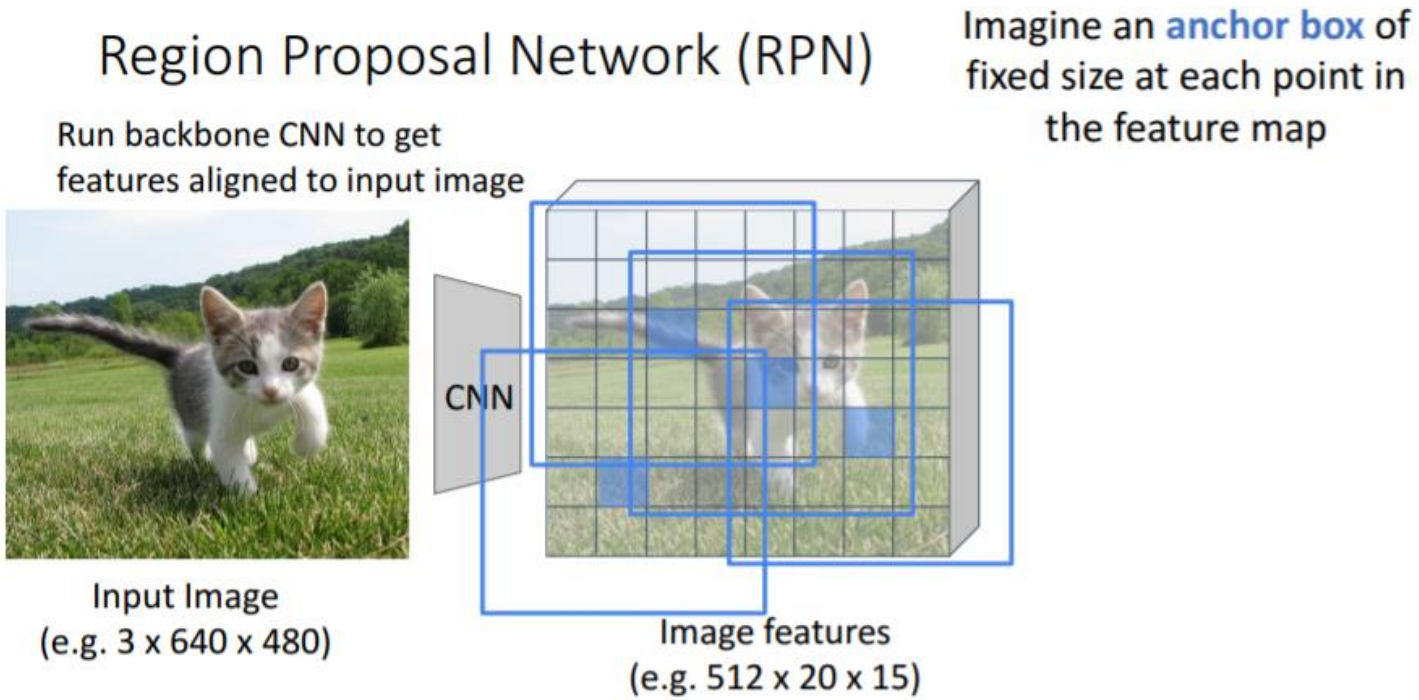
Otherwise same as Fast R-CNN: Crop features for each proposal, classify each one



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

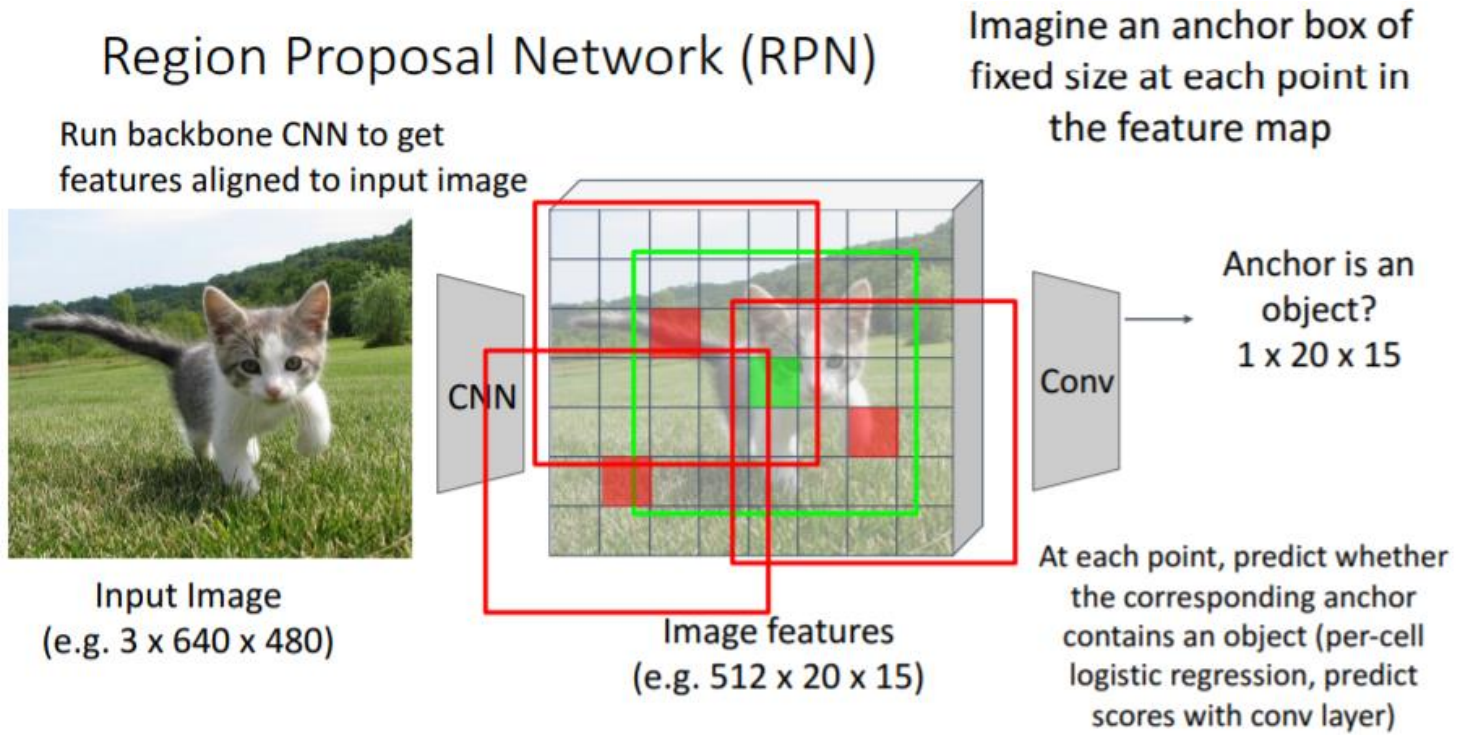
Output feature map passes through **the RPN to predict the region proposals**, and with the extracted region proposals, **all process the rest of the process in the same way as in Fast R-CNN**.

Region Proposal Network (RPN)



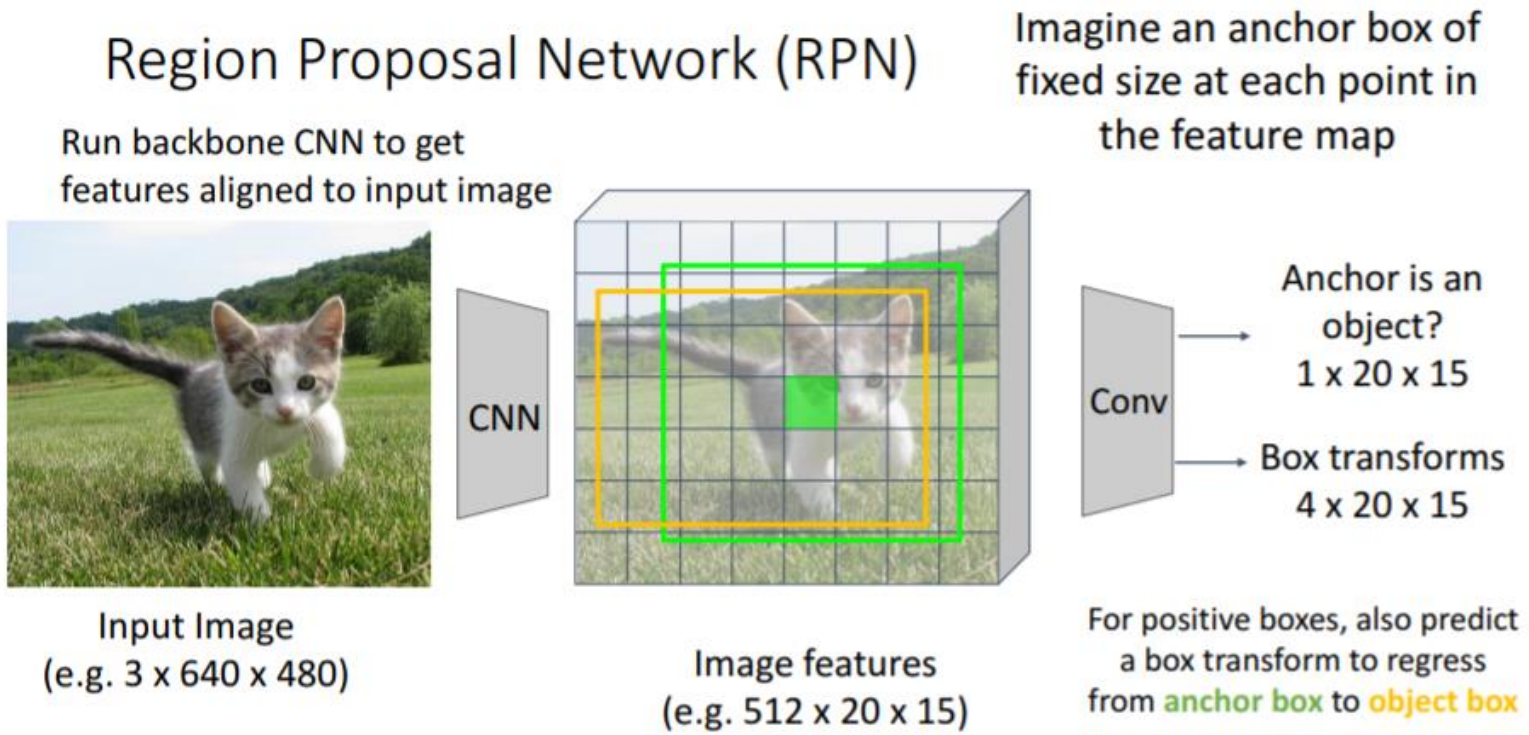
Fixed size of anchor box at each point in the feature map

Region Proposal Network (RPN)



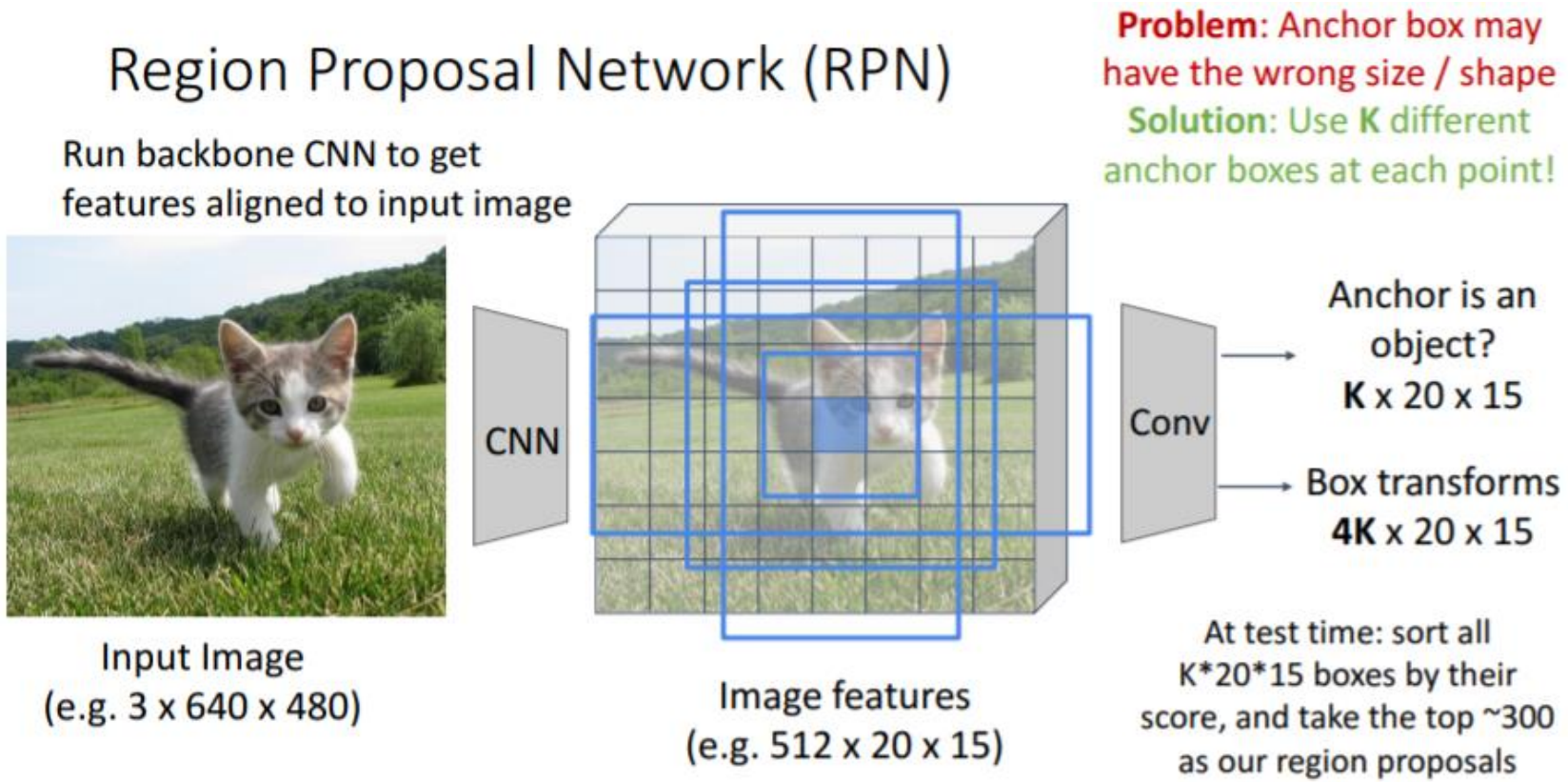
In each anchor box, **binary classify whether or not these anchor boxes contain objects through CNN**.

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)



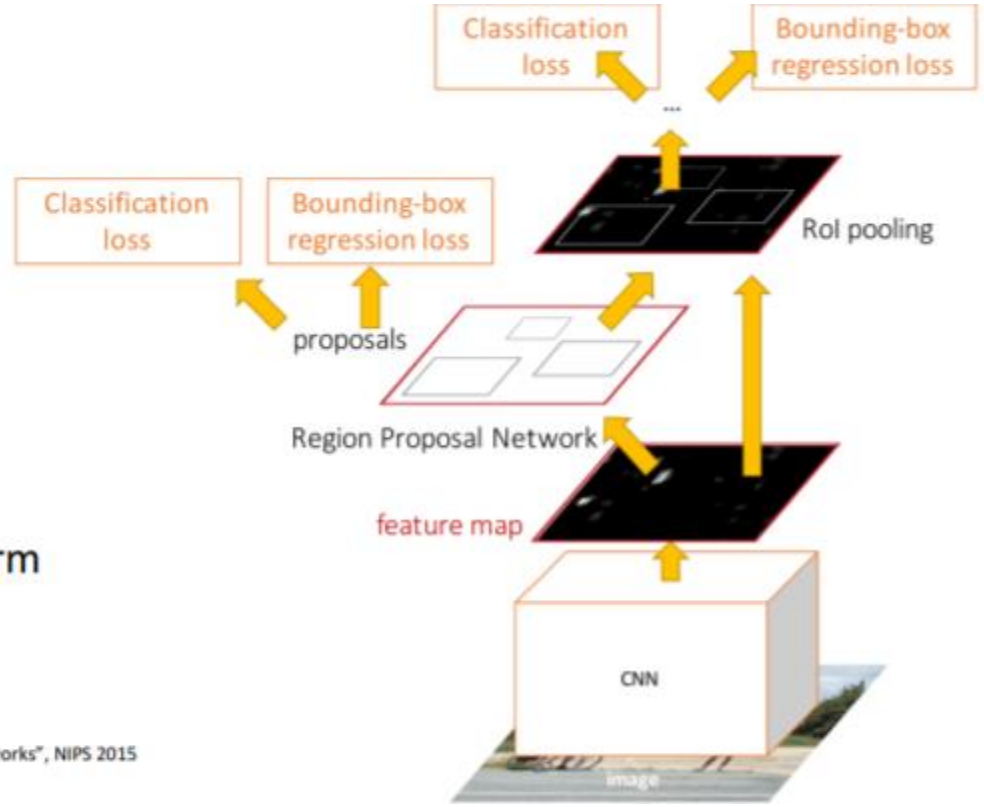
Using only one fixed size anchor box is **not enough to fit well on all kinds of objects**.

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

Faster R-CNN: 4 Losses

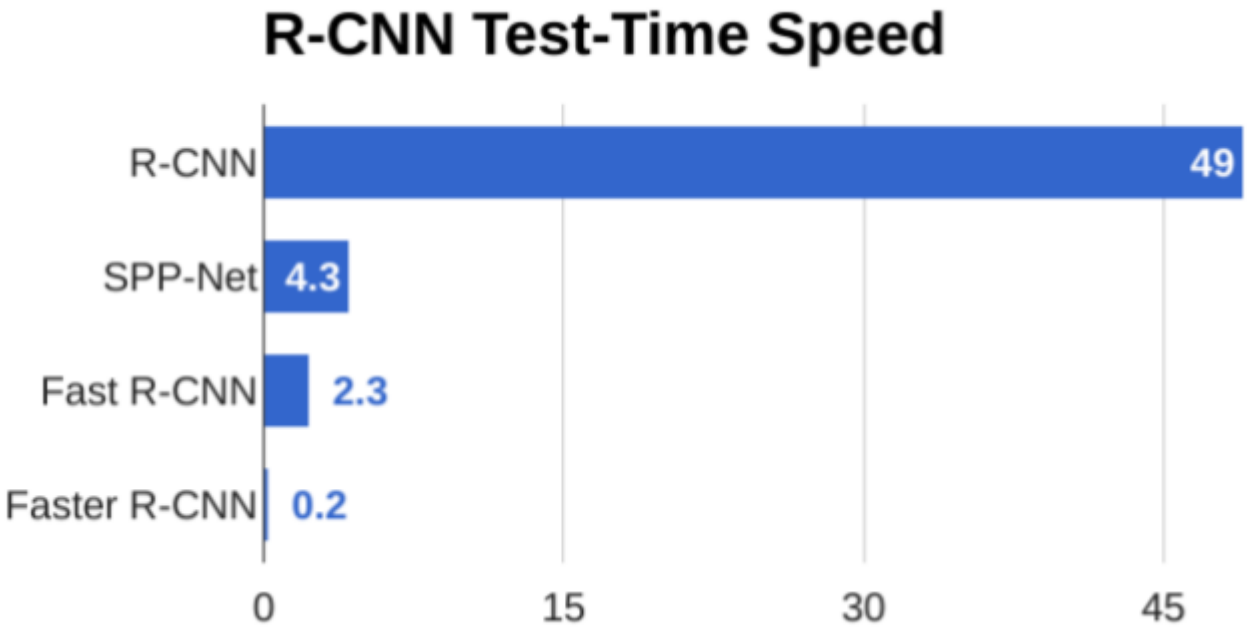
Jointly train with 4 losses:

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



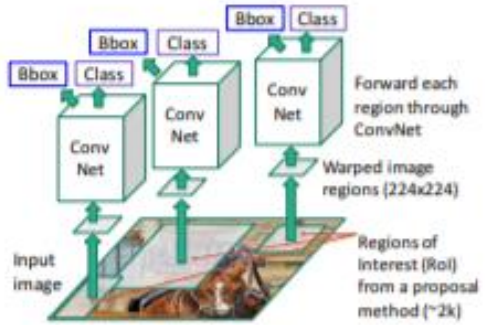
Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015
Figure copyright 2015, Ross Girshick; reproduced with permission

Faster R-CNN

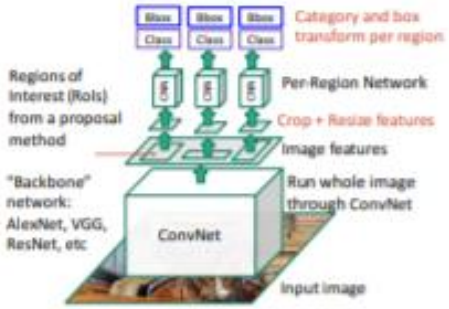

R-CNN Test-Time Speed
R-CNN
SPP-Net
Fast R-CNN
Faster R-CNN
49
4.3
2.3
0.2
0
15
30
45

Summary

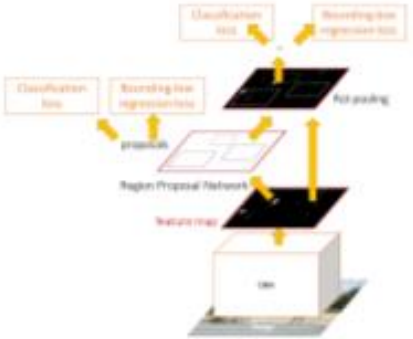
"Slow" R-CNN: Run CNN independently for each region



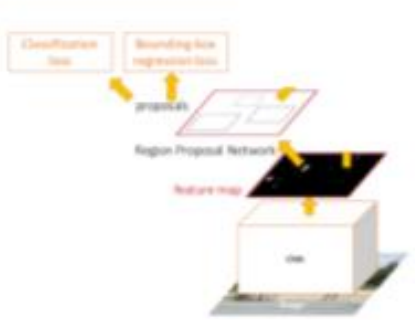
Fast R-CNN: Apply differentiable cropping to shared image features



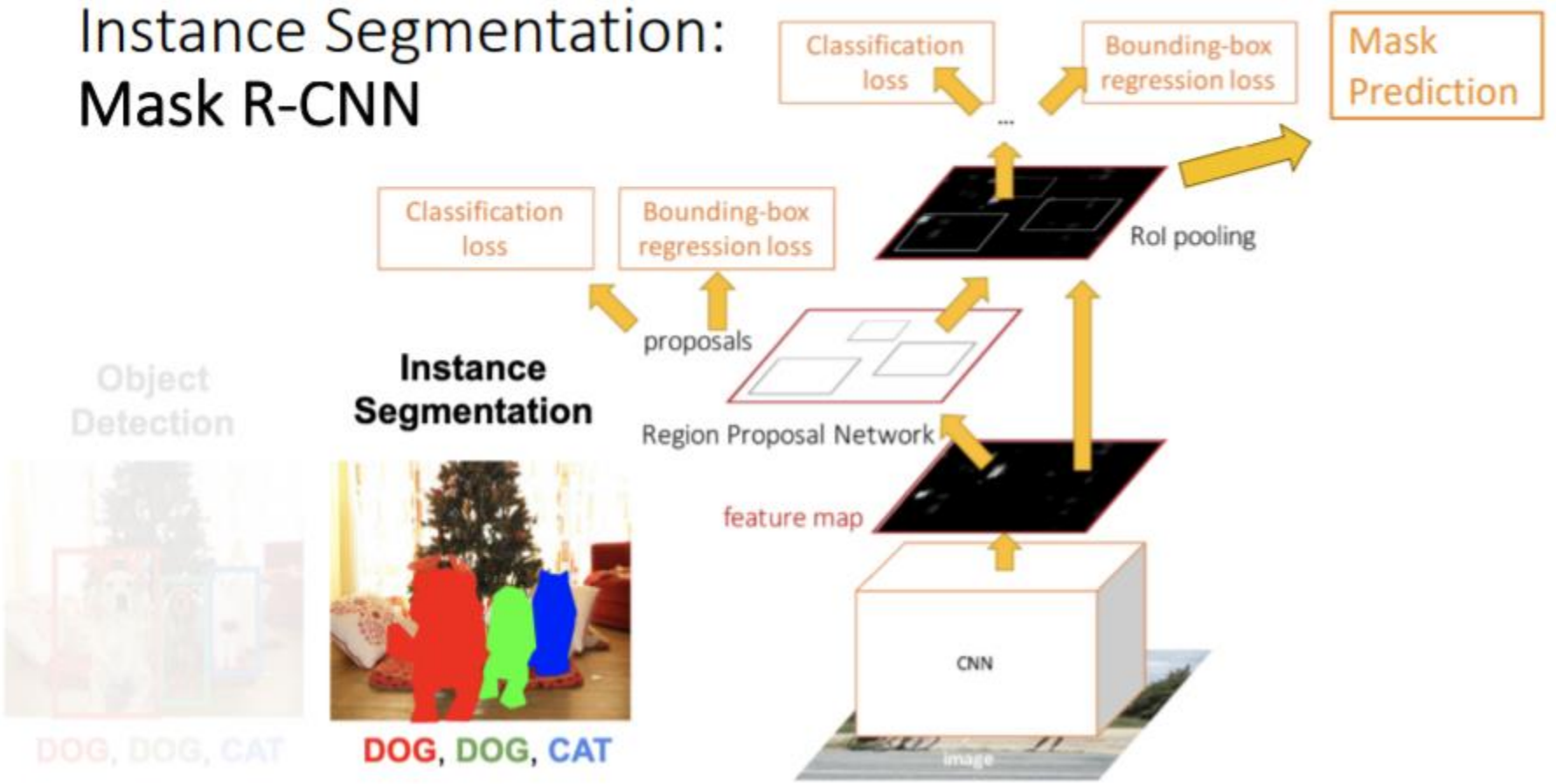
Faster R-CNN: Compute proposals with CNN



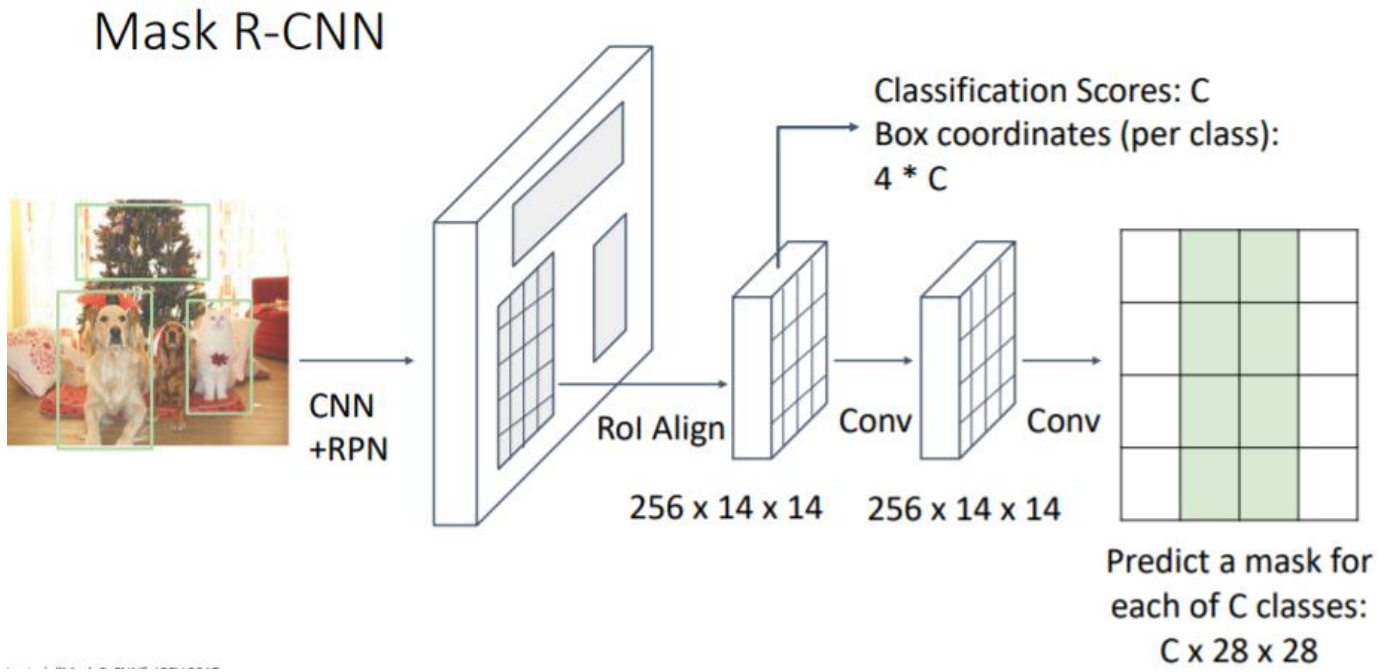
Single-Stage: Fully convolutional detector

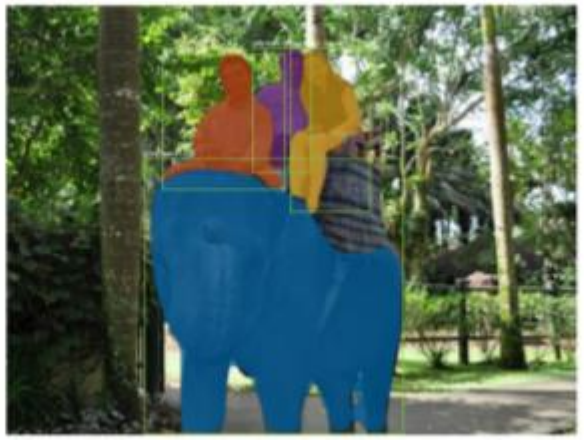


Mask R-CNN

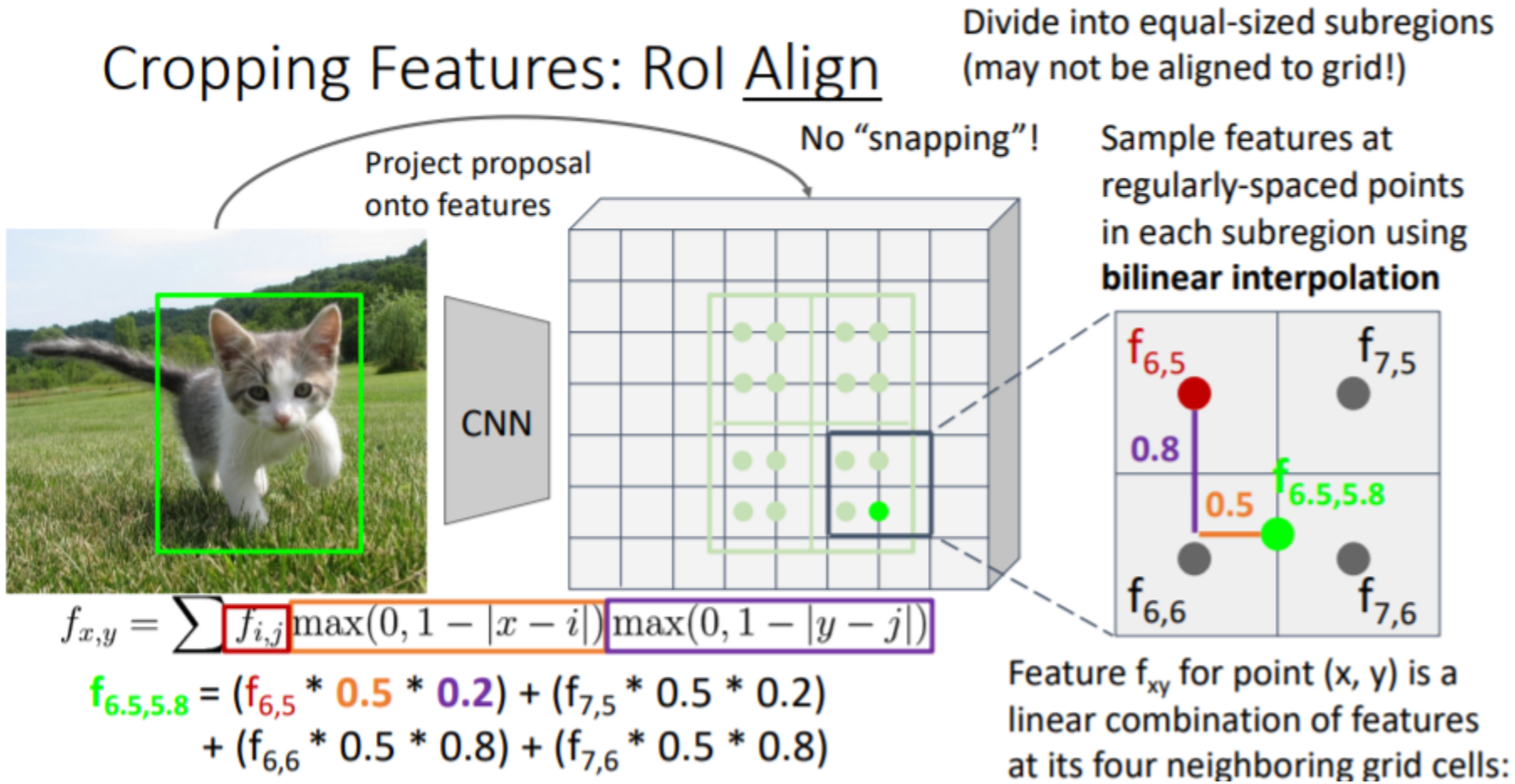

Instance Segmentation:
Mask R-CNN
Classification loss
Bounding-box regression loss
Mask Prediction
...
RoI pooling
Classification loss
Bounding-box regression loss
proposals
Region Proposal Network
feature map
CNN
image
Object Detection
Instance Segmentation
DOG, DOG, CAT
DOG, DOG, CAT

Mask R-CNN

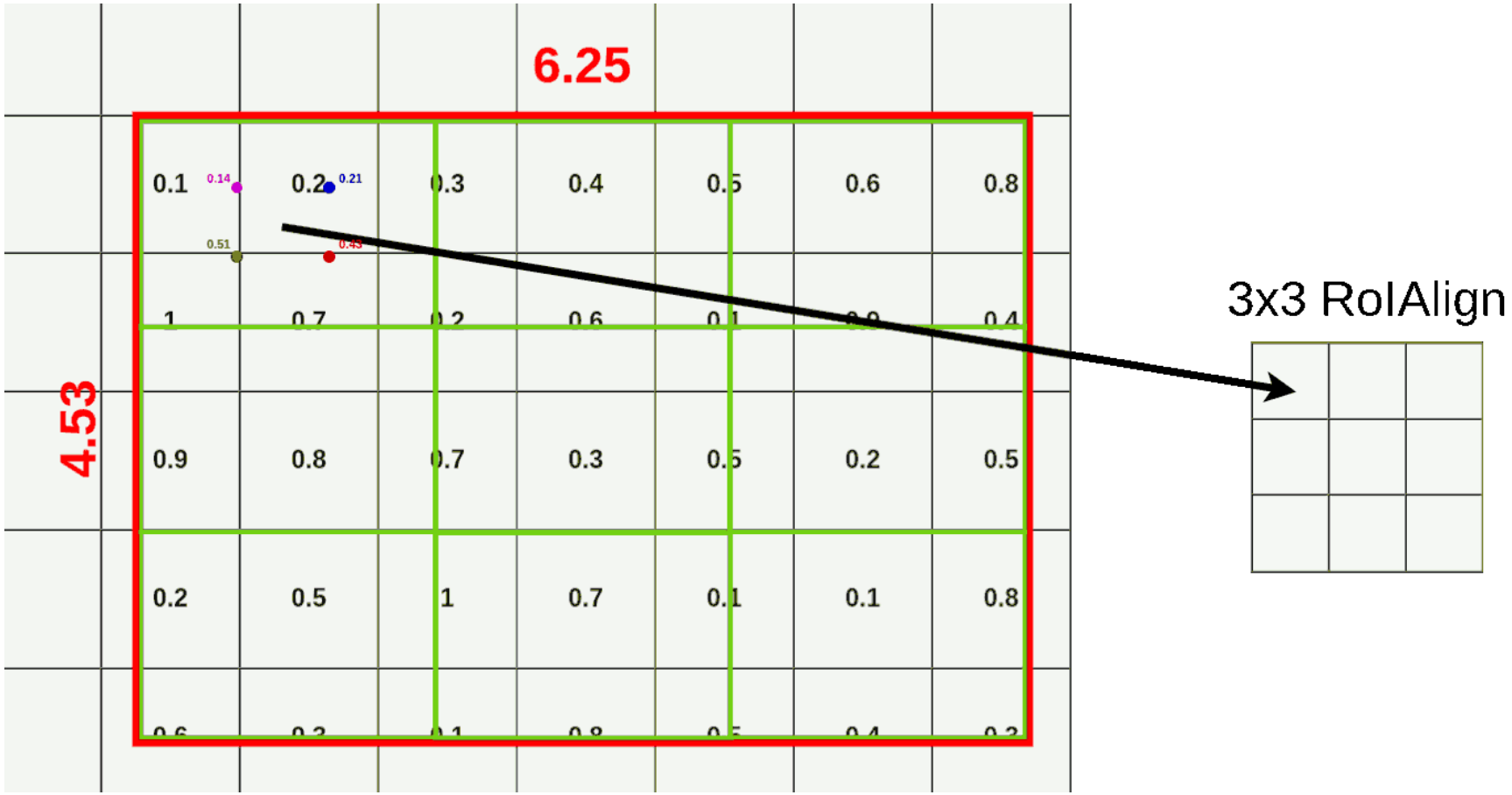

Mask R-CNN
Classification Scores: C
Box coordinates (per class):
4 * C
CNN
+RPN
RoI Align
Conv
Conv
256 x 14 x 14
256 x 14 x 14
Predict a mask for
each of C classes:
C x 28 x 28



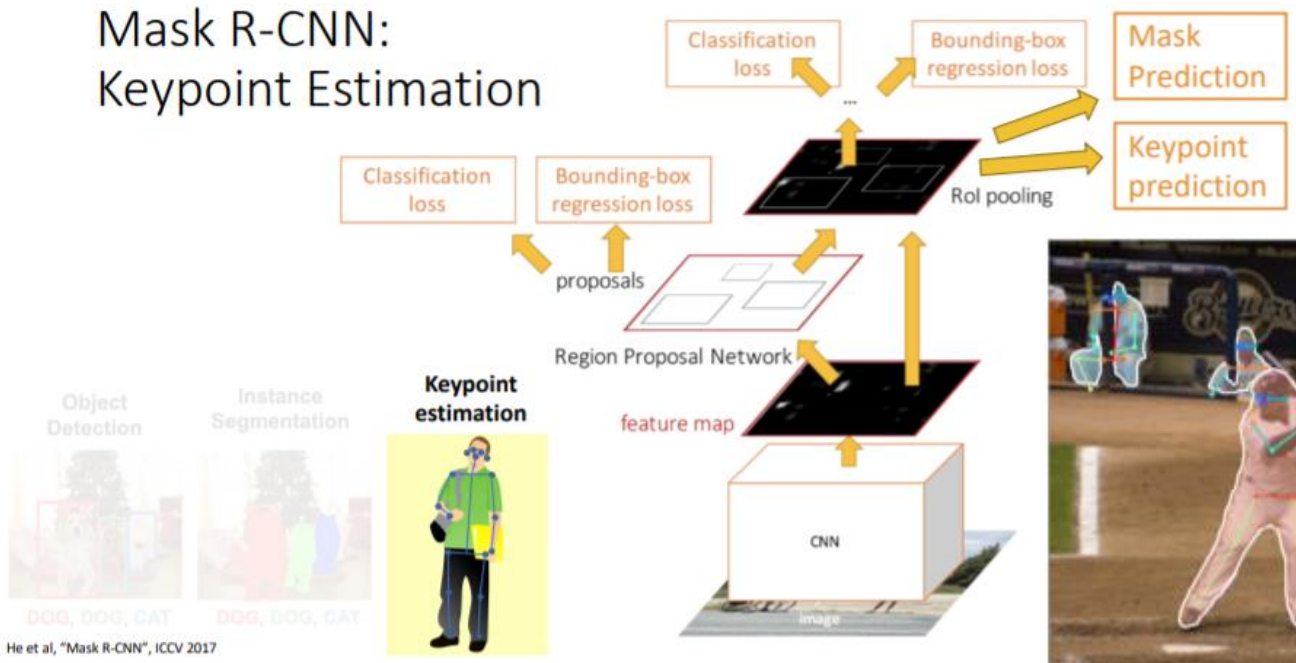
Mask R-CNN: RoI Align



Mask R-CNN: RoI Align

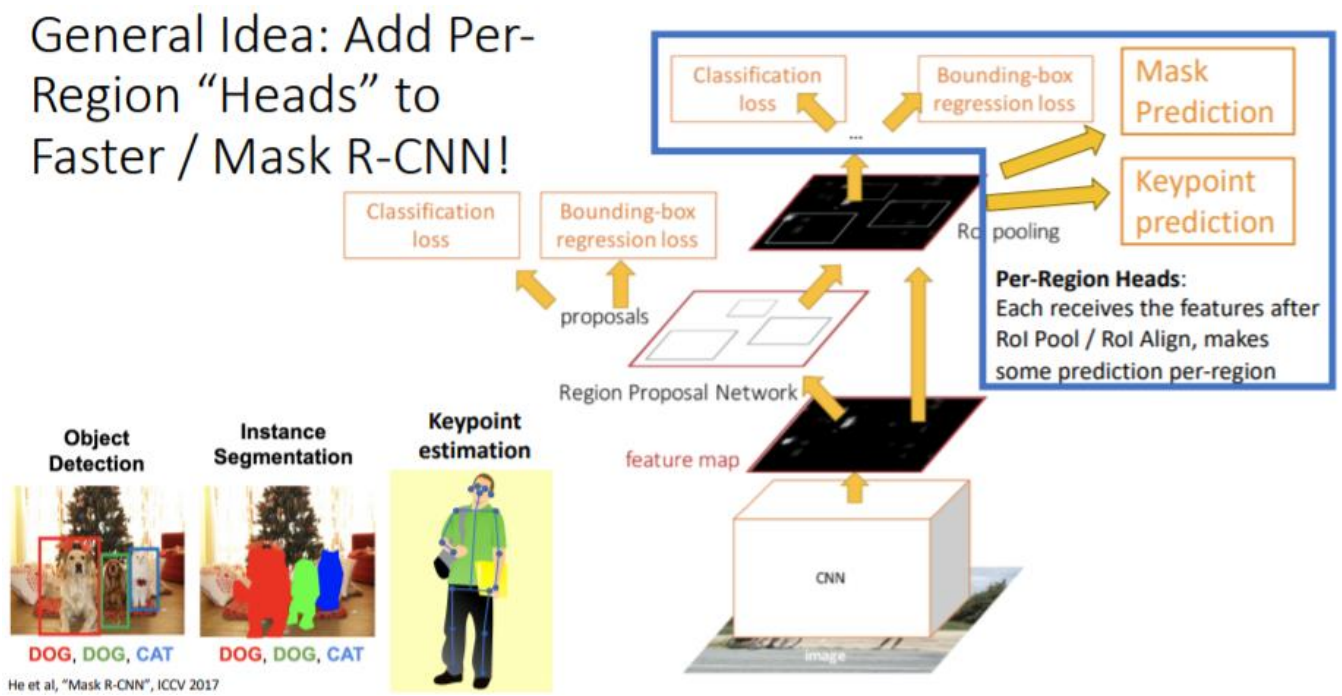


Mask R-CNN: KeyPoints



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

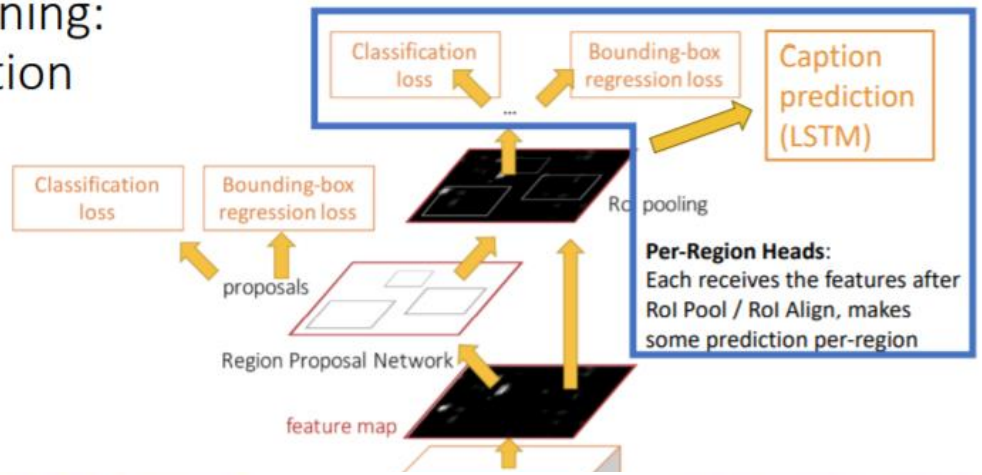
General Idea: Add PerRegion “Heads” to Faster / Mask R-CNN!

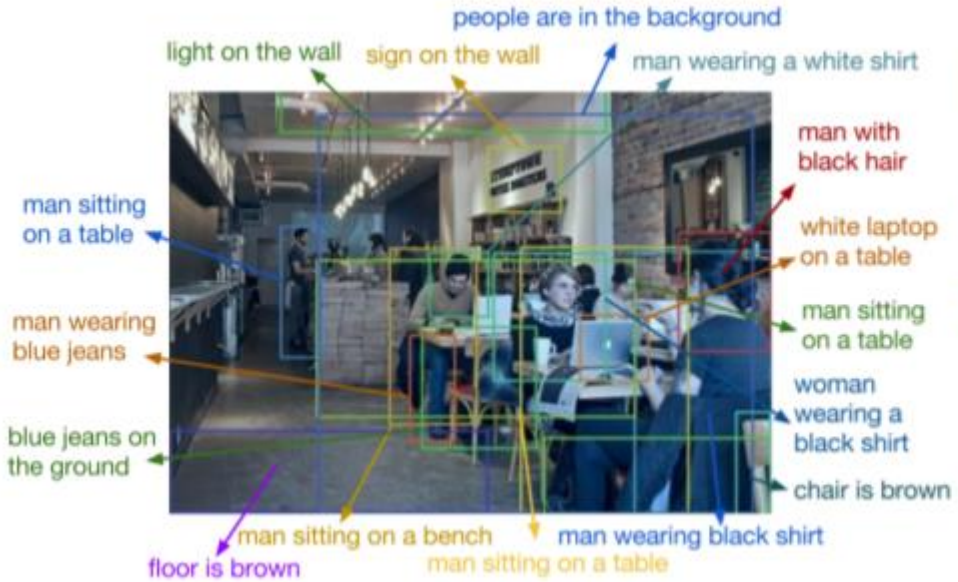


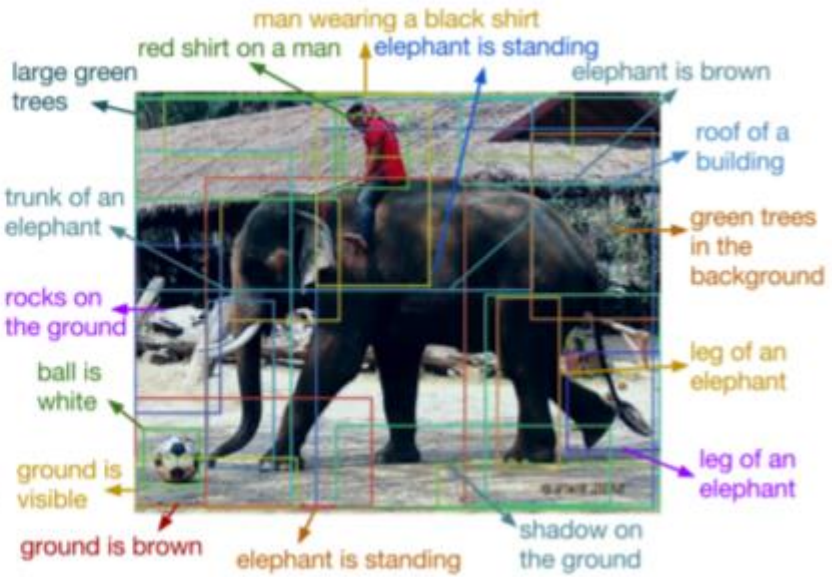
It can be applied to a **new computer vision task** using the method of **adding branches**.

Dense Captioning

Dense Captioning:
Predict a caption
per region!

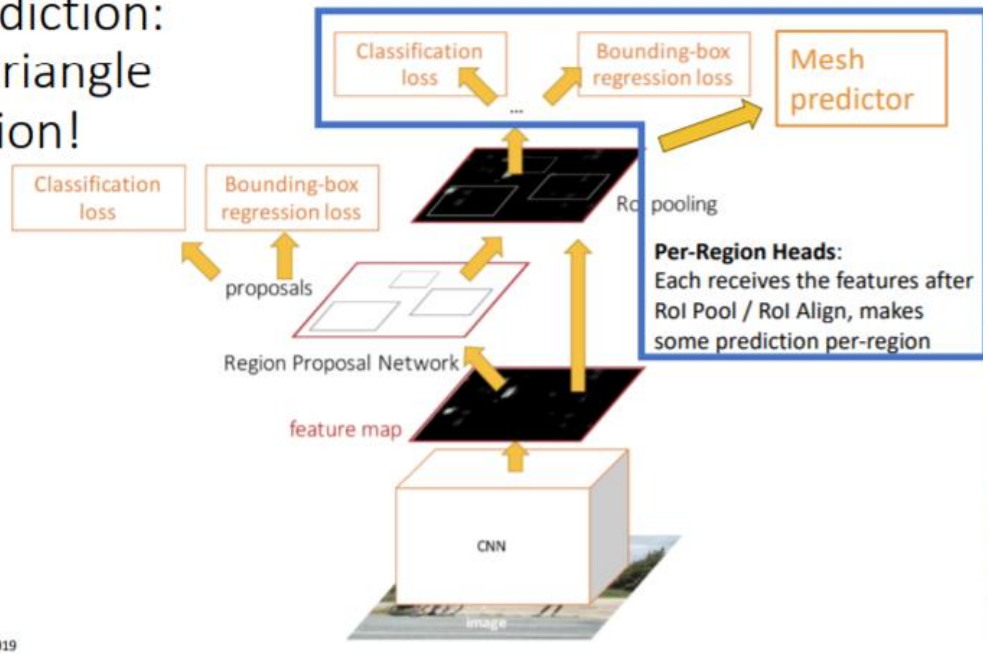






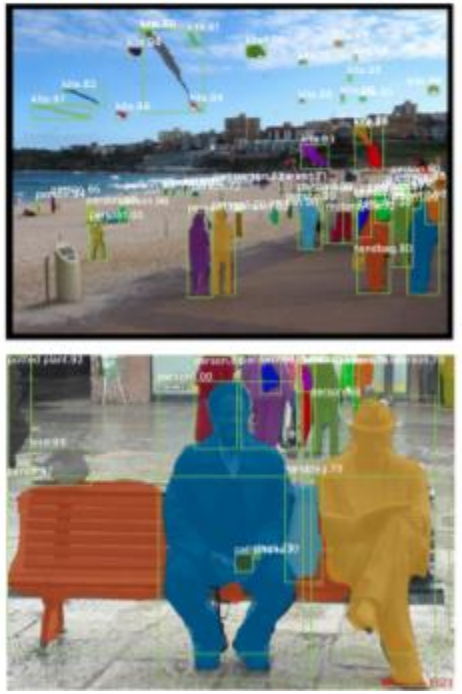
Mesh R-CNN

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

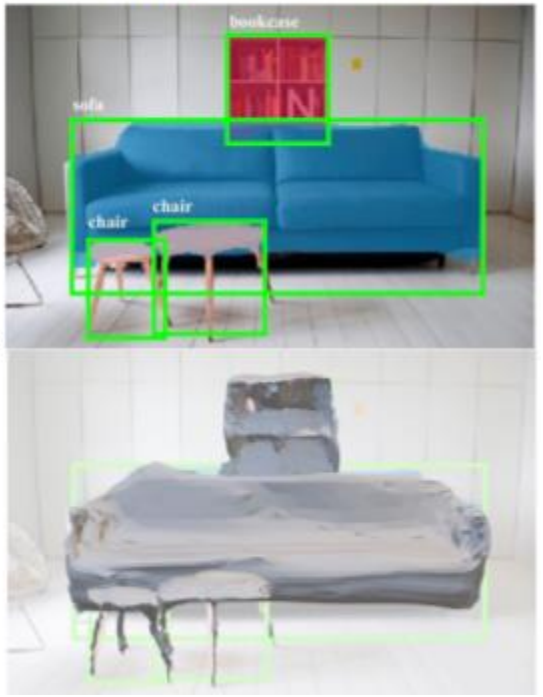


Gkioxari, Malik, and Johnson, "Mesh R-CNN", ICCV 2019

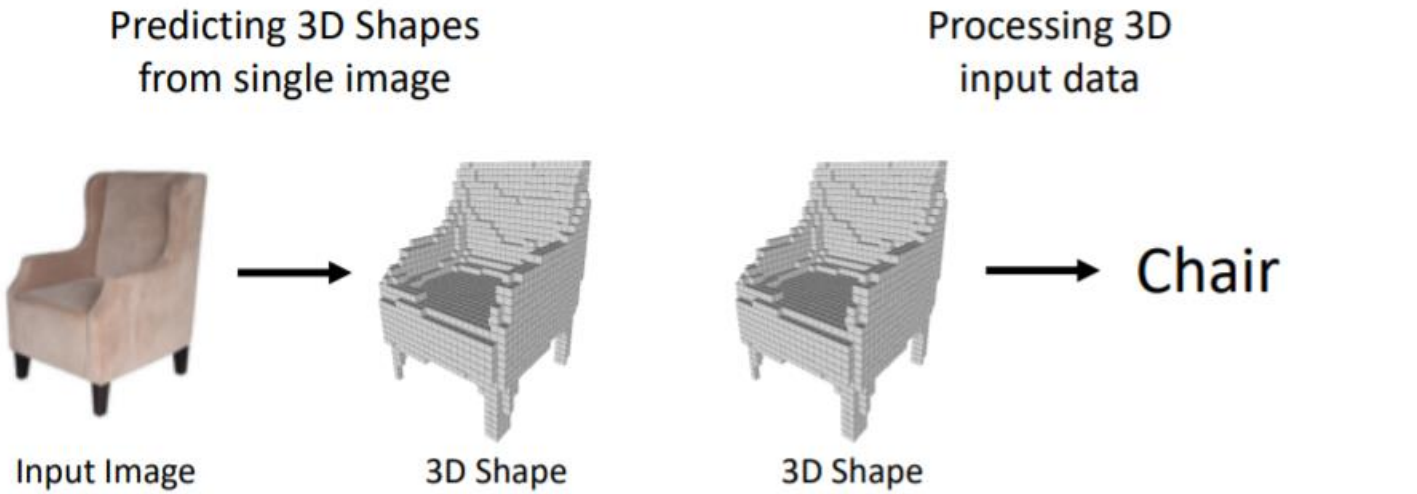
Mask R-CNN:
2D Image -> 2D shapes

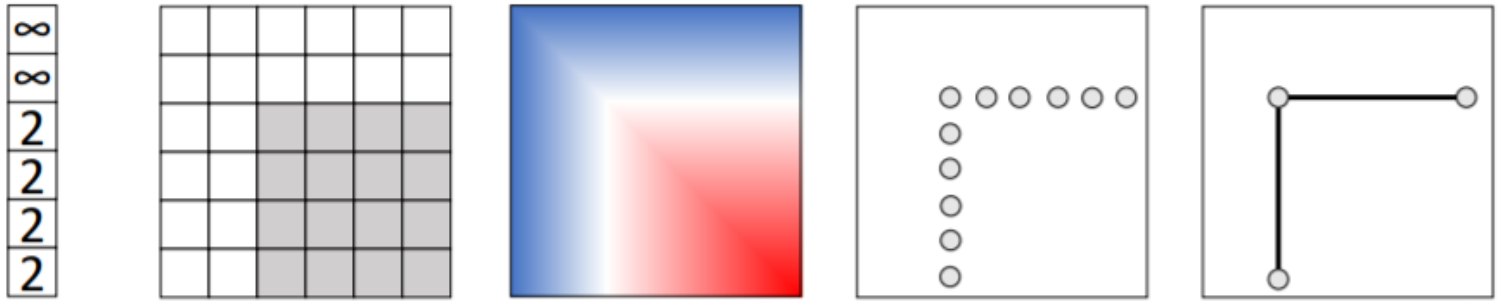


Mesh R-CNN:
2D Image -> **3D** shapes



3D Shape Representations





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

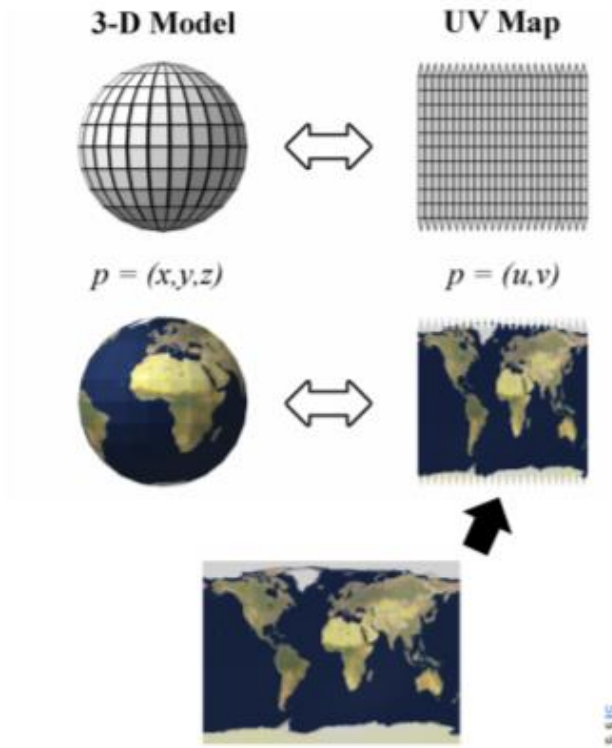
(+) Standard representation for graphics

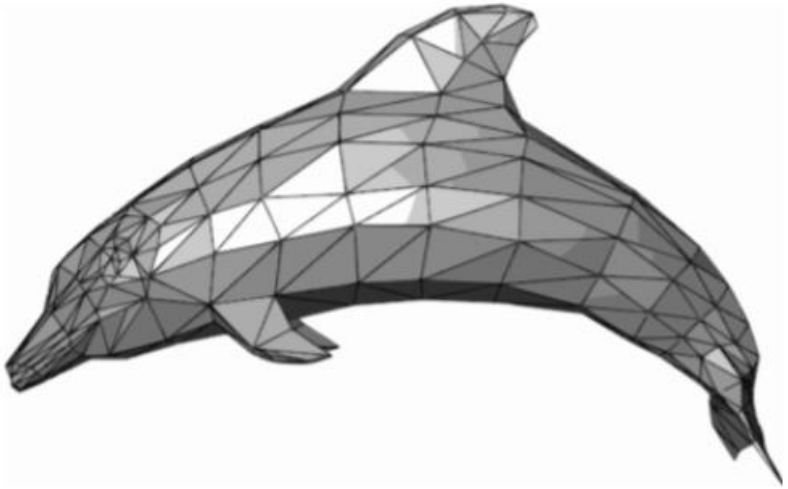
(+) Explicitly represents 3D shapes

(+) 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

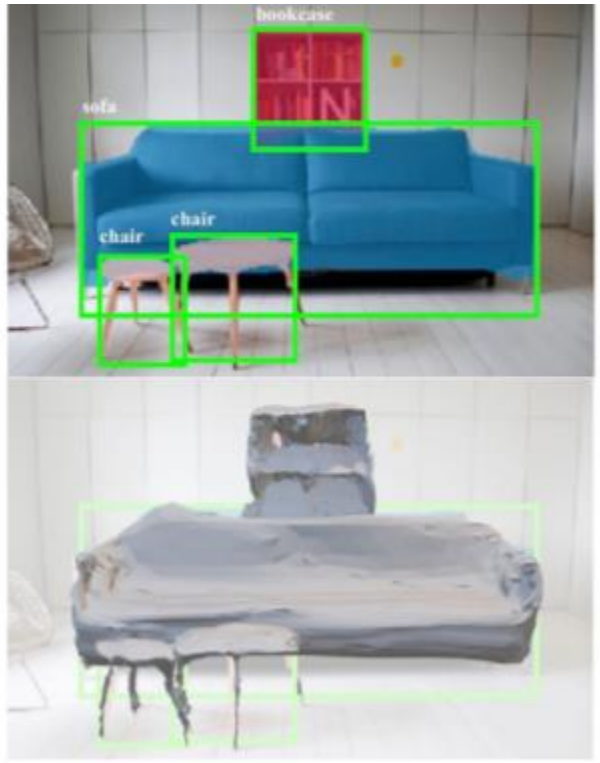
Output:

A set of detected objects

For each object:

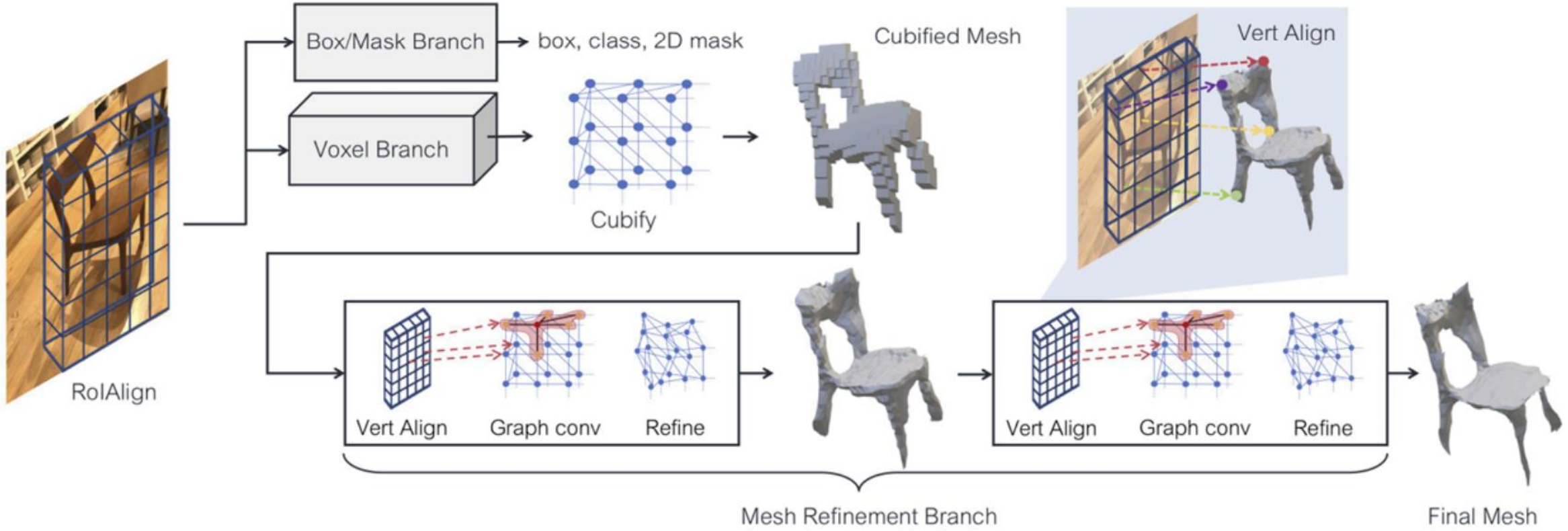
Mask R-CNN
- Bounding box
- Category label
- Instance segmentation

Mesh head
- 3D triangle mesh



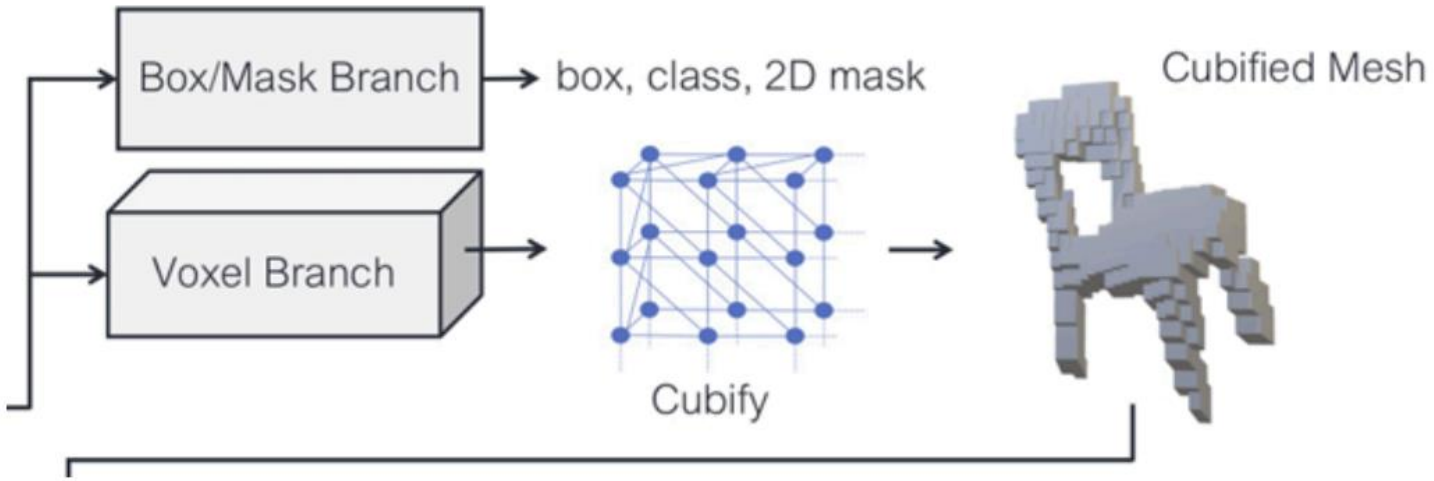
Mesh R-CNN

Mesh R-CNN: Method



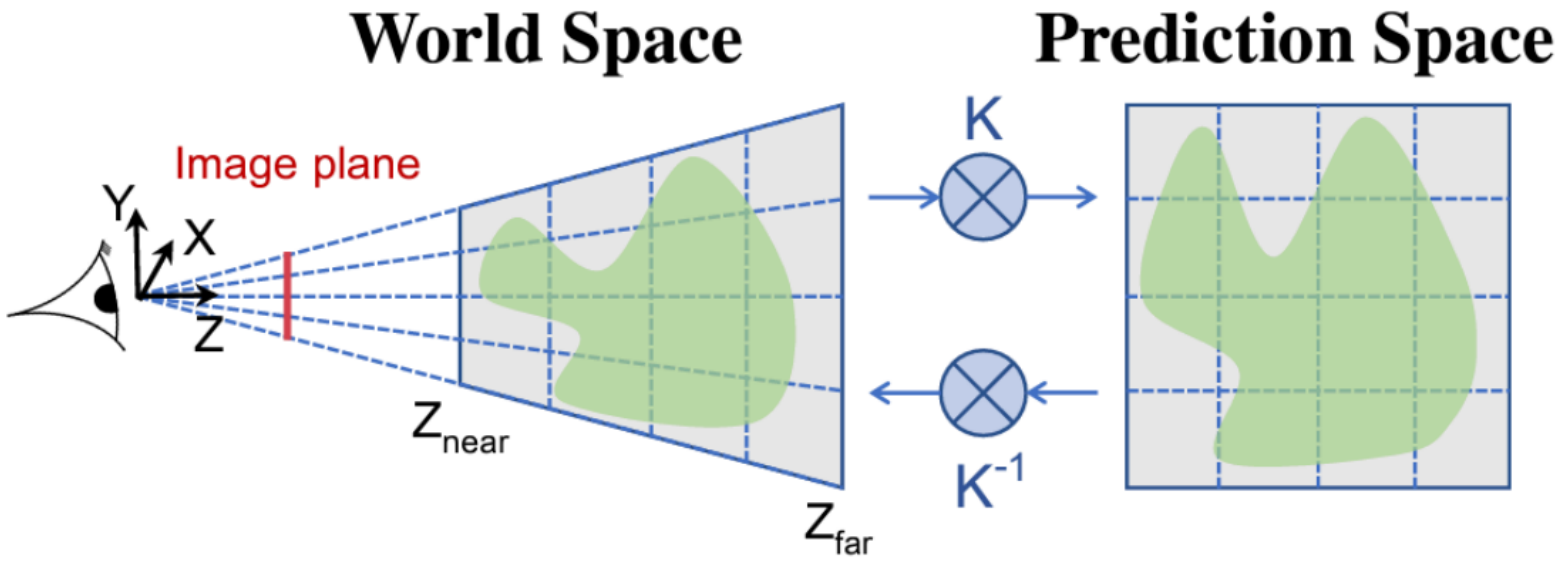
The **voxel branch** predicts a **coarse shape** for each detected object which is further deformed with a sequence of refinement stages in the **mesh refinement branch**

Mesh R-CNN: Voxel Branch



- The **voxel branch** predicts **a grid of voxel occupancy probabilities**.
- Rather than predicting a M×M grid giving the object's shape in the image plane, we instead **predict a G×G×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.

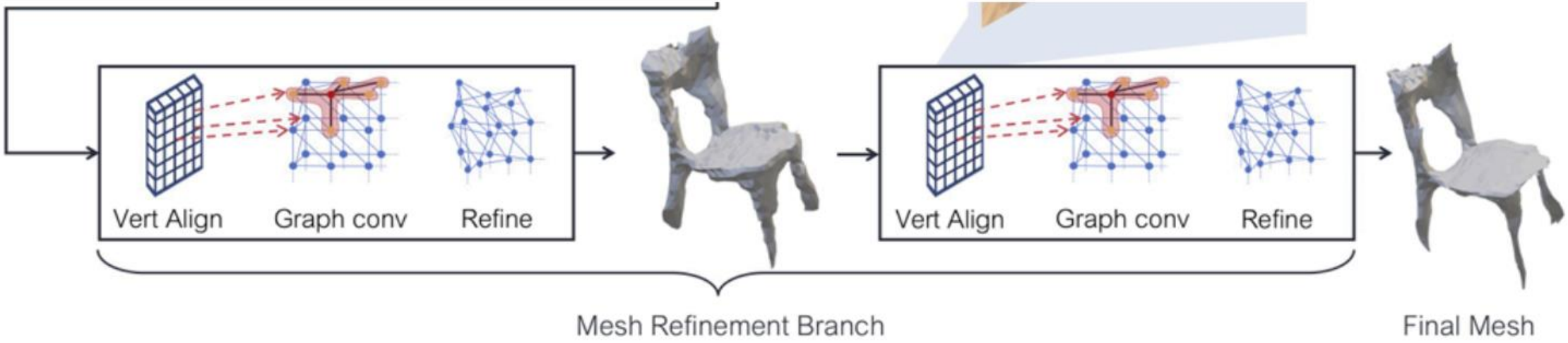
Mesh R-CNN: Voxel Branch



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.**

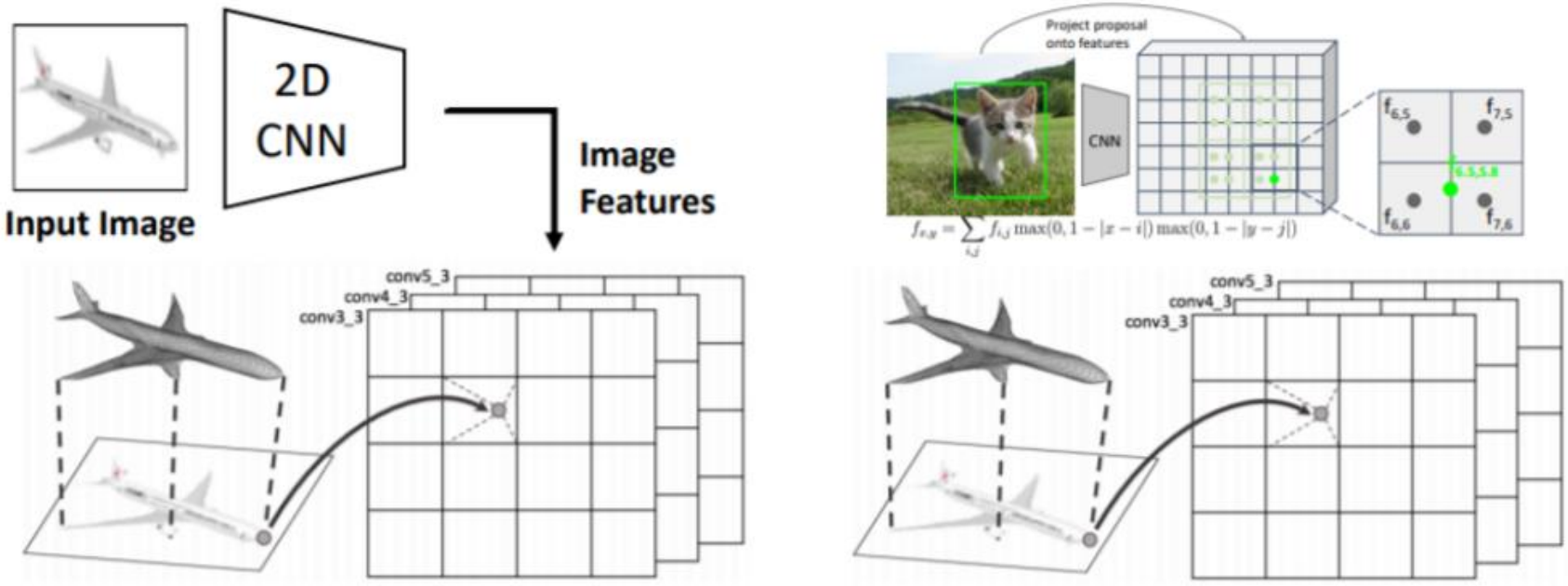
Mesh R-CNN: Mesh Refinement Branch



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

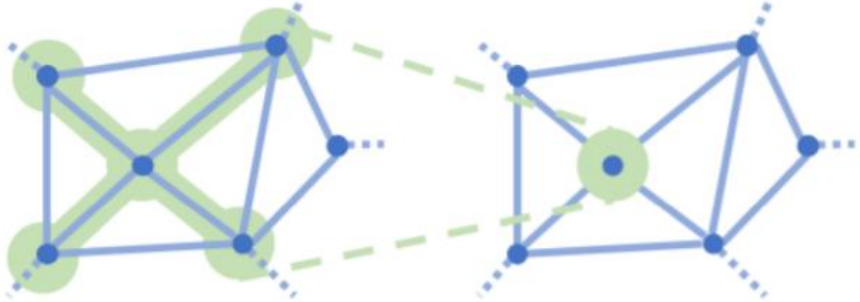
Mesh R-CNN: Vertex Graph Convolution

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

Vertex v_i has feature f_i

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

Use same weights W0 and W1 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

$$v_i' = v_i + \tanh(W_{vert}\,[\bar{f}_i; v_i])$$

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

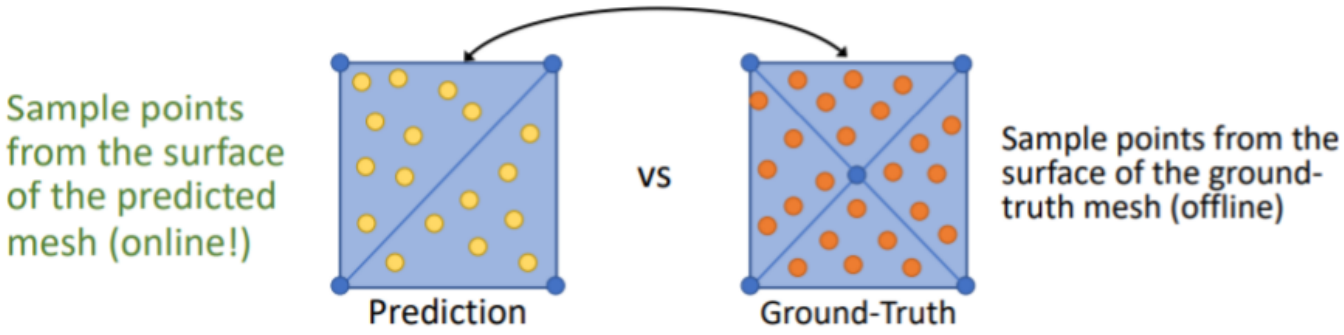
Mesh R-CNN: Loss

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**.

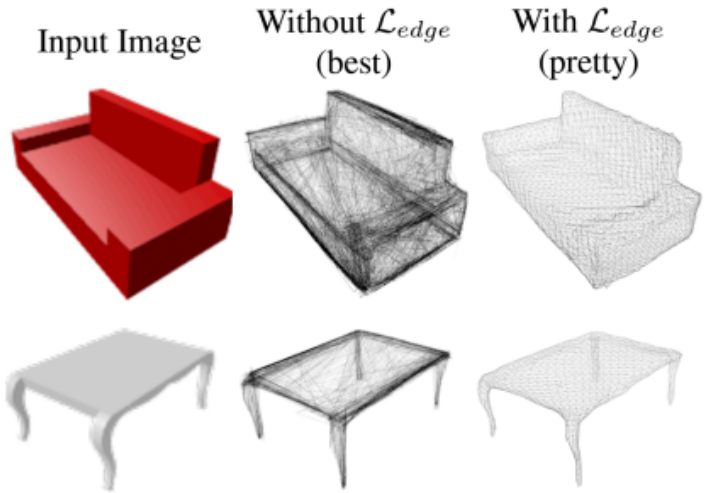


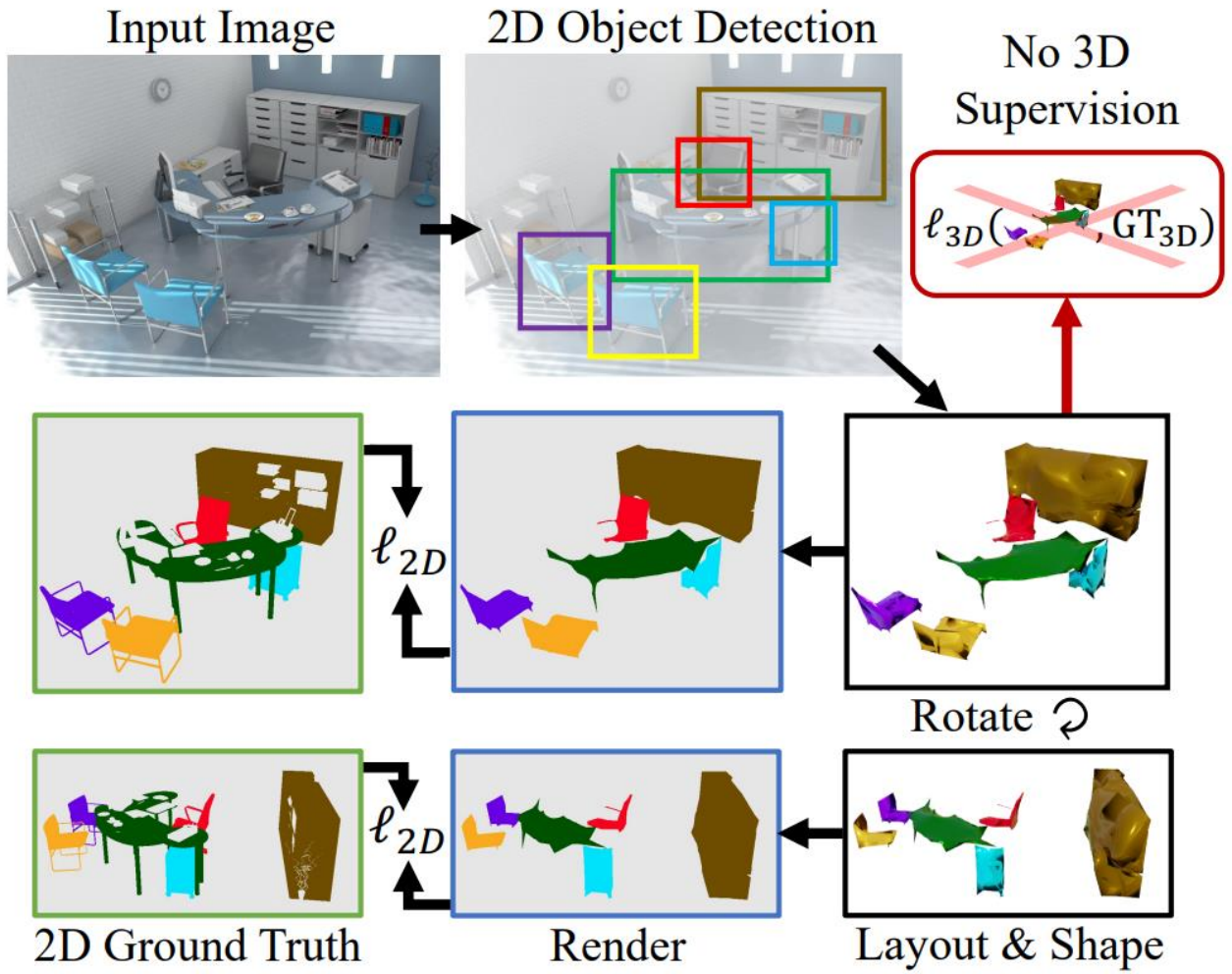
Figure 5. Training without the edge length regularizer $\mathcal{L}_{edge}$ results in degenerate predicted meshes that have many overlapping faces. Adding $\mathcal{L}_{edge}$ eliminates this degeneracy but results in worse agreement with the ground-truth as measured by standard metrics such as Chamfer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

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

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

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

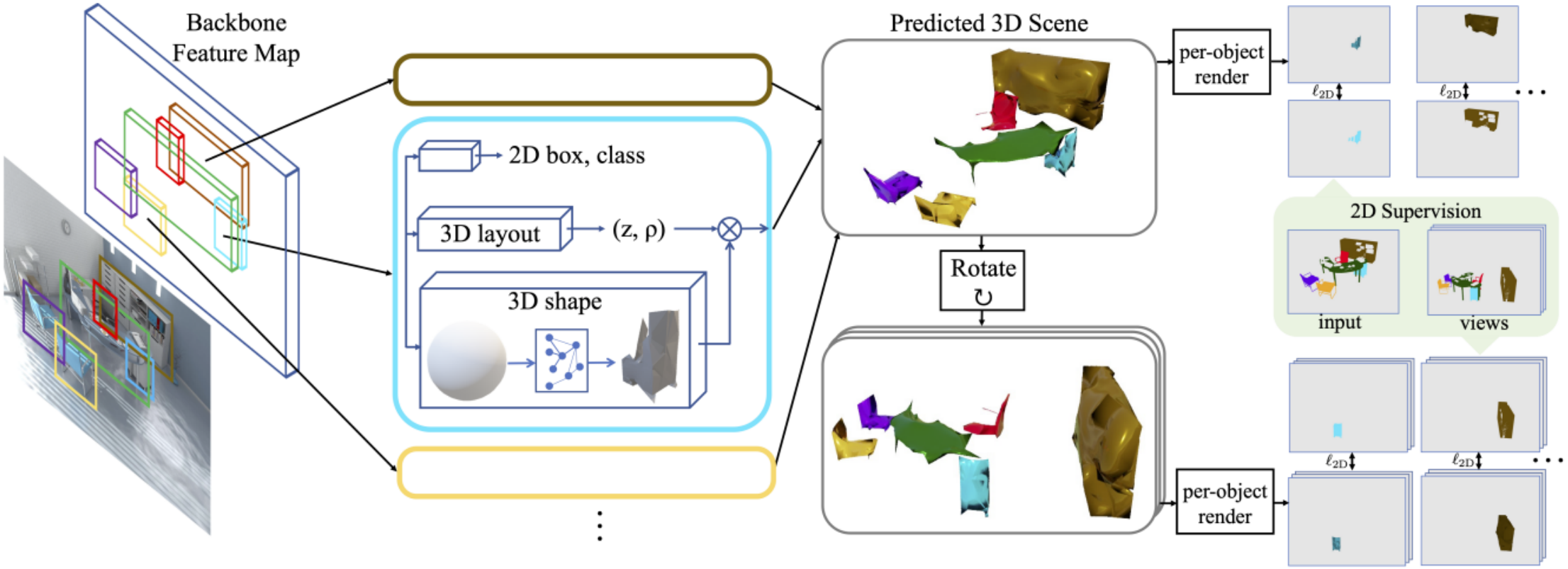


Figure 2. Our model takes as input an RGB image, detects all objects in 2D and predicts their 3D location and shape via *layout* and *shape* heads, respectively. The output is a scene composed of all detected 3D objects. During training, the scene is differentiably rendered from other views and compared 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

```python
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

```python
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
```

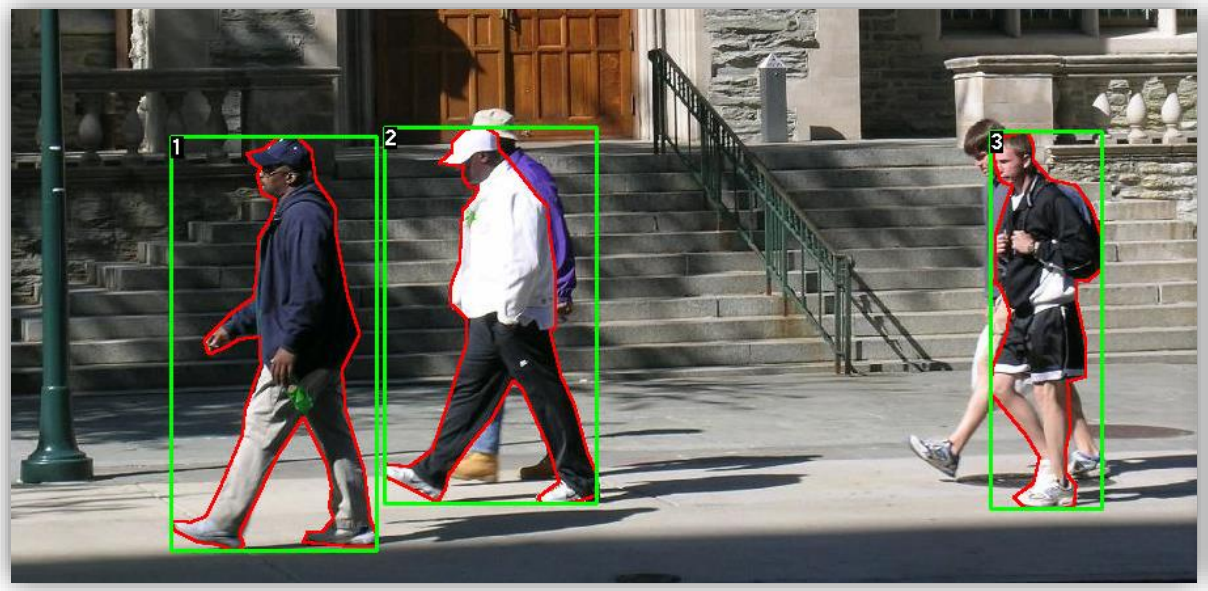
PyTorch

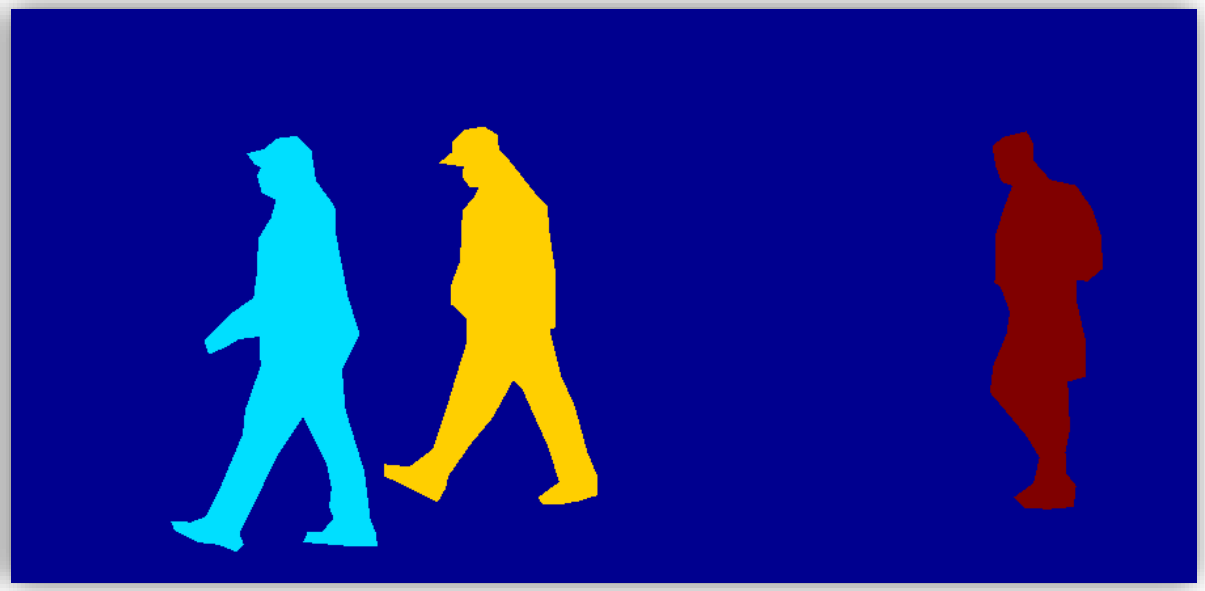
- **Custom dataset __len__**

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

Assignment #2

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





Assignment #2

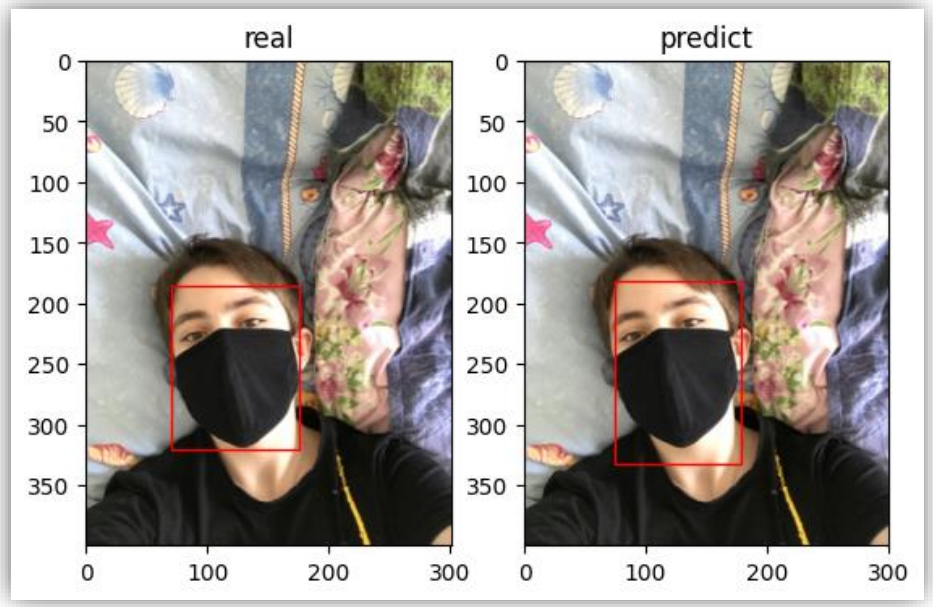
- You will use mask wearing dataset.
- Each image, there is an **annotation file (xml)**. You should close look at those annotation files and figure out what is **bounding boxes, lables, idx**.
- You should fill all the **TODOs** in the code. (Custum Dataset and Model)

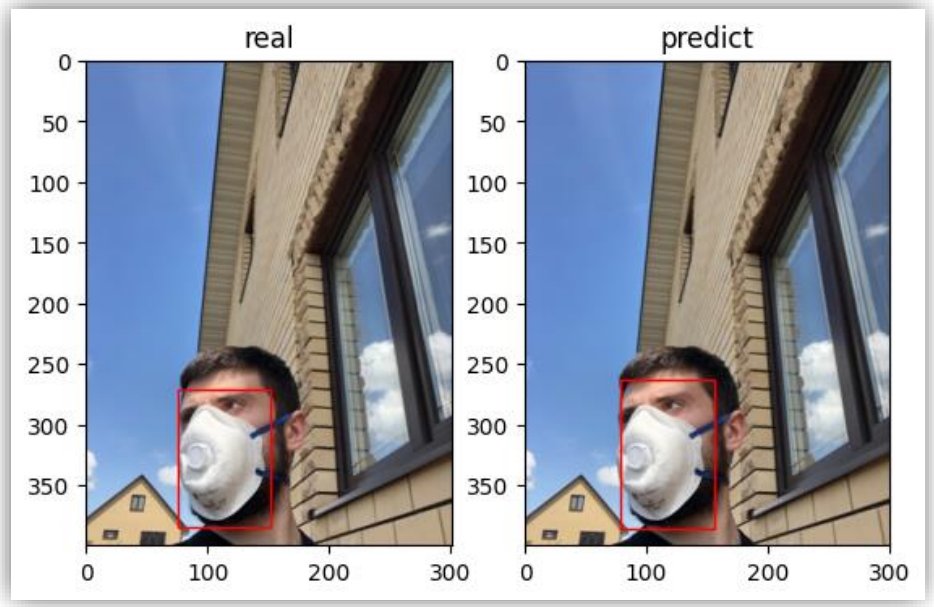
```
# get bounding box coordinates for each mask
boxes = []
labels = []
###########################################################################
# TODO: for each image, get the bounding box coordinates and labels
# boxes and labels are lists
# boxes: [[xmin, ymin, xmax, ymax], ...]
# labels: [label,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

###########################################################################
```

Assignment #2

- **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/KUGODS-2022-GDSC-Deeplearning-Session)





Thank you!
Q & A