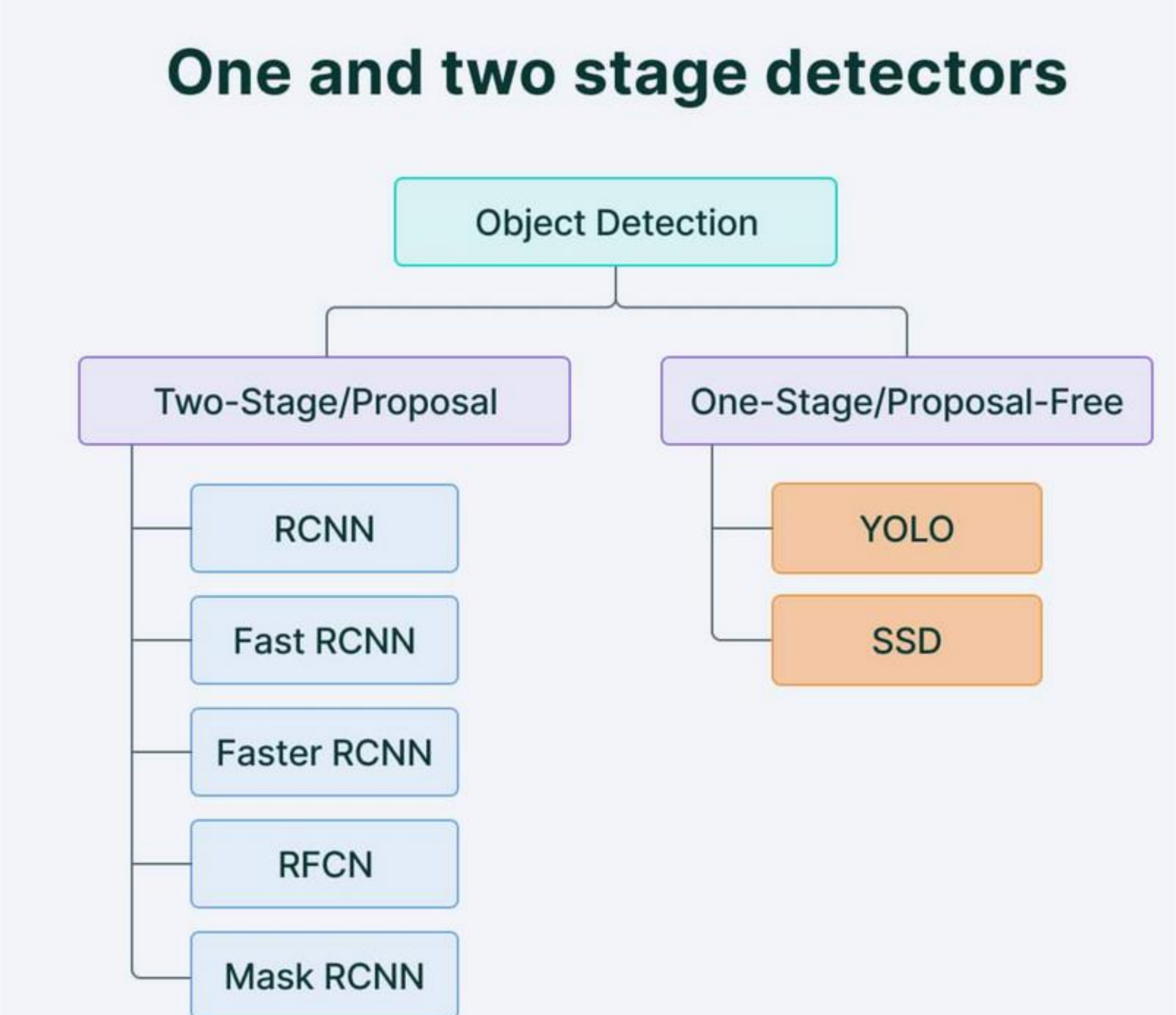
Object Detection

- Object detection is a task that involves **identifying and locating objects** in images or videos.

  - Applications:
    - Self-driving cars
    - Surveillance,
    - Robotics.



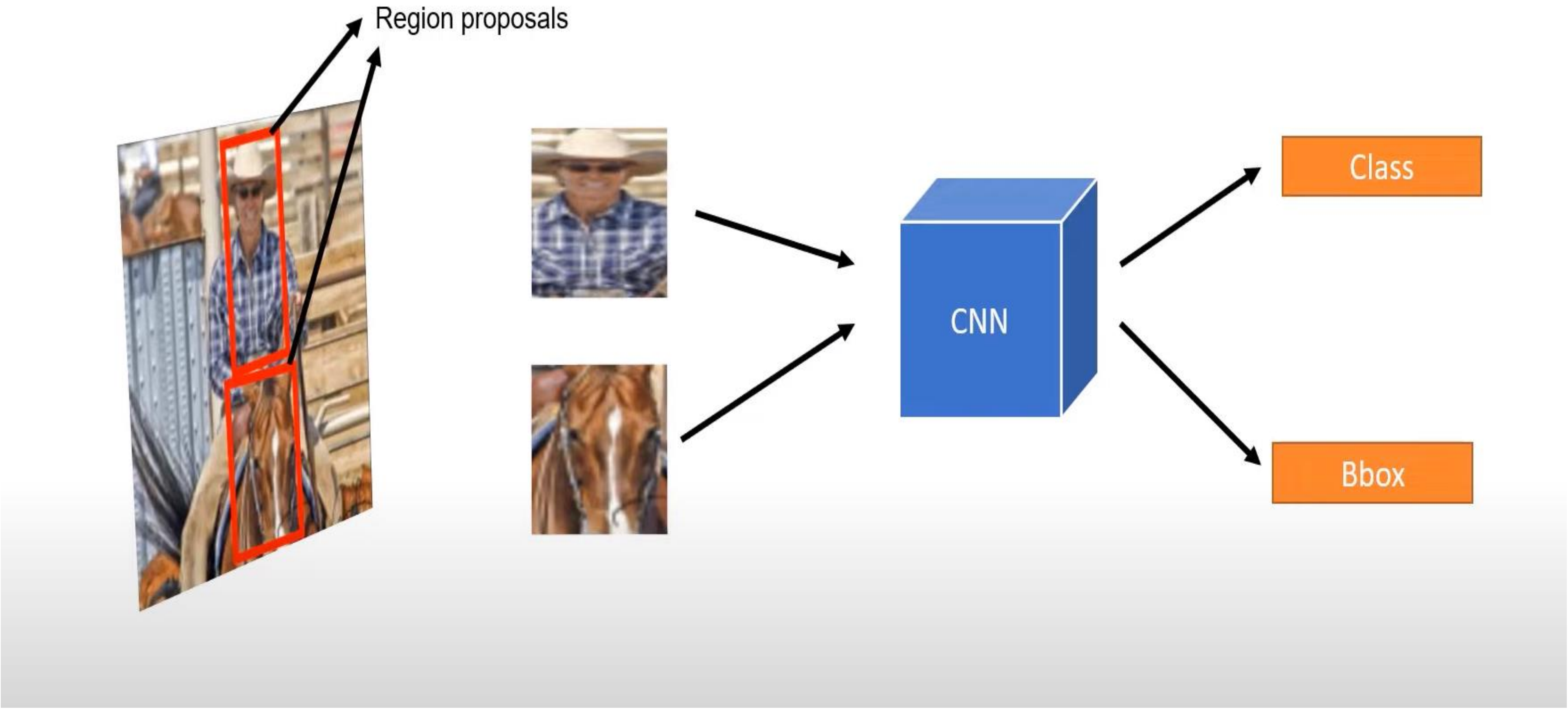
s based on how many

Region-based Convolutional Neural Network (R-CNN)

RCNN consists of three modules.

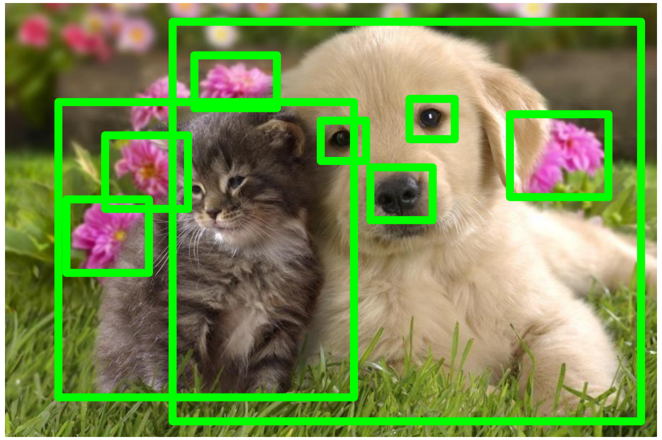
1. The first generates category-independent region proposals. These proposals define the set of candidate detections available to our detector
2. The second module is a large convolutional neural network that extracts a fixed-length feature vector from each region.
3. The third module is a set of class specific linear SVMs. (ONE VS ALL) and regression models

RCNN = Region proposal + CNN

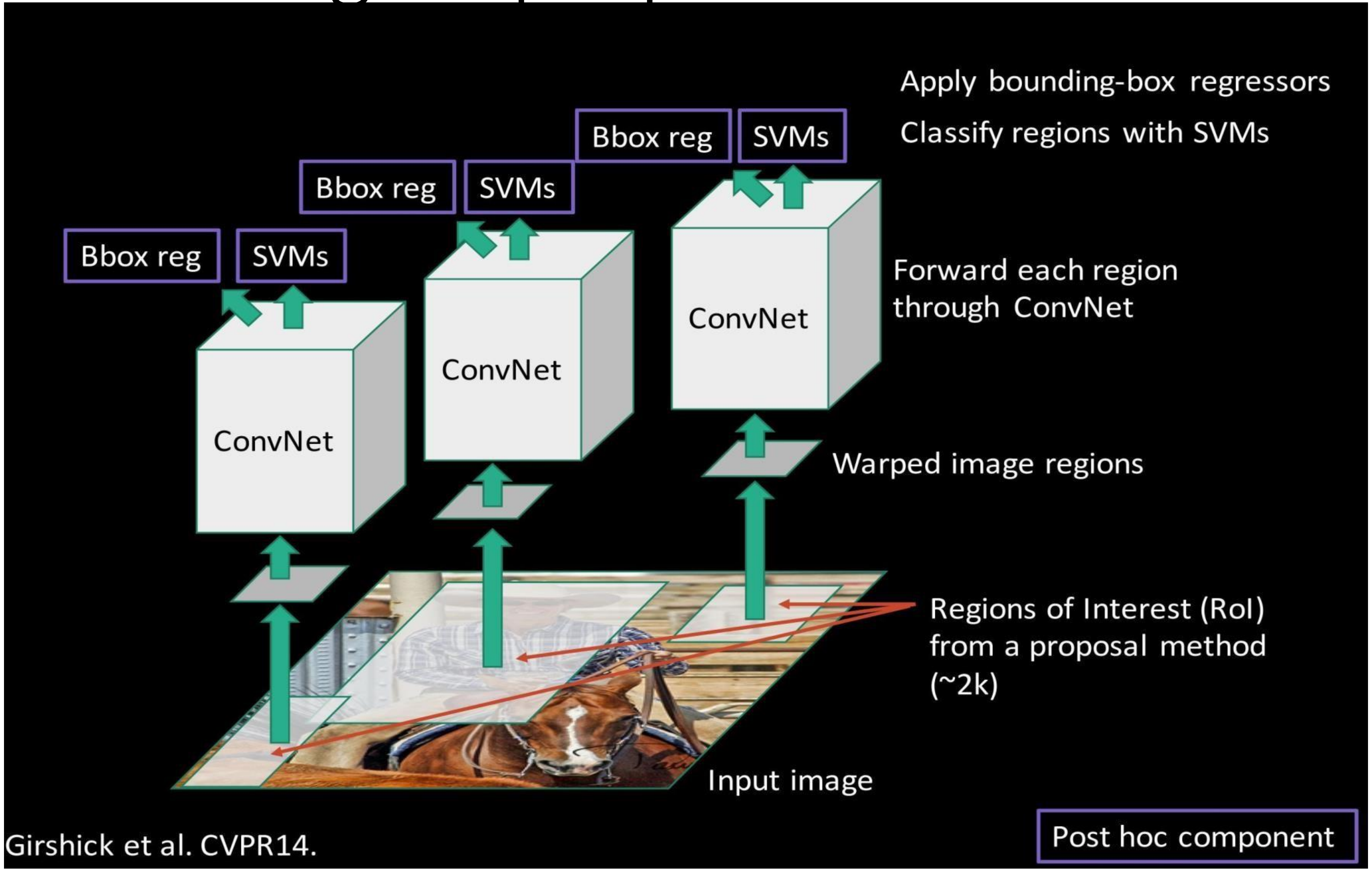


Region Proposals

- Find “blobby” image regions that are likely to contain objects
- “Class-agnostic” object detector (likeHarris corner detection)
- Look for “blob-like” regions



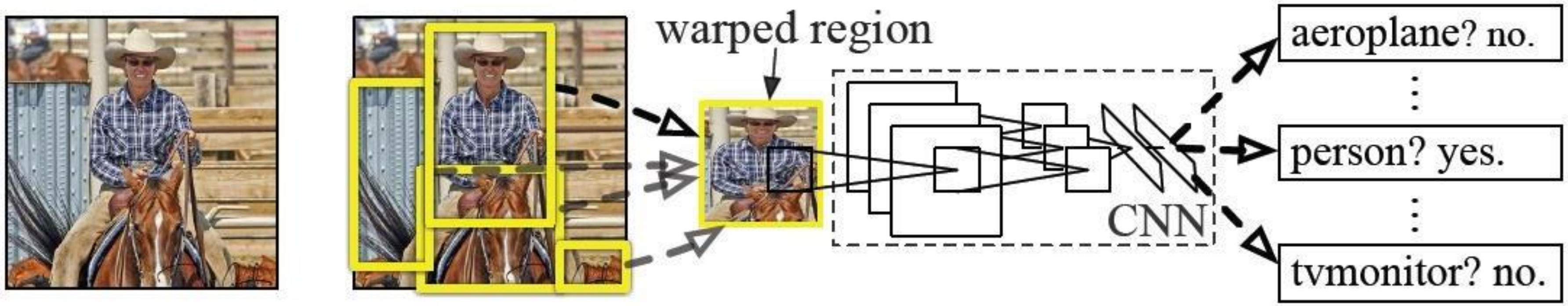
R-CNN: Region proposals + CNN features



Source: R. Girshick

R. Girshick, J. Donahue, T. Darrell, and J. Malik, [Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation](), CVPR 2014.

R-CNN details

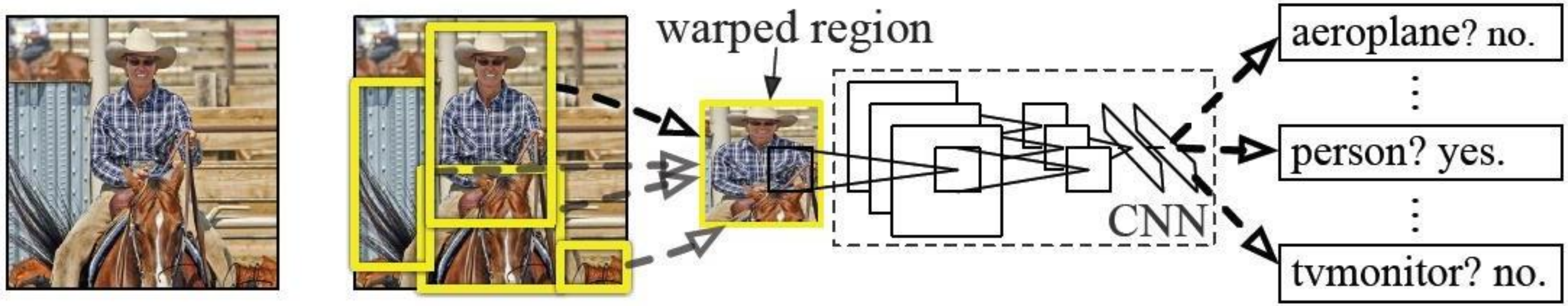


- **Regions**: uses ~2000 Selective Search proposals
- **Network**: uses AlexNet *pre-trained* on ImageNet (1000 classes), *fine-tuned* on PASCAL (21 classes)
- **Final detector**: (training steps/details)
  - first warp proposal regions,
  - then extract fc7 network activations [AlexNet] (4096 dimensions),
  - Finally, classify with linear SVM
- **Bounding box regression** is also used to refine box locations

R. Girshick, J. Donahue, T. Darrell, and J. Malik, Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation, CVPR 2014.

Issue #1 with R-CNN

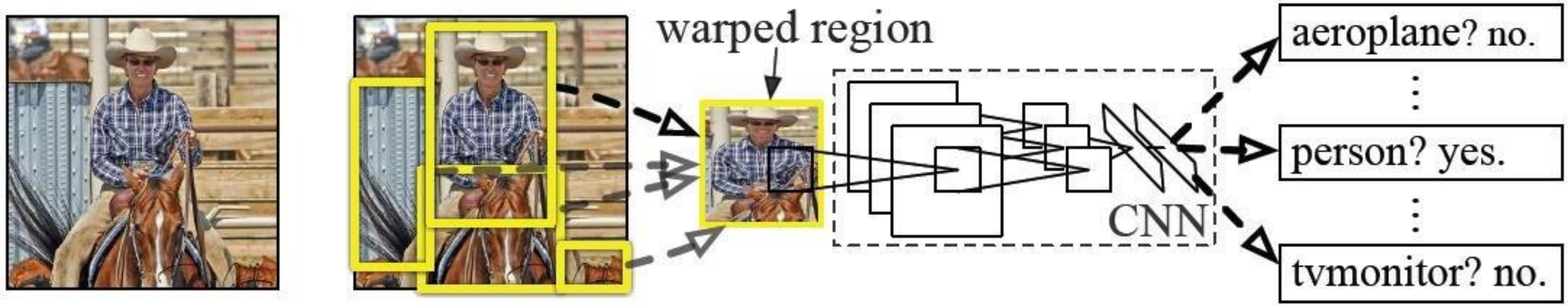
- Slow in run-time
  - Multiple forward passes for each proposal
  - There are thousands of proposals



- Solution
  - Single forward pass for each image?

Issue #2 with R-CNN

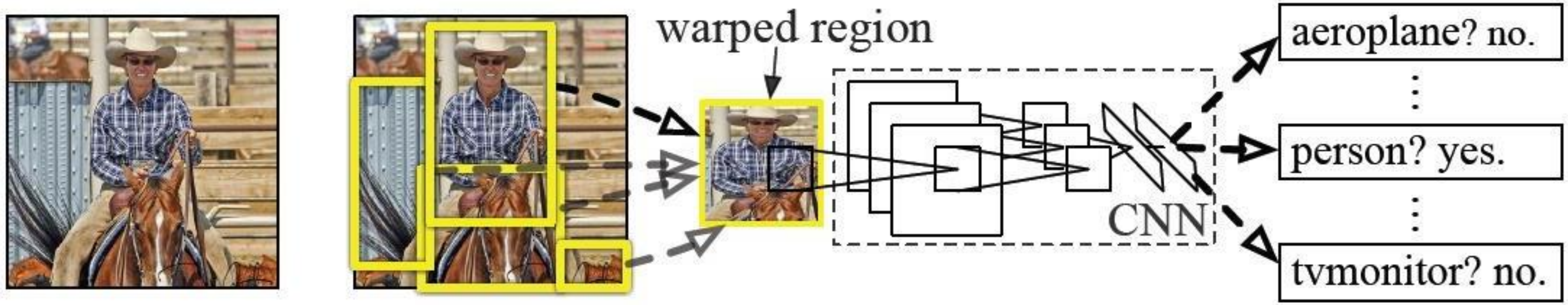
- Separate classifier training
  - CNN feature extractor is not trained with classifier and regressor



- Solution
  - End-to-end training?

Issue #3 with R-CNN

- Complex training pipeline
  - Proposals
  - Feature extraction
  - Classification

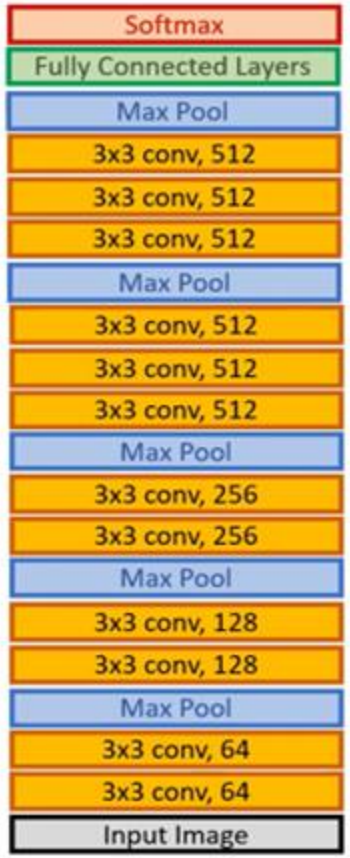


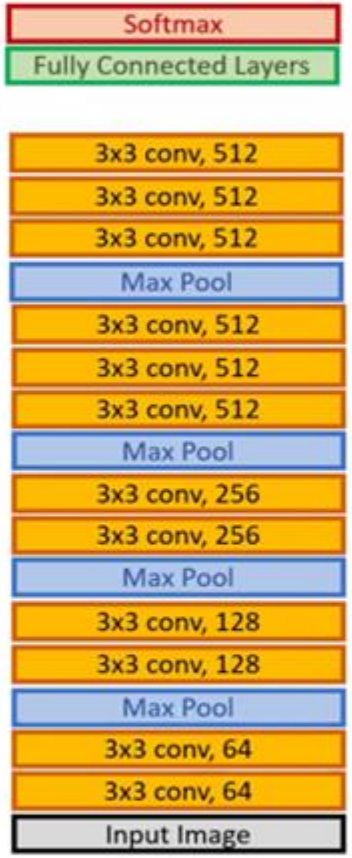
- Solution
  - Single forward pass for each image?

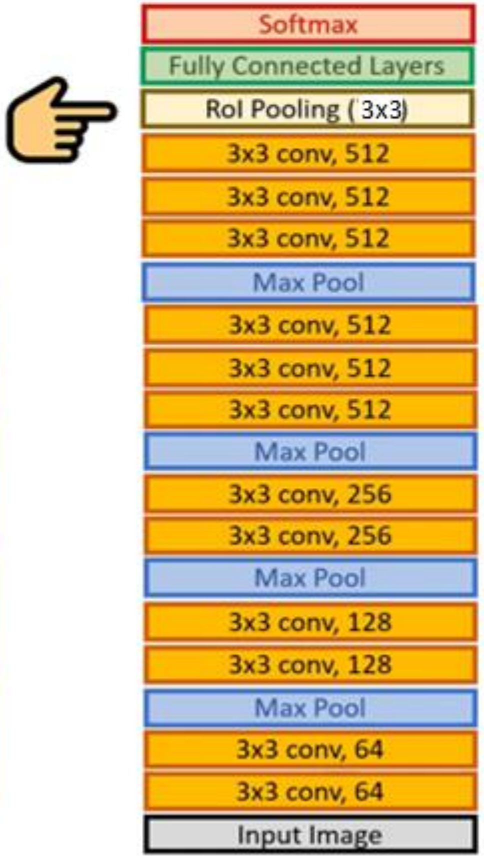
Solution

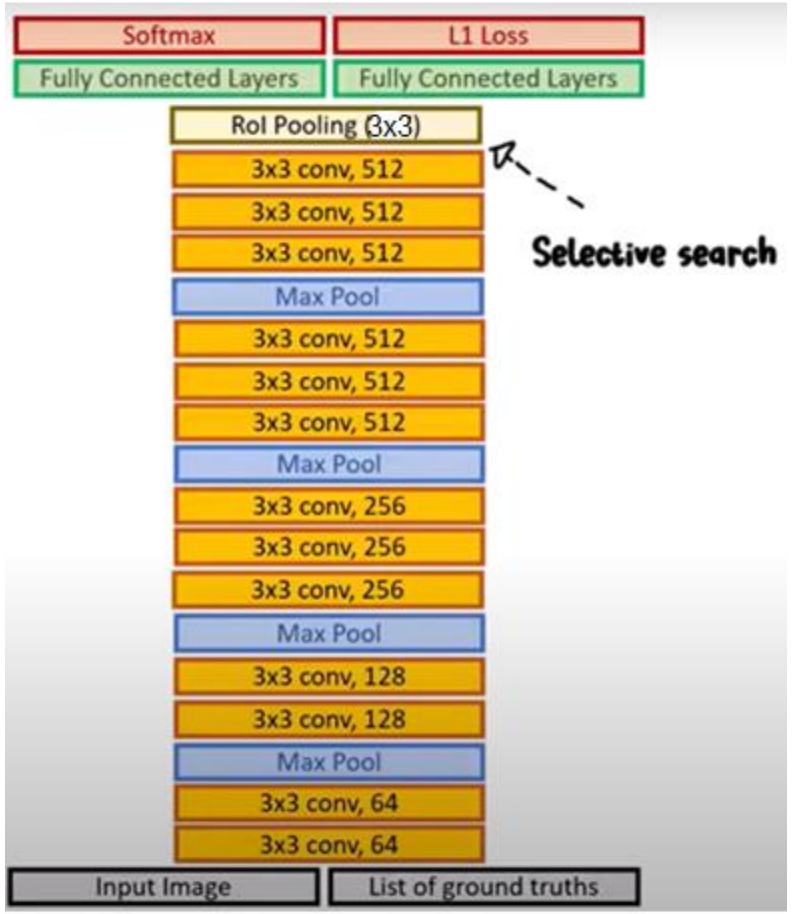
- Fast R-CNN
  - Single forward pass for each image
  - No separate classifier
  - End-to-end training

VGG 16 modification for Fast R-CNN





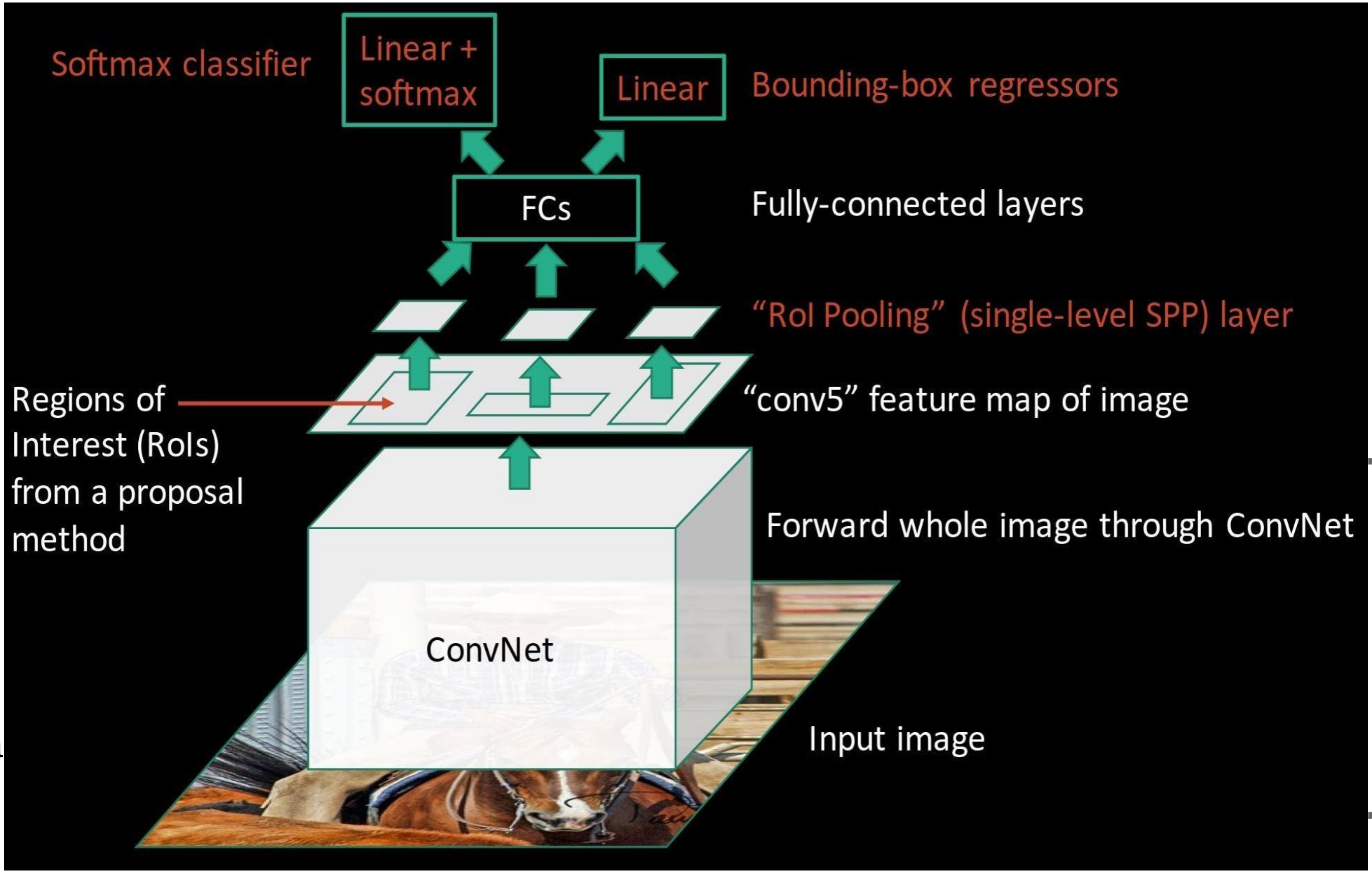




Fast R-CNN

Solution #2:
Train softmax classifier jointly

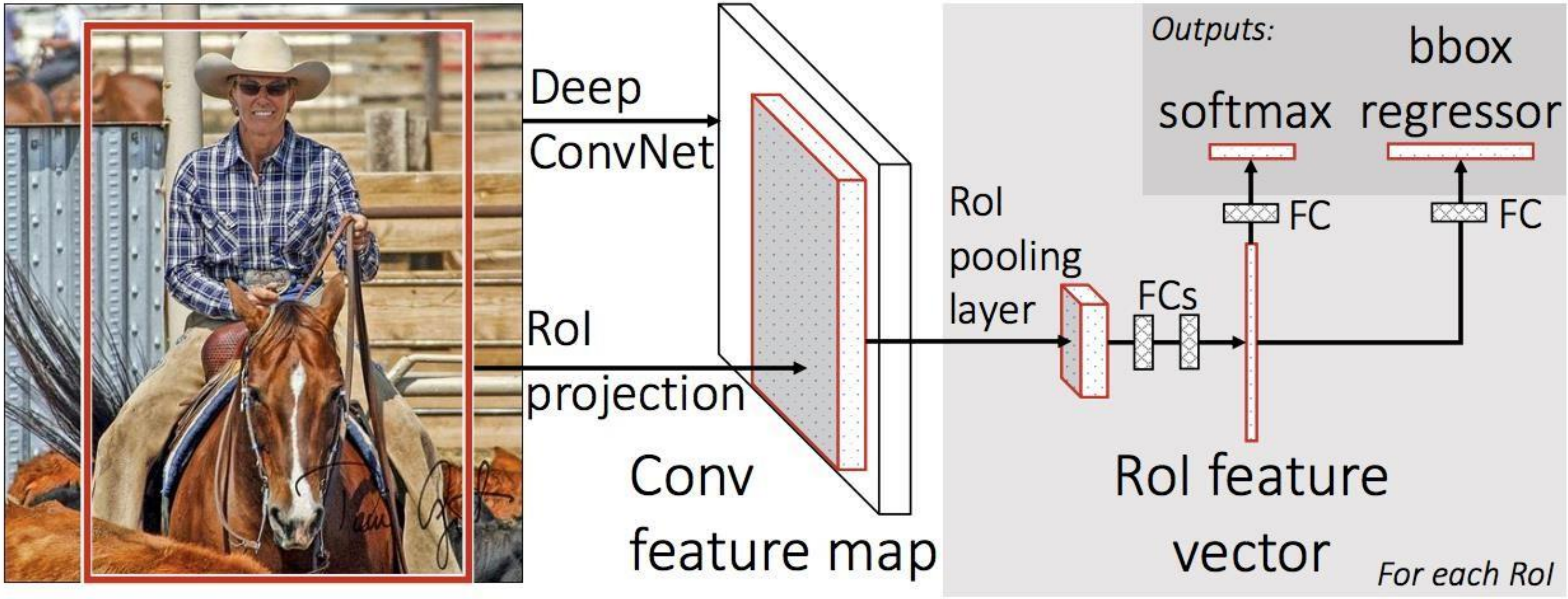
Solution #3:
Just train the whole system end-to-end a at once!



Solution #1:
Share computation of convolutional layers between proposals for an image

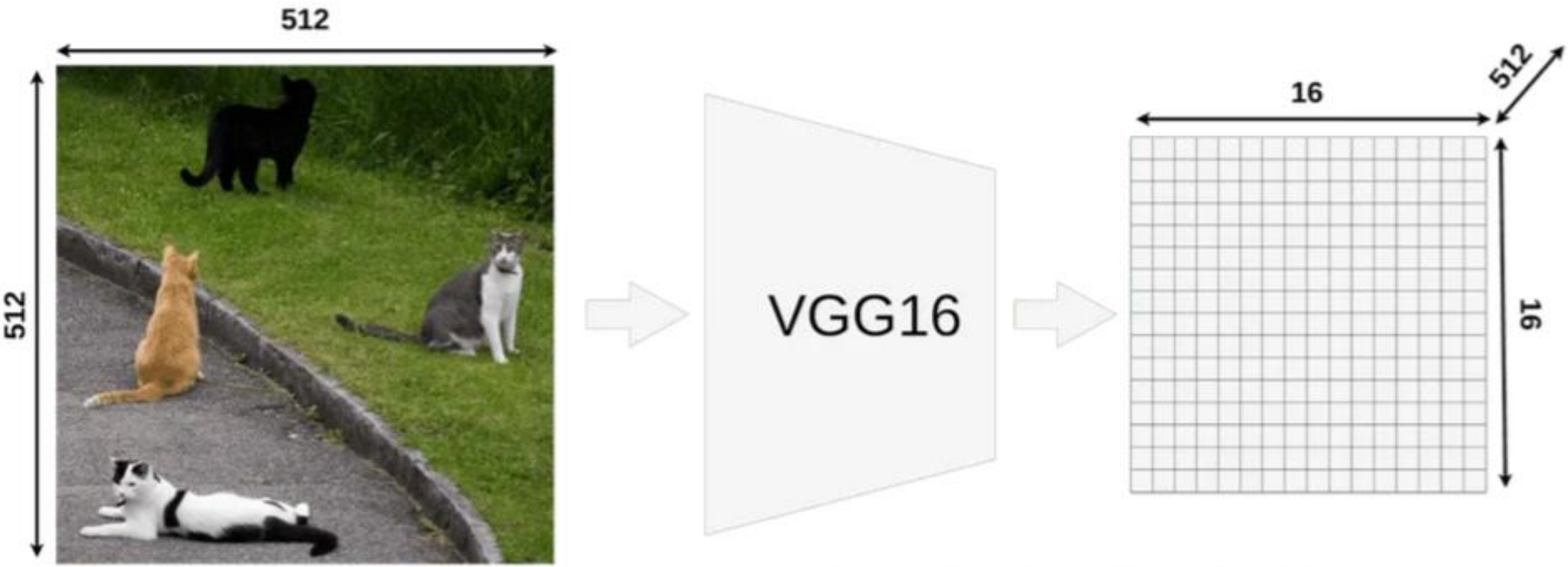
Slide credit: Ross Girschick, Girschick, "Fast R-CNN", ICCV 2015

Fast R-CNN: Another view



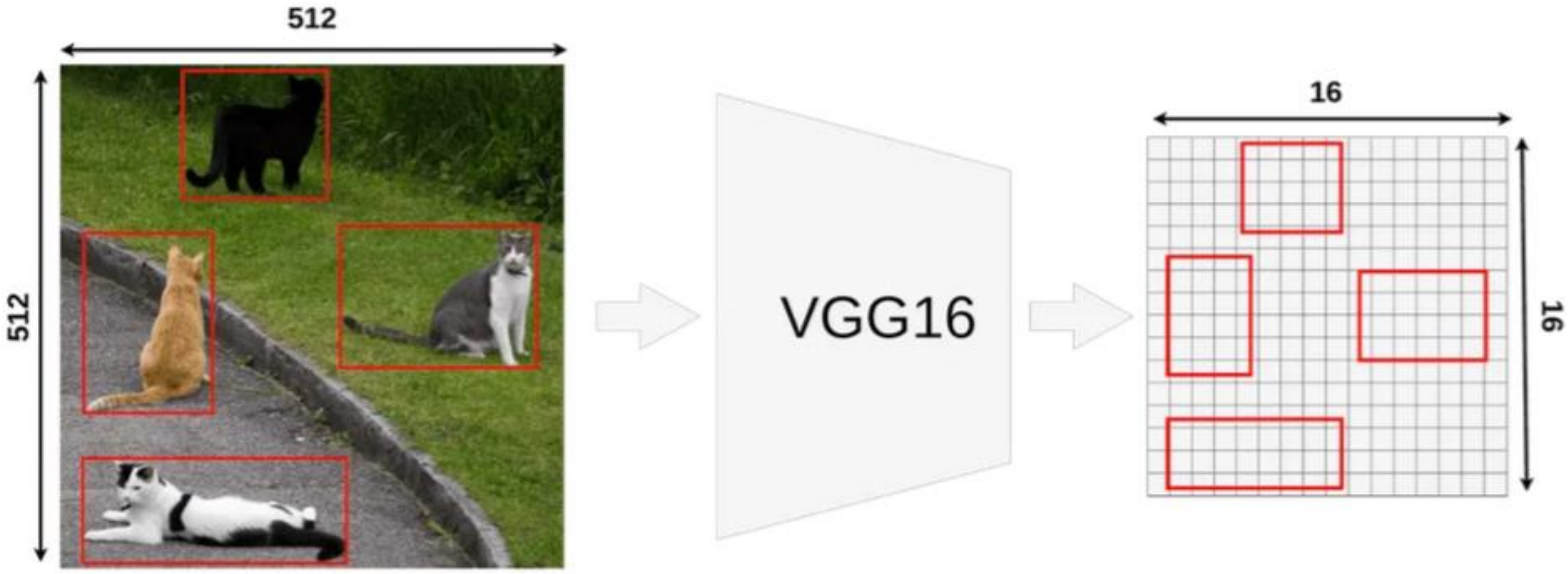
R. Girshick, [Fast R-CNN](), ICCV 2015

ROI-Pooling



If you look at the output matrix you should notice that it's width and height is
exactly 32 times smaller than the input image (512/32 = 16). That's important

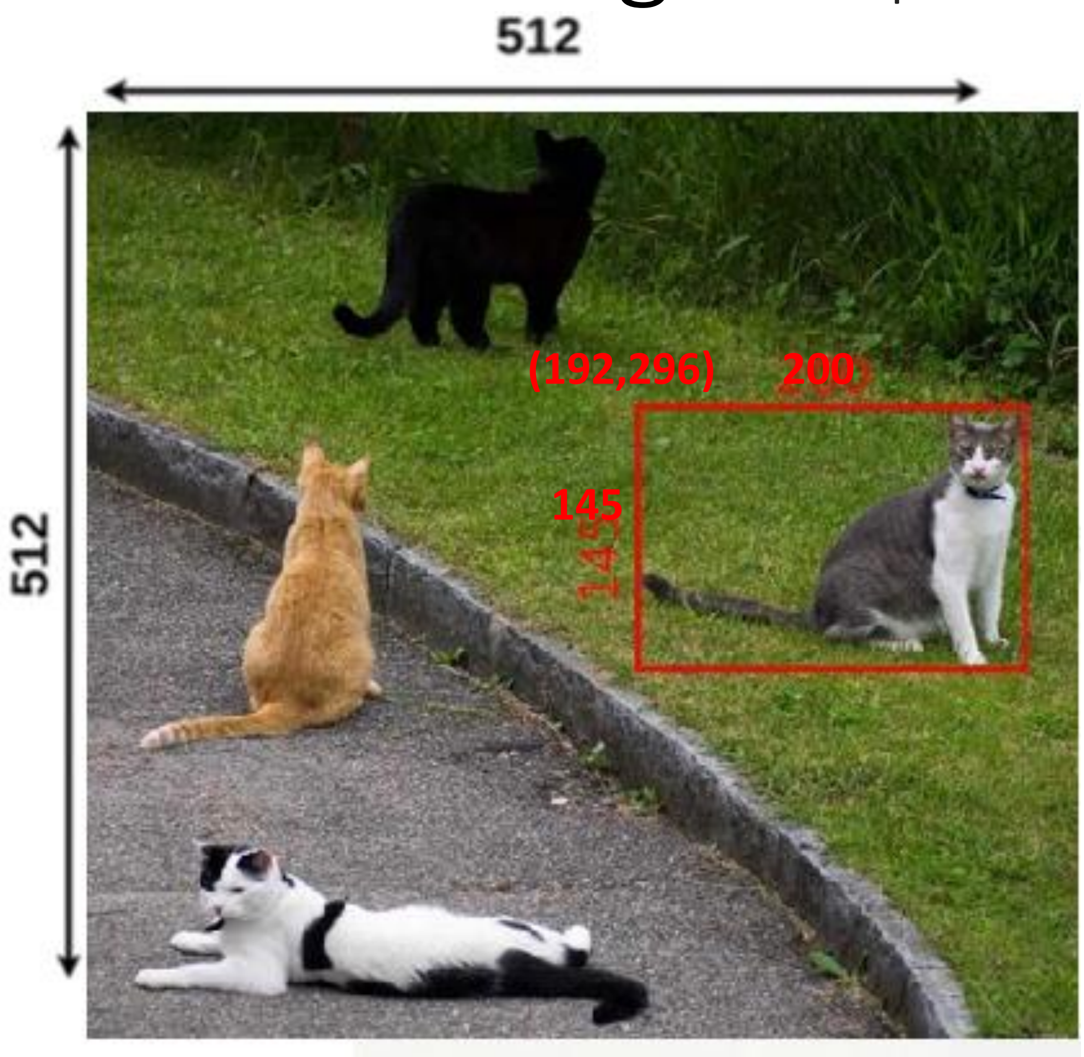
ROI-Pooling



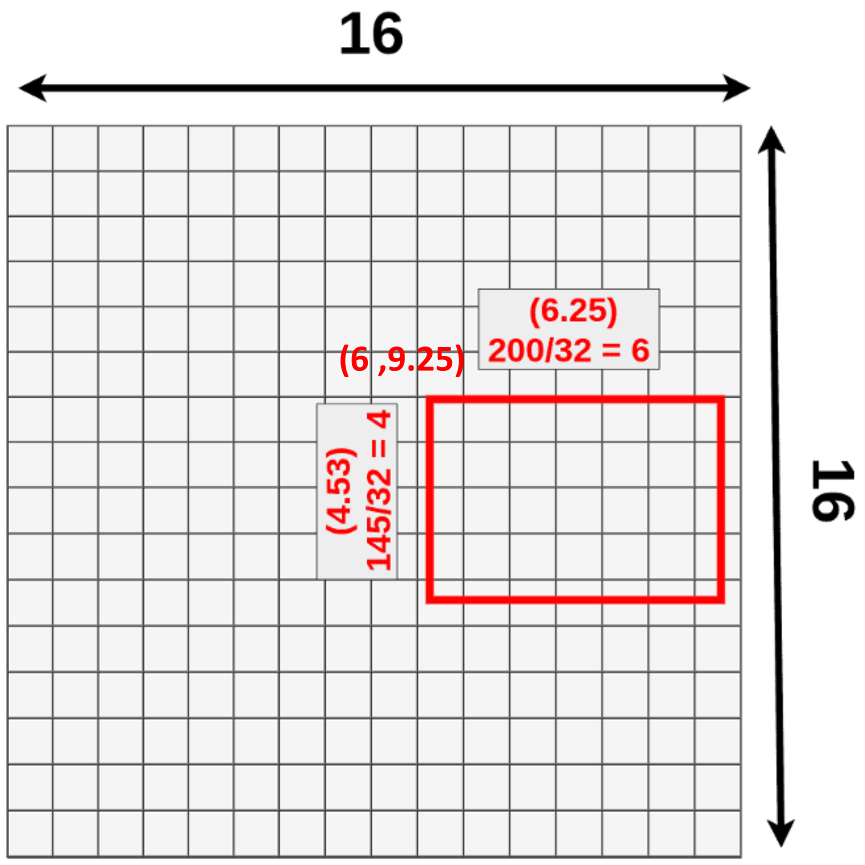
ROI-Pooling

- Original size = **145x200**
- Top left corner= **(192,296)**

- Scale factor = **32**
- (145/32, 200/32) = **(4.53, 6.25)**
- (192/32 , 296/32) = **(6 , 9.25)**



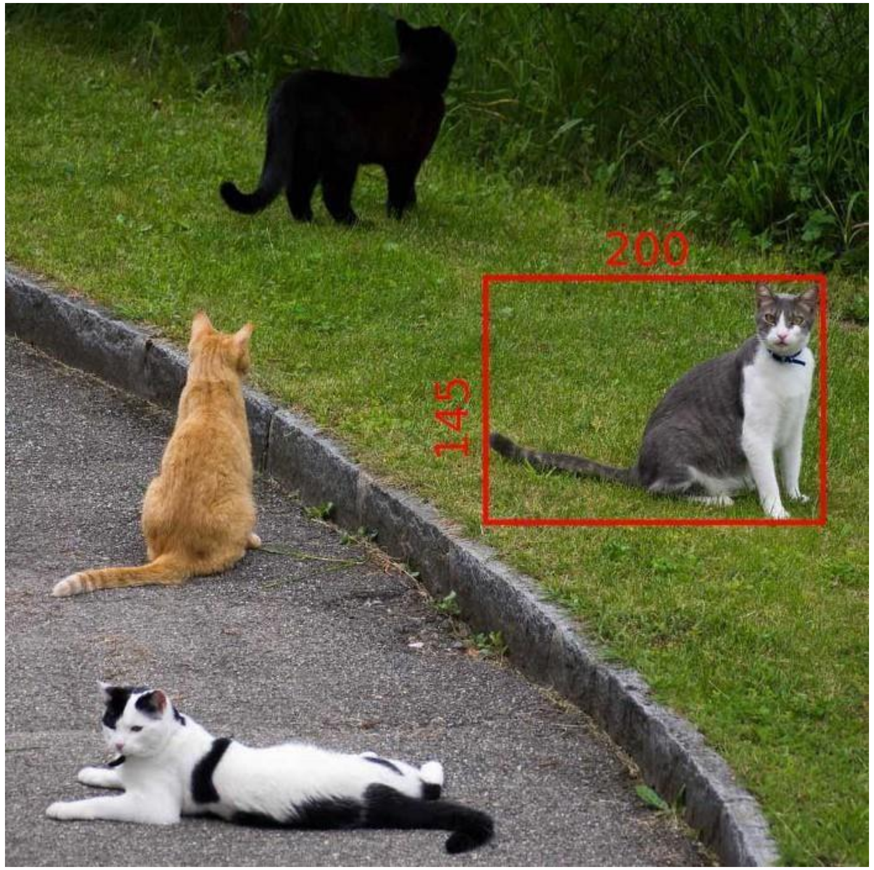
https://tinyurl.com/y6xpm24d

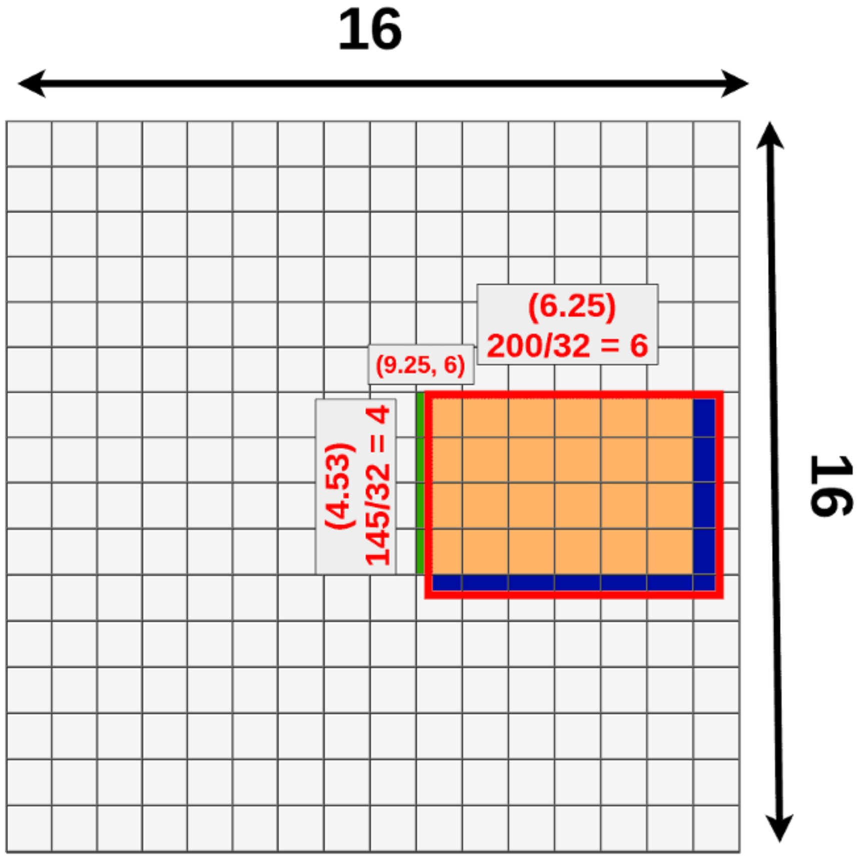


ROI-Pooling

Apply Quantization

Lost data = dark blue
Gain new data = green

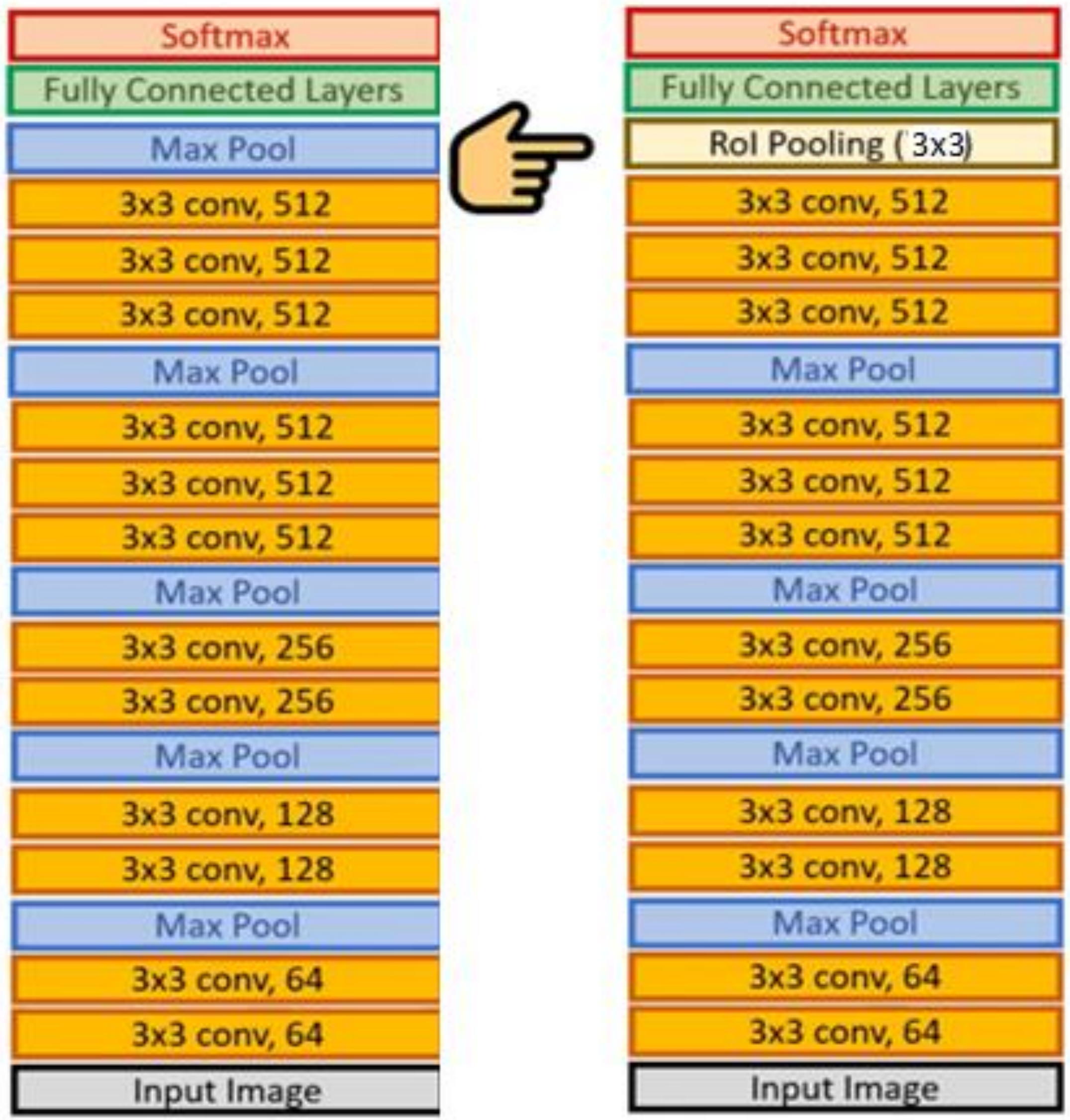




https://tinyurl.com/y6xpm24d

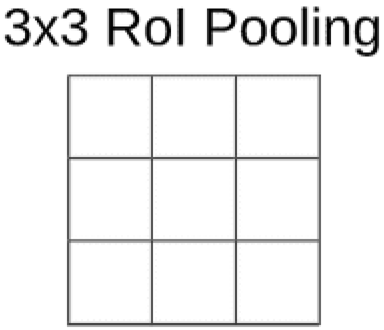
It's still going to work in case of Object detection but case of segmentation it will handle with **RoIAlign.**

VGG-16 architecture for Fast R-CNN



- After **RoI Pooling Layer** there is a **Fully Connected layer** with a fixed size.
- Because our RoIs have different sizes we have to pool them into the same size (**3x3x512** in *our* example).
- At this moment our mapped RoI is a size of **4x6x512** and as you can imagine we cannot divide **3x3.** That’s where quantization strikes again.

ROI-Pooling (Apply Quantization)

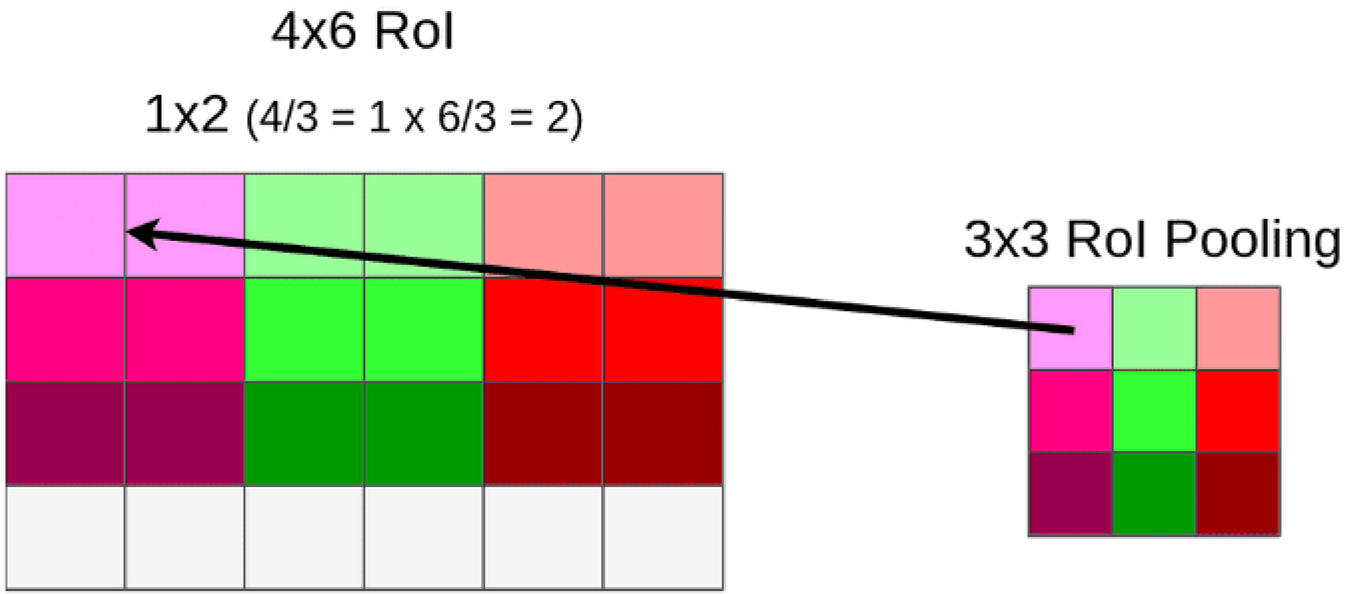


This time we don’t have to deal with coordinates, only with size.

- **4 / 3 = 1.33 = 1** After applying quantization (round down)
- **6 /3 = 2**
- **Use 1 x 2 kernel** for ROI-Pooling with **stride 2** to produce **3x3** output.

https://tinyurl.com/y6xpm24d

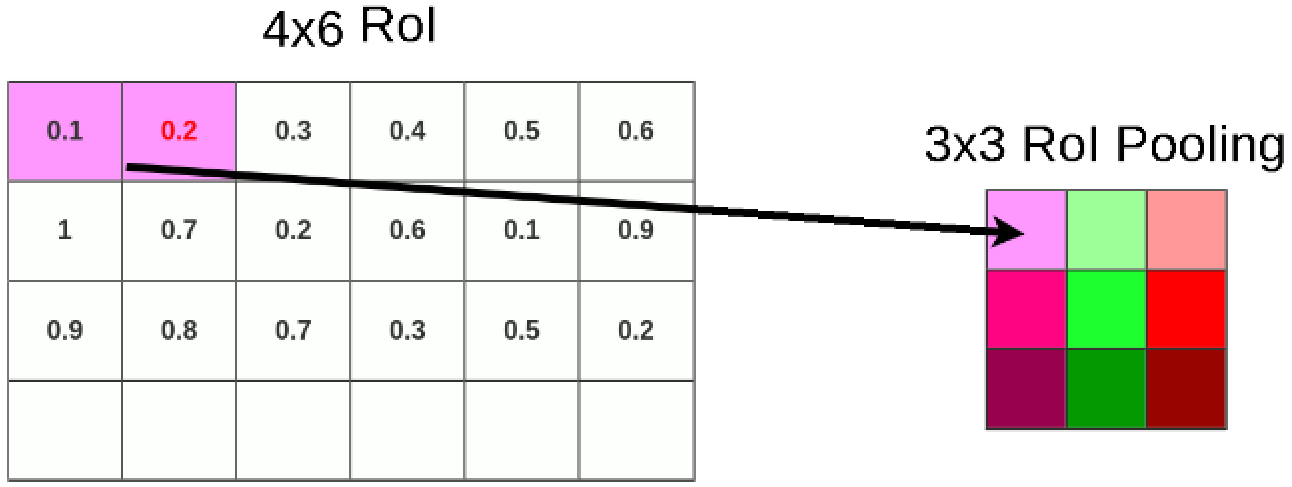
ROI-Pooling



Because of quantization, we’re losing whole bottom row once again.

https://tinyurl.com/y6xpm24d

ROI-Pooling - Example



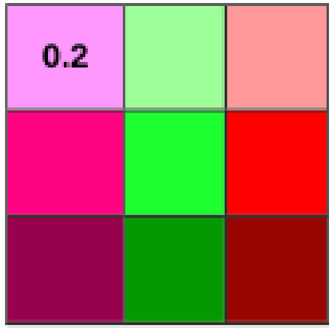
Apply **max pool** operation using **1x2** filter with stride **2**

ROI-Pooling - Example

4x6 RoI

0.1	0.2	0.3	0.4	0.5	0.6
1	0.7	0.2	0.6	0.1	0.9
0.9	0.8	0.7	0.3	0.5	0.2

3x3 RoI Pooling



ROI-Pooling - Example

4x6 RoI

0.1	0.2	0.3	0.4	0.5	0.6
1	0.7	0.2	0.6	0.1	0.9
0.9	0.8	0.7	0.3	0.5	0.2

3x3 RoI Pooling

0.2	0.4	0.6
1	0.6	0.9
0.9	0.7	0.5

Data pooling process

ROI-Pooling

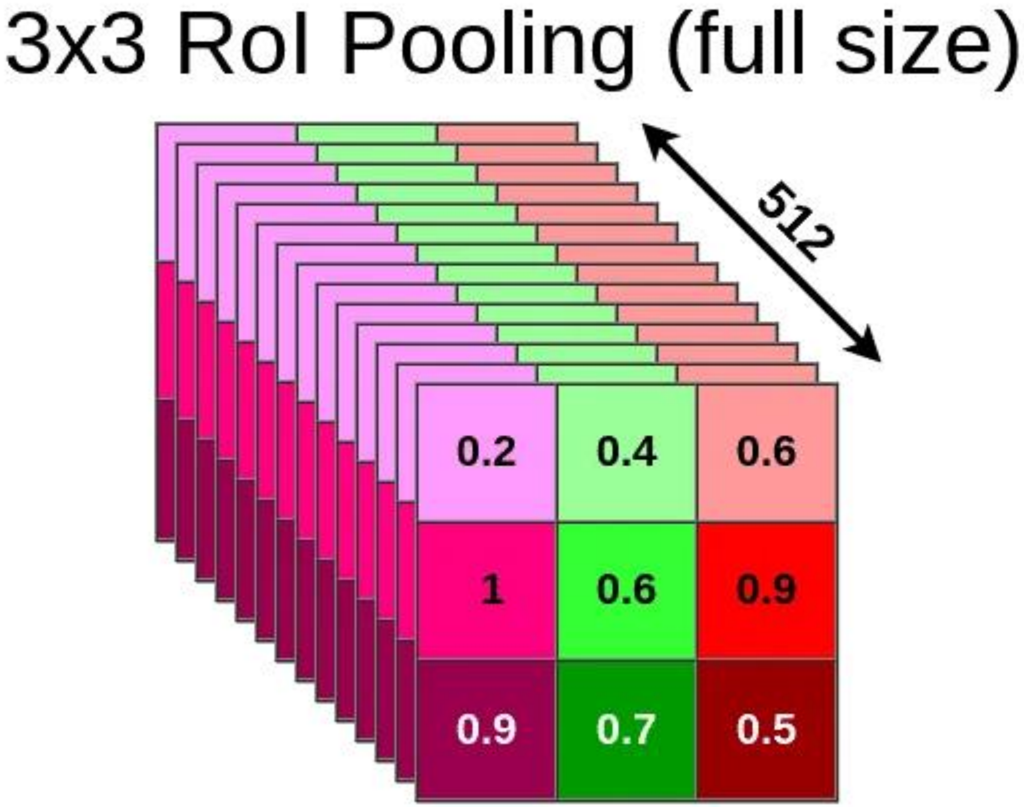
4x6 RoI (Lost Data)

0.1	0.2	0.3	0.4	0.5	0.6
1	0.7	0.2	0.6	0.1	0.9
0.9	0.8	0.7	0.3	0.5	0.2
0.2	0.5	1	0.7	0.1	0.1

ROI-Align

https://tinyurl.com/y6xpm24d

RoI Pooling on Input 4x6x512 matrix (ROI volume)

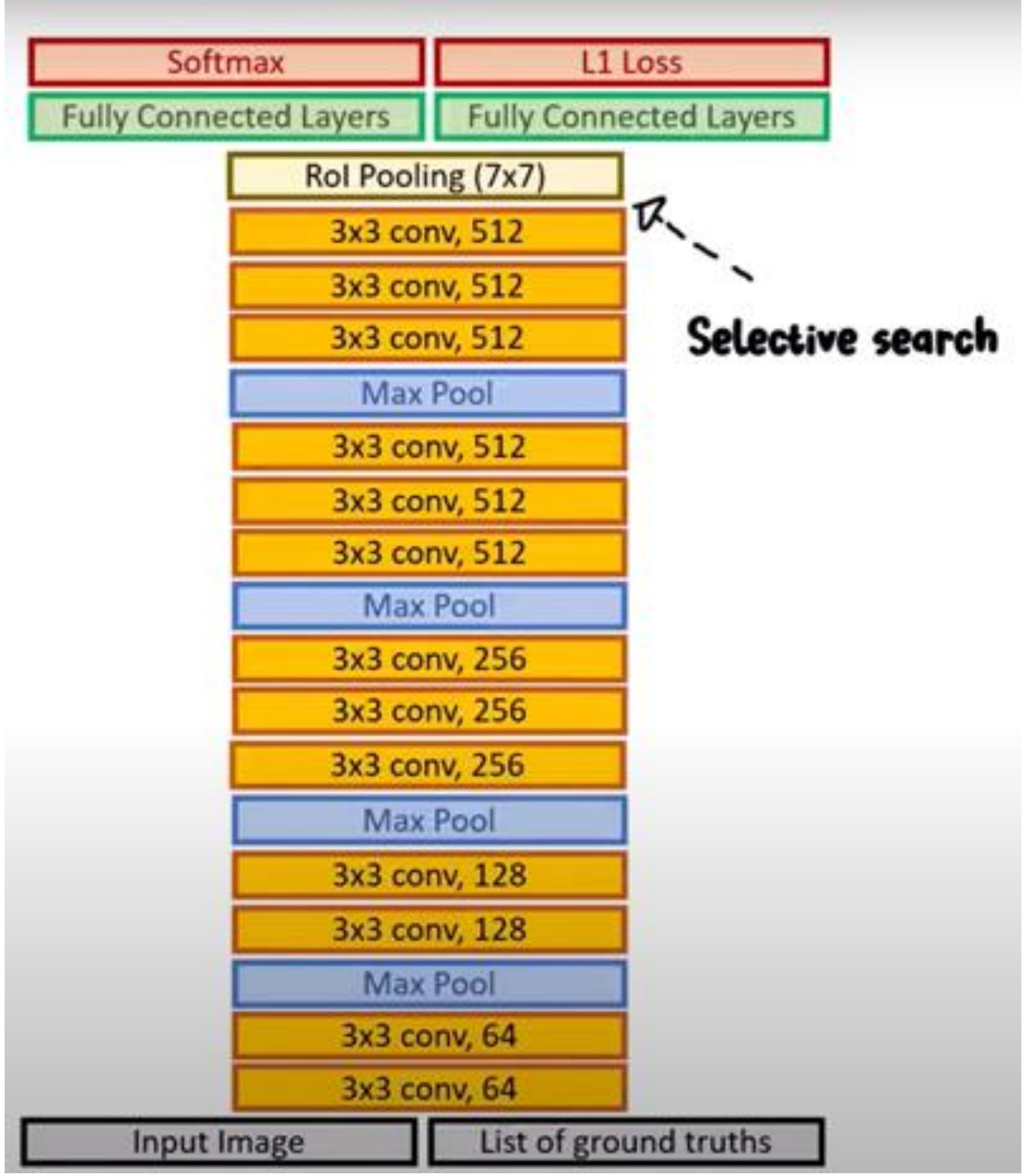


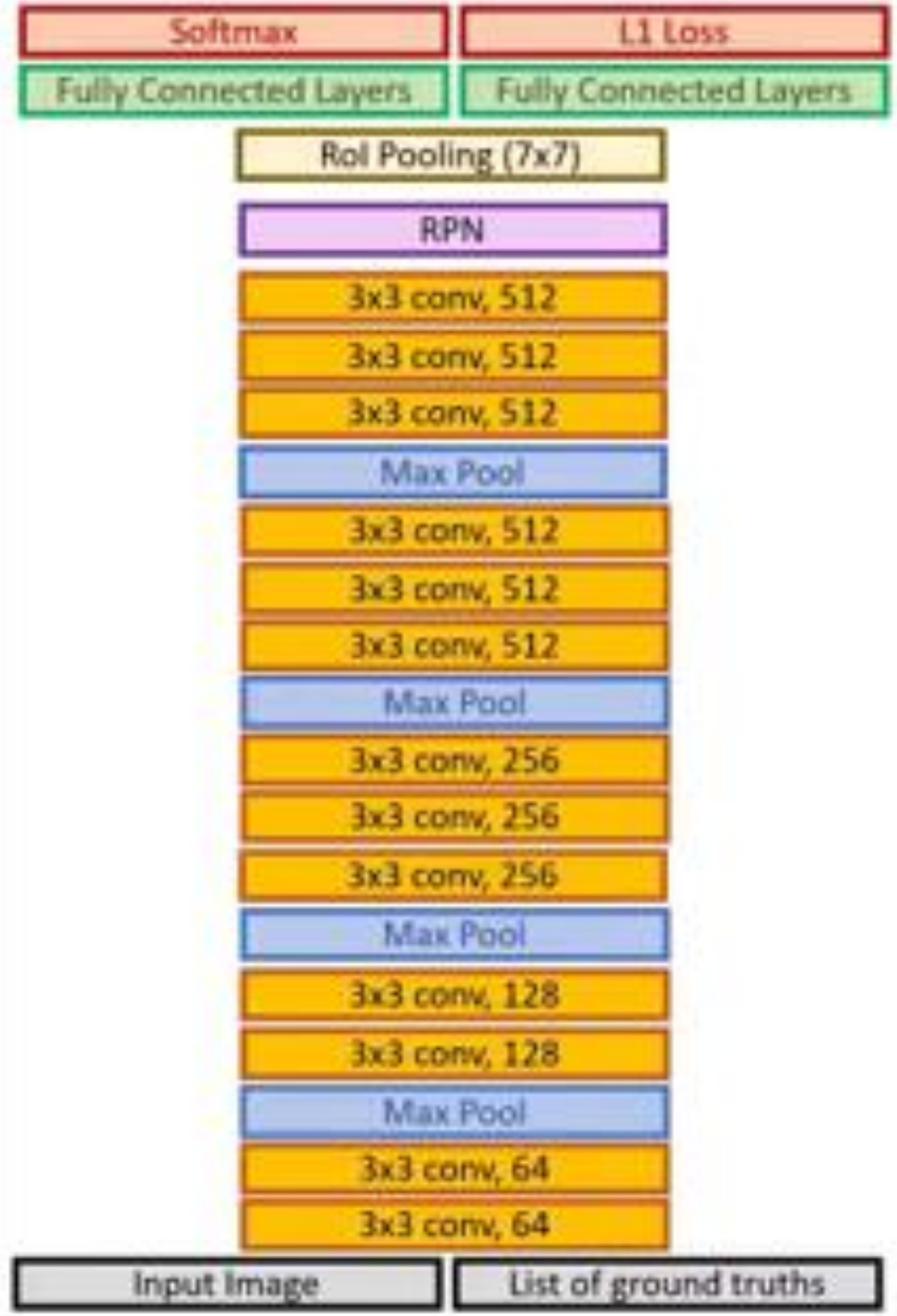
Full-size pooling output

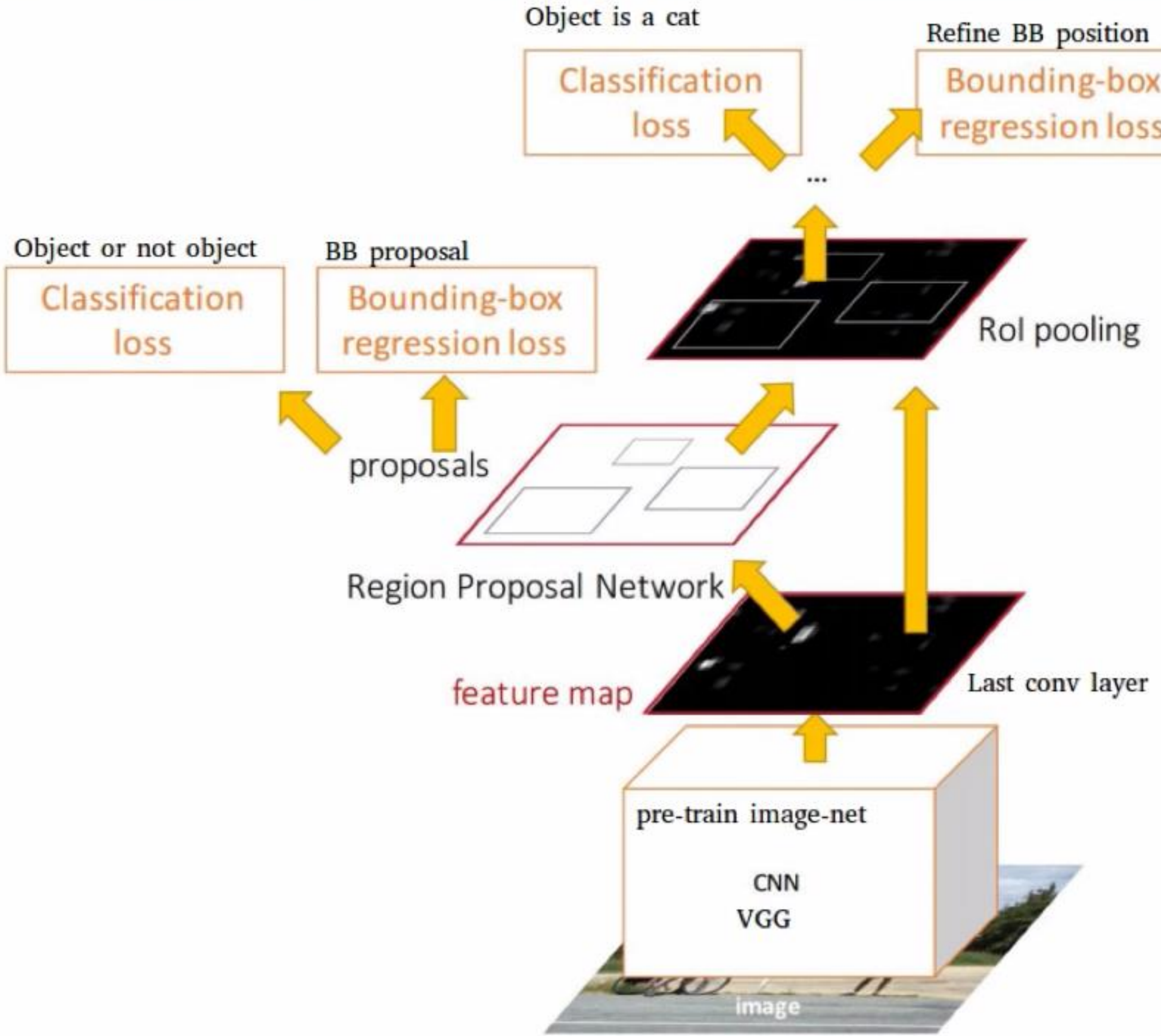
- The same process is applied to **every single RoI** from our original image so in the end, we might have many **of 3x3x512 matrixes**

Faster RCNN

Model	Test Time per Image (seconds)
R-CNN	45-50 seconds
Fast R-CNN	2-3 seconds
Faster R-CNN	0.2-0.3 seconds

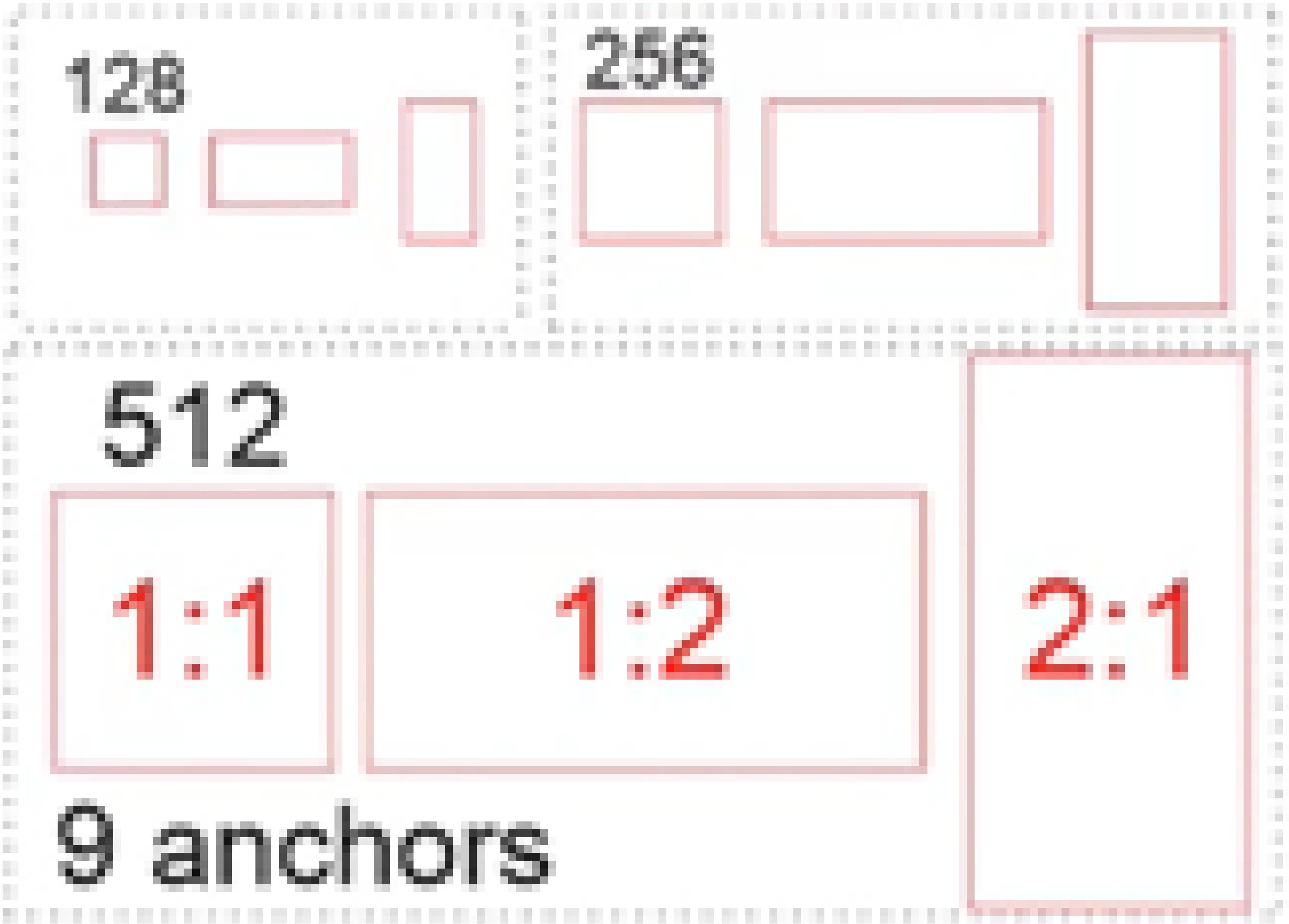

Softmax
L1 Loss
Fully Connected Layers
Fully Connected Layers
RoI Pooling (7x7)
Selective search
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Max Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Max Pool
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
Max Pool
3x3 conv, 128
3x3 conv, 128
Max Pool
3x3 conv, 64
3x3 conv, 64
Input Image
List of ground truths


Softmax
L1 Loss
Fully Connected Layers
Fully Connected Layers
RoI Pooling (7x7)
RPN
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Max Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Max Pool
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
Max Pool
3x3 conv, 128
3x3 conv, 128
Max Pool
3x3 conv, 64
3x3 conv, 64
Input Image
List of ground truths



Region Proposal Network

- **3×3 Conv Kernel on Feature Map**
- **Receptive Field Concept**
- **k Anchors per Location**
- **IoU Threshold for Positive Anchor**



128
256
512
1:1
1:2
2:1
9 anchors

PyTorch's torchvision library includes a pre-trained Faster R-CNN model:
python

```
import torchvision
from torchvision.models.detection import fasterrcnn_resnet50_fpn

# Load the model
model = fasterrcnn_resnet50_fpn(pretrained=True)
model.eval()
```

```python
import cv2
import numpy as np

import torch
import torchvision
from torchvision.transforms import functional as F
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.patches as patches


from google.colab import drive
drive.mount('/content/drive')

# Load image
image_path = "/content/drive/MyDrive/cat.png"
image = Image.open(image_path).convert("RGB")
# Transform image to tensor
img_tensor = F.to_tensor(image)


# Load pre-trained Faster R-CNN model
model =
torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True
)
model.eval()


# Run inference
with torch.no_grad():
    prediction = model([img_tensor])[0]
```

```python
# Draw boxes
fig, ax = plt.subplots(1, figsize=(12, 8))
ax.imshow(image)

for box, score, label in zip(prediction["boxes"],
prediction["scores"], prediction["labels"]):
    if score > 0.5:  # confidence threshold
        x1, y1, x2, y2 = box
        width, height = x2 - x1, y2 - y1
        rect = patches.Rectangle((x1, y1), width, height,
linewidth=2, edgecolor='r', facecolor='none')
        ax.add_patch(rect)
        ax.text(x1, y1, f'{label.item()} ({score:.2f})',
color='white',
                bbox=dict(facecolor='red', edgecolor='red',
boxstyle='round,pad=0.2'))

plt.axis('off')
plt.show()
```