Fully Convolutional Networks

Team:
Sri Iyengar (si2468)
Radhika Patel (rpp2142)
Anushka Pachaury (ap4617)

EECS E6691 Advanced Deep Learning, 2025 Spring

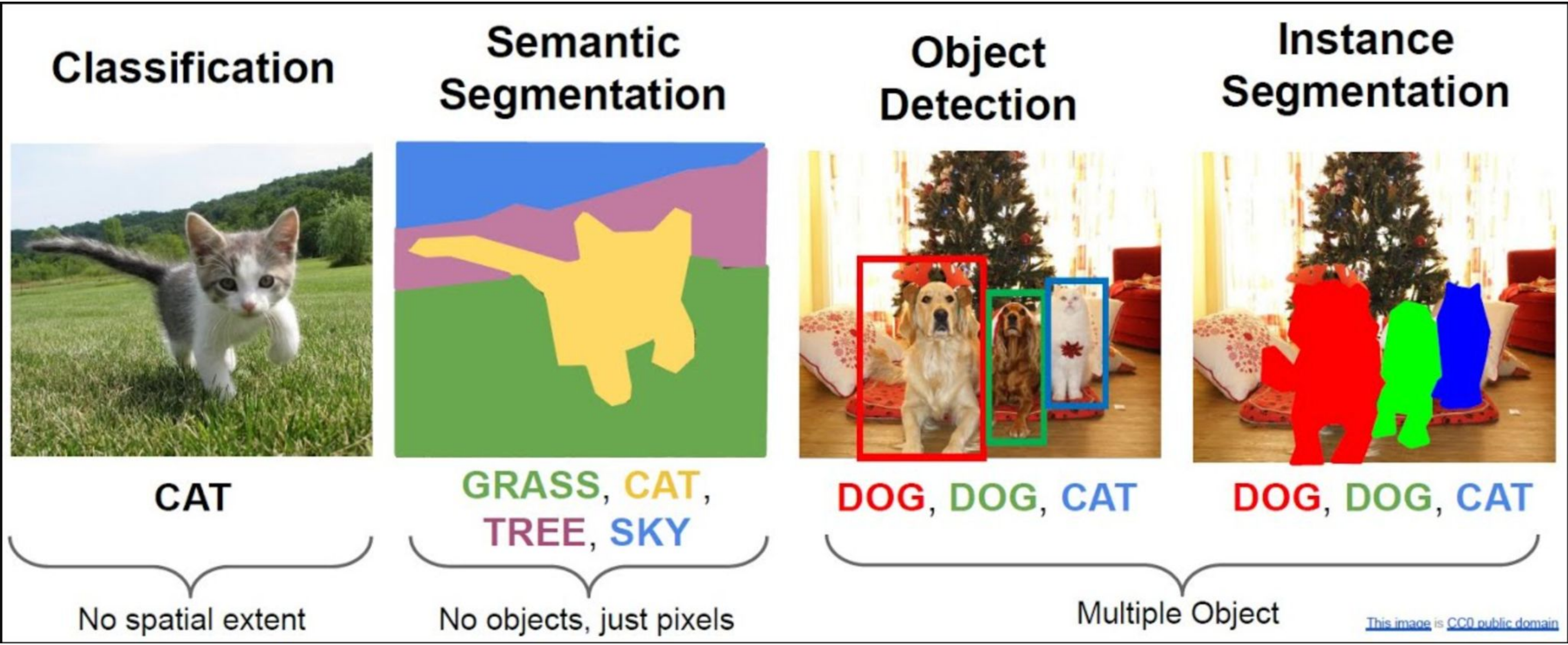
Outline of the Presentation

- Problem Definitions
- Semantic Segmentation
  - Motivations for FCN
  - FCN Architecture
  - UNet
- Object Detection
  - Model History (R-CNN and successors)
  - RPNs
  - R-FCNs

Object Detection Vs Segmentation

Objective	Classification	Semantic Segmentation	Instance Segmentation	Object Detection
Output Type	Per-image class label	Per-pixel class label	Per-pixel class and instance labels	Set of bounding boxes with class labels
Differentiates Multiple Objects	No	Only if different classes	Yes	Yes
Overlapping Objects	Not handled	Merged into one group if same class	Handled via per-instance masks	Handled via per instance bounding boxes
Use case Example	Is this a dog or a cat?	Which pixels belong to a dog/cat?	Which pixels belong to *which* dog/cat?	What are all dogs/cats in this image and where are they?

Comparisons



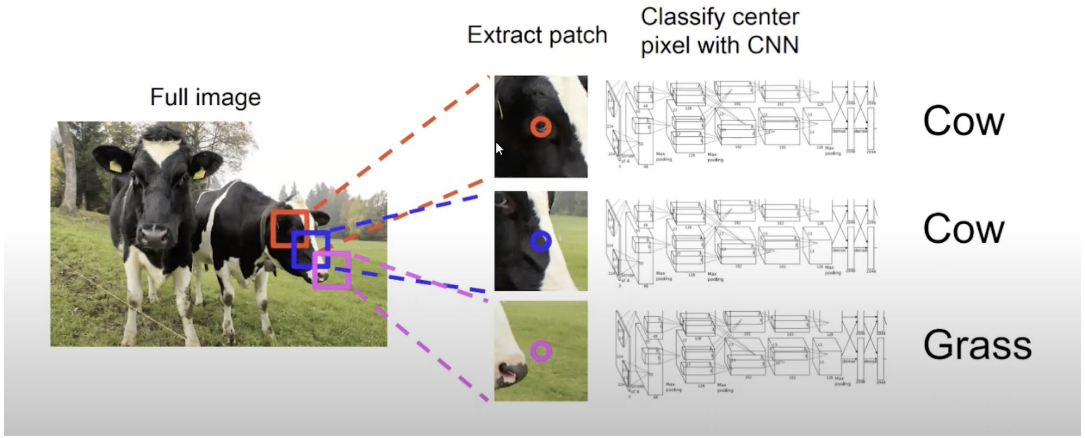
FCNs for Semantic Segmentation

Why not traditional CNNs?

1. Maintaining spatial resolution
    - Semantic segmentation requires per-pixel classifications
    - CNNs lose track of “pixels” as spatial resolution is reduced by the pooling layers
    - Dense layers have lost lose spatial information
2. Output size
    - CNN: $X \in \mathbb{R}^{HxWxD} \rightarrow F(X) \in \mathbb{R}^{C}$, C = number of classes
    - HxWxC output for segmentation

CNN with sliding windows

- Choose some window size (e.g. 32x32, 64x64)
- Consider all possible windows of that size (potential padding)
- Treat each window as input to a CNN to **classify the middle pixel**



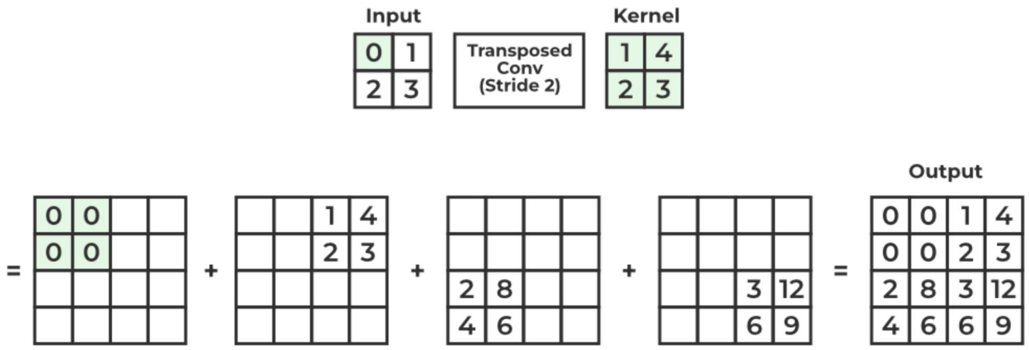
- Inefficient (redundant computations)
- Fixed window size?
- Loss of global context within each prediction

FCN motivation

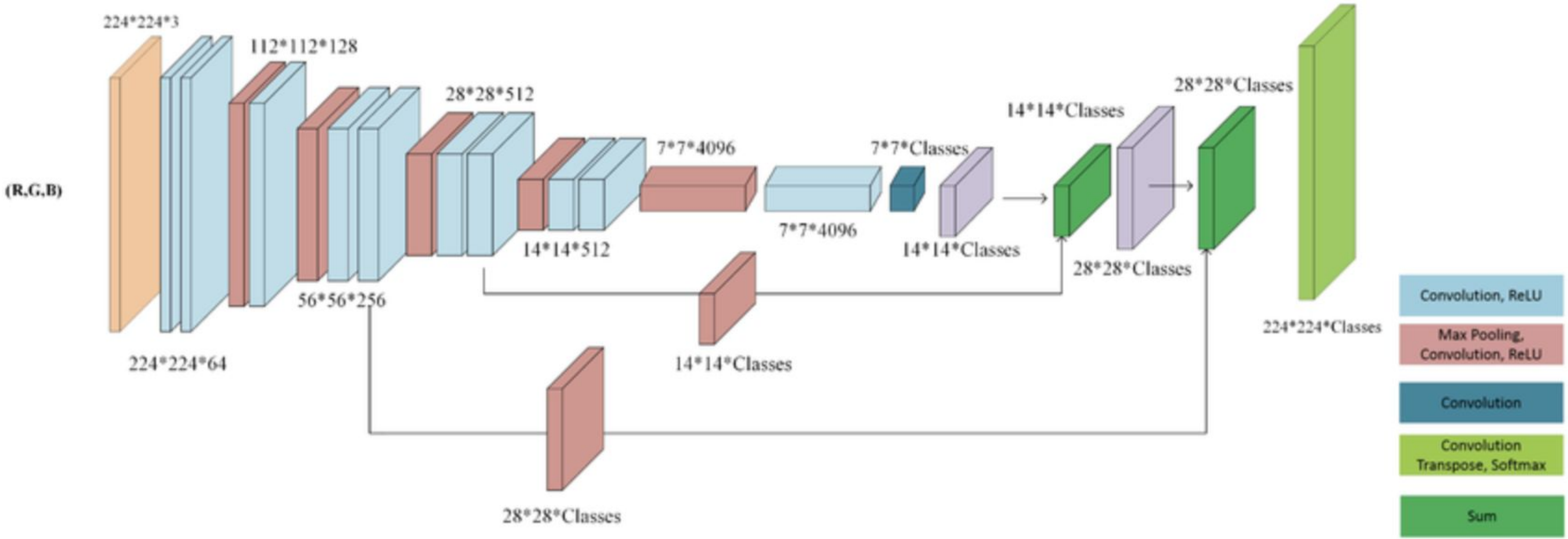
- Maintain spatial awareness
- Dense predictions per pixel instead of one label
- Adaptability with any input size
  - Do standard CNNs work with any input size?
- Feasible end-to-end training
  - Must be efficient

FCN Architecture

- Alter traditional CNN
    - “Fully” Convolutional Network
    - 1x1 convolution
    - Replace dense layers with more (slightly different) convolutional layers
        - Transposed convolutions - upsamples feature maps back into pixel-level interpretability



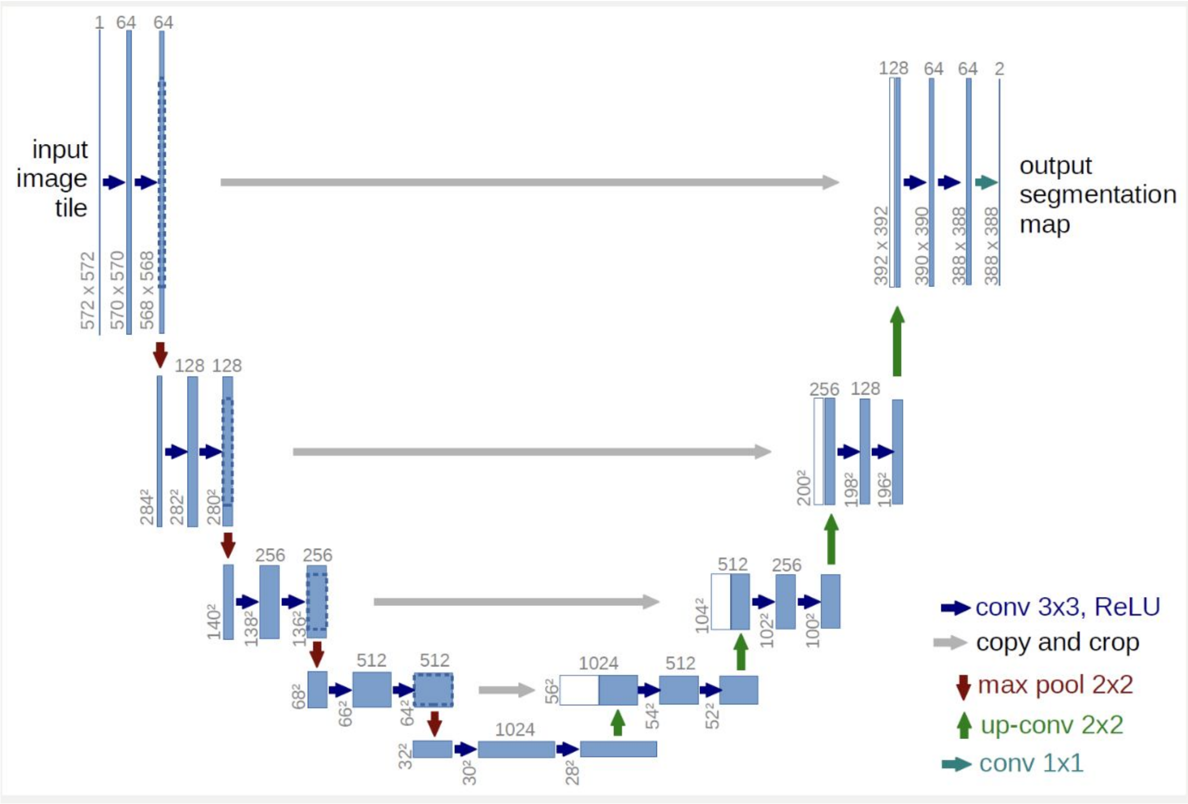
FCN Architecture



UNET Motivation

- FCNs suffer from coarse segmentation
  - Continuous downsampling
  - Losing fine-grained spatial details
- Use skip connections
  - Use inputs from each downsampling operation directly
  - Deep CNN layers capture semantic meaning
  - Early CNN layers capture edges and fine textures without global context
  - Combine both with skip connections

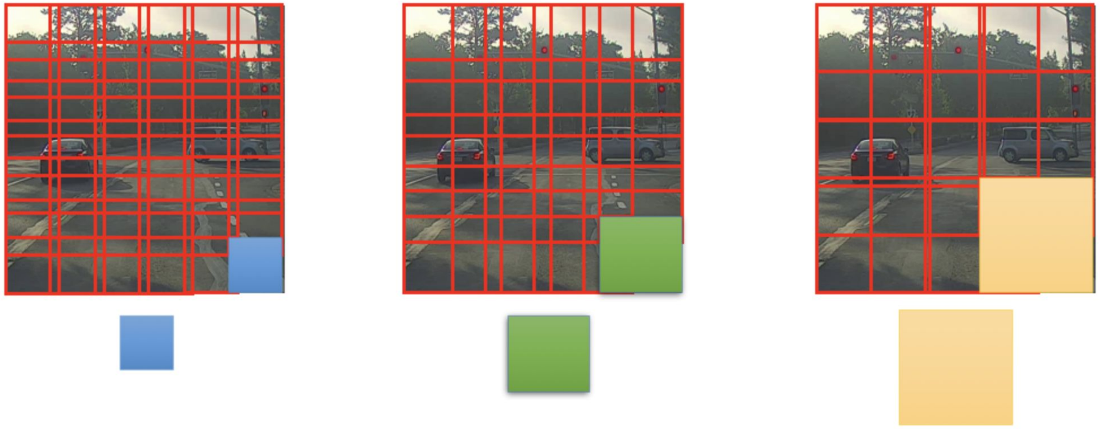
UNET Architecture



Object Detection

Again - CNN with sliding windows

- Choose a sufficiently exhaustive set of bounding boxes in the image (varying size, position)
- Classify each of the windows using a CNN classifier
- If window shows significant classification, accurate bounding box found



- Inefficient with smaller strides
- Large strides miss out on well-aligned bounding boxes

R-CNN Family: The General Architecture

Family Members: R-CNN, Fast R-CNN, Faster R-CNNs

1) CNN Feature Extraction
   - Extract feature maps from final layer of fine tuned CNN backbone
   - CNN layers capture hierarchical image features from low-level edges to high-level object shapes

2) Region Proposal Generation - Identify candidate regions and classify if they contain objects or not
   - Sliding Windows
   - Selective Search
   - Edge Boxes
   - **RPNs**

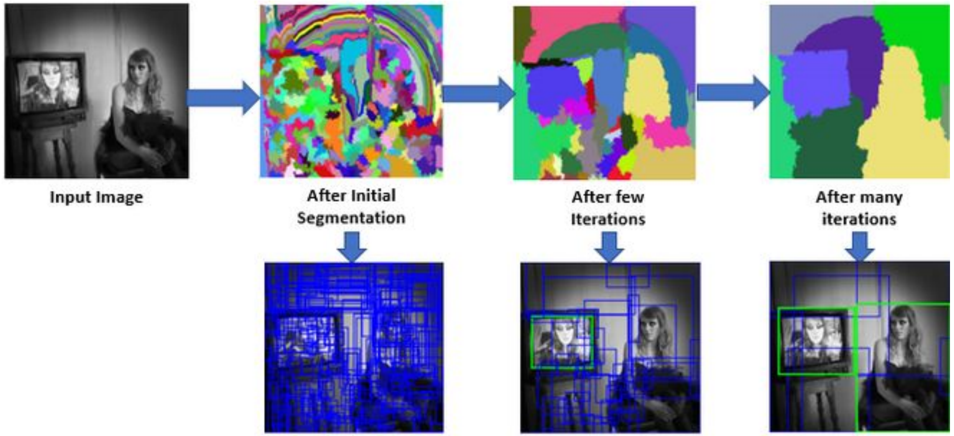
R-CNN Family: The General Architecture

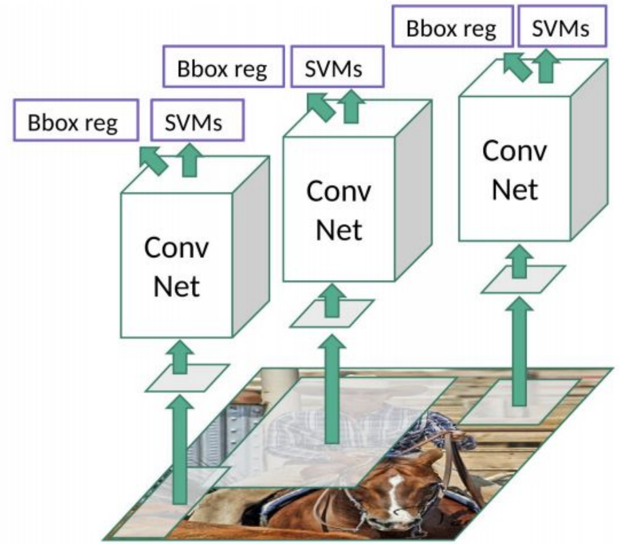
3) Classification and Bounding Box Regression
- Used Fully Connected Sibling Layers/Mechanisms
- Classify extracted region features to determine object category
- Use regresser to generate bounding box coordinates
- Combine classification scores + bounding box predictions for final detection

Models in R-CNN family perform all 3 steps, but differ significantly in how steps are staged an integrated

R-CNNs

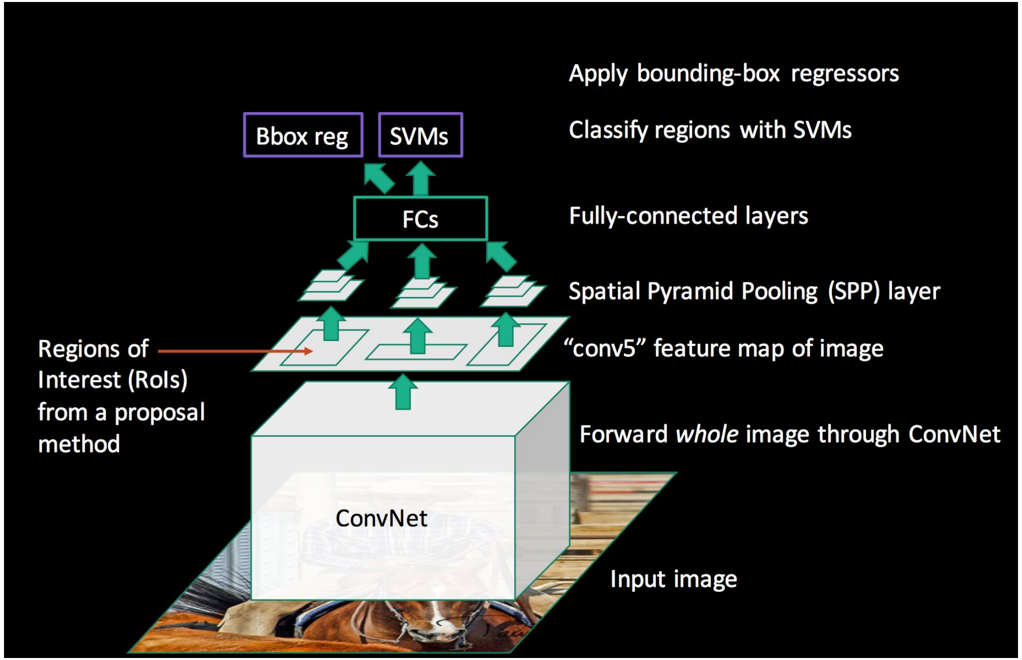
- Used Selective Search algorithm to generate proposal (~2000 per image)
- All separately fed into pretrained AlexNet, later fine tuned for Image Classification Task
- Discard Classification Layer (Softmax + Output) to get the feature vector representation for each proposal
- Classify using the K SVM Classifiers
- Separate BBox Regressor
- Really slow inference time (~47 seconds)





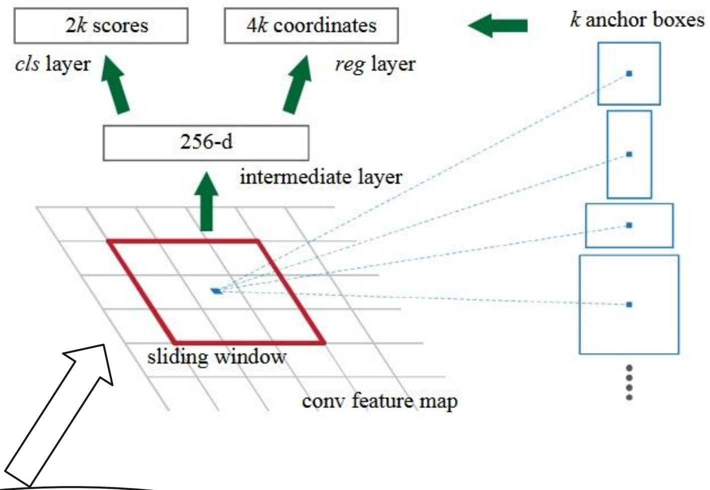
Fast R-CNNs

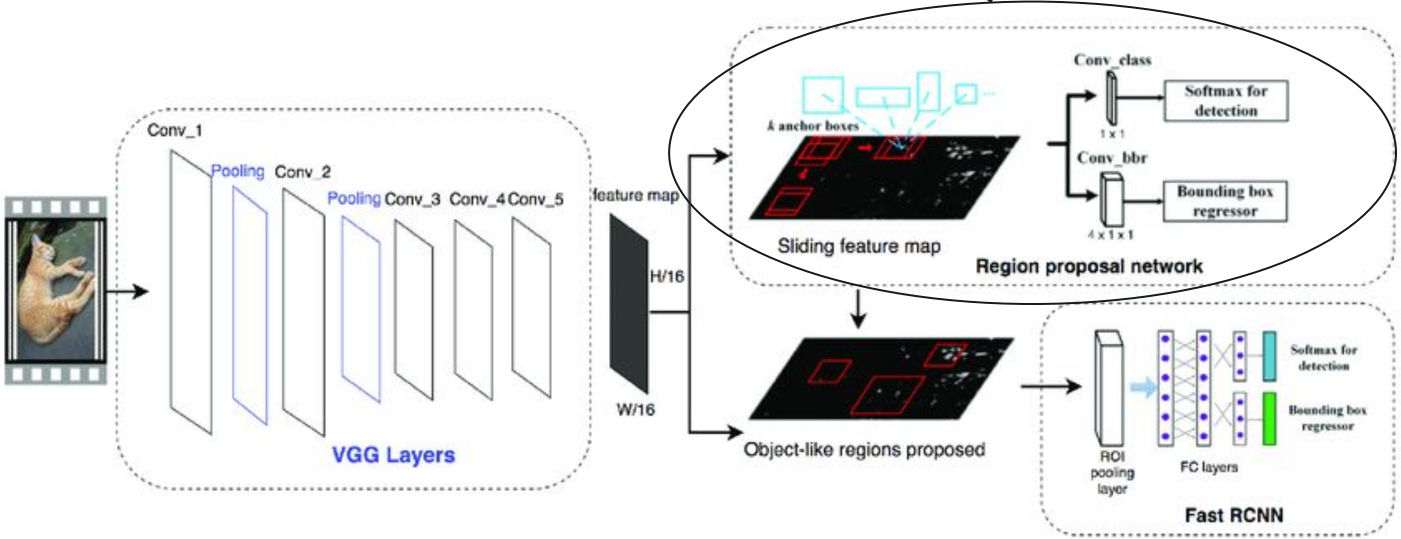
- Used single image as an input to ConvNet
- Selective Search used for region proposal generation, but mapped onto the final feature map
- Added the **ROI Pooling Layer -** Divides each feature map for each region proposal into n regions uniformly, perform max pooling over each region to get n values
- Flattened ROI outputs passed to sibling FC Layers (classification, BBox regression)
- Single training pipeline unlike R-CNN
- Inference Sped up to 100-150x



Faster R-CNNs

- Introduced the RPN to **learn** Region Proposals (replacing Selective Search) using Anchor Boxes





Region Proposal Networks (RPNs)

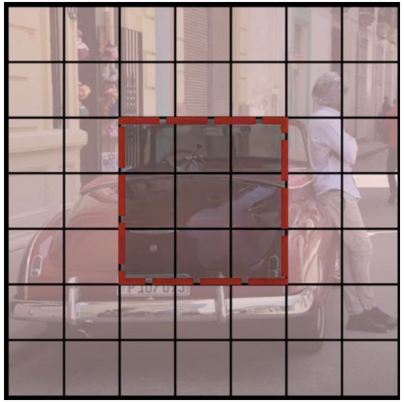
Motive:

- Remove region proposal bottleneck from SS
- Learn regions that possibly contain objects instead of statically generating them

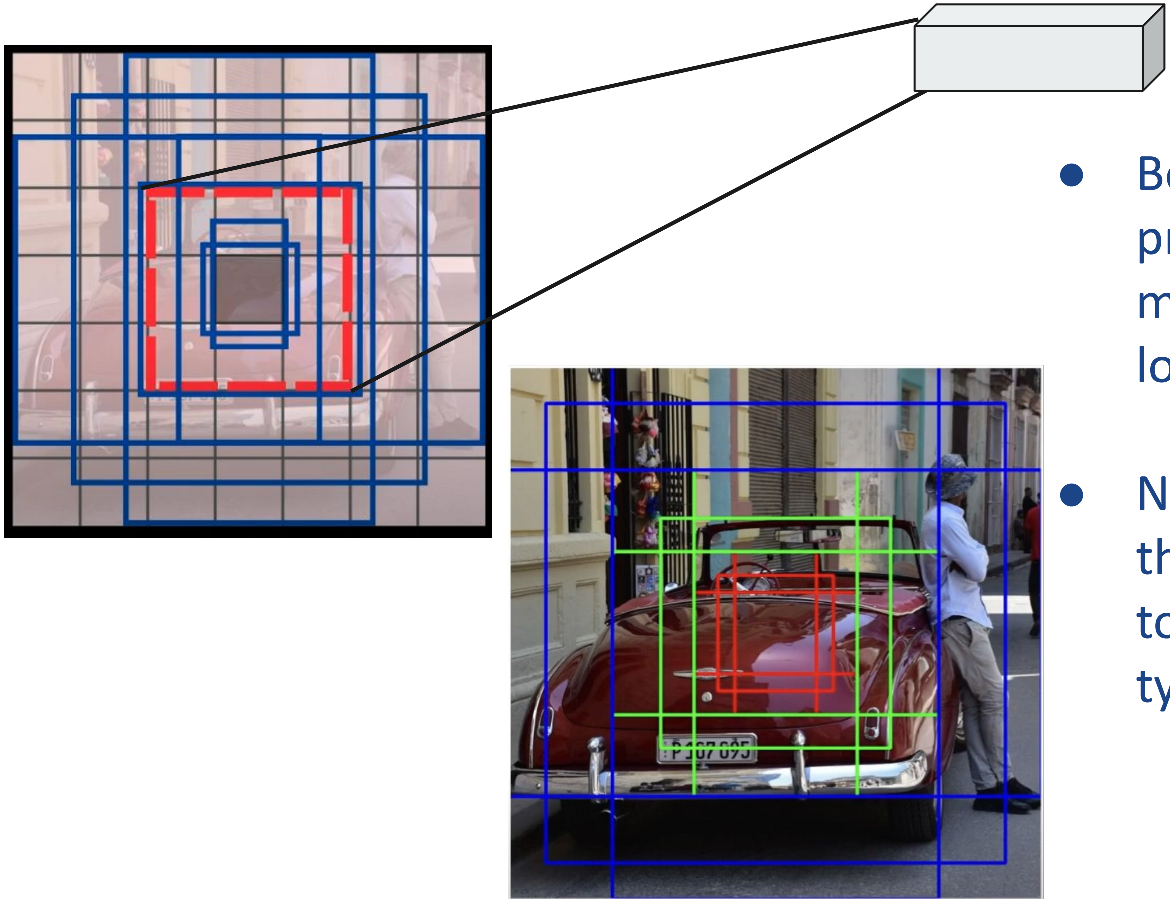
How do you train such a Network?

- Instead of extracting region proposal directly from an image itself, learn them by using a sliding window over the feature map
    - Convolve feature map generated by last conv layer (eg 3*3)
    - Will capture spatial information for each fixed size frame on feature map
- Problem:

  This still only captures features for FIXED aspect ratios and scales



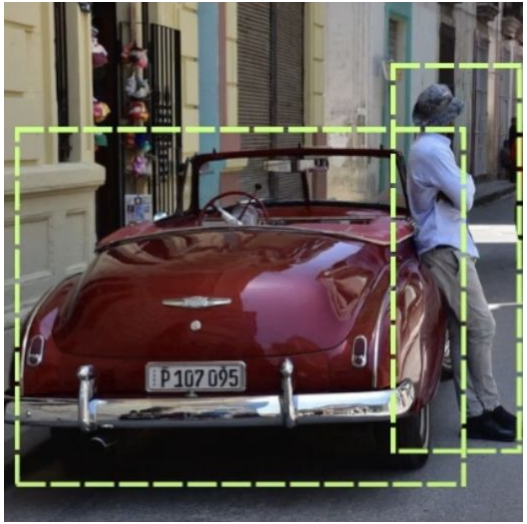
Anchor Boxes



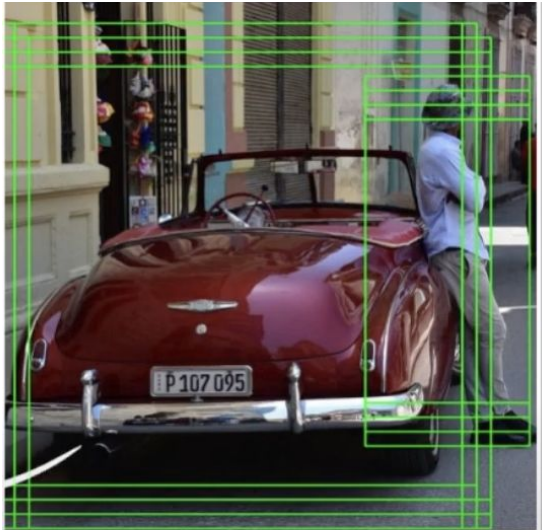
1 x 1 x C

- Bounding boxes with predefined aspect ratios mapped for each spatial location (k=9 is standard)
- Network learns to associate the features for each window to all defined anchor box types

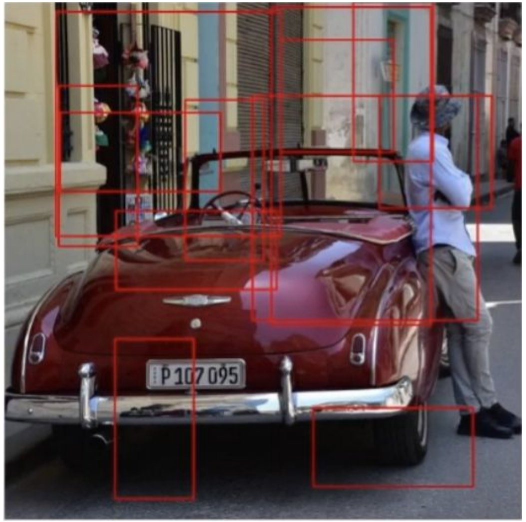
RPN Training



MAKING OUR TRAINING SET



Foreground if >= 0.7 IOU with any ground truth box



Background if < 0.3 IOU with ALL ground truth boxes

Anchor boxes for each spatial location are mapped onto original image => IOU calculated relative to ground truth box on image => Corresponding feature maps sampled

RPN Training

INPUTS: (X_{feat}, (foreground vs background, (dx, dy, dw, dh))

GOAL:

- Classify object in proposed region as foreground vs background
- Learn transformation t to better align proposed anchors to ground truth bounding box

Targets:

$$t_x = (g_x - p_x) / p_w$$
$$t_y = (g_y - p_y) / p_h$$
$$t_w = \log(g_w / p_w)$$
$$t_h = \log(g_h / p_h)$$

Learned By:

$$\mathbf{t} = W * X_{feat} + b$$

RPN Training

$$L(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

Bounding Box Loss

Classification Loss

$L_{reg} = \text{smooth}_{L_1}(t_i - t_i^*)$ $\quad \text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$

L_{cls} := Cross Entropy Loss

N_{reg} = No of anchor locations

N_{cls} = No of anchors in batch

p_i is either 0/1

t_i = transformation for anchor box i

RPN Training

Outputs:

- Softmax classification Layer: 2*k classification scores
- 4*k refinement scores (t_x, t_y, t_w, t_h)

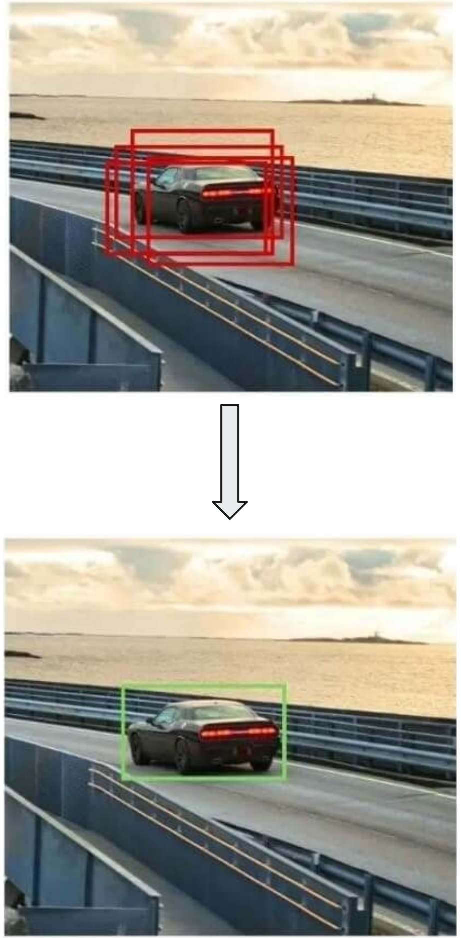
INFERENCE:

$$g_x^* = p_w t_x + p_x$$
$$g_y^* = p_h t_y + p_y$$
$$g_w^* = p_w \exp(t_w)$$
$$g_h^* = p_h \exp(t_h)$$

Non-Maximum Suppression

Selecting best bounding box amongst a set of overlapping boxes

- For each class, sort boxes by confidence score
- Remove boxes with IOU < pre-defined threshold
- Start with the first box b_i that has highest confidence
    - Calculate IOU of b_i with every other b_j
    - If IOU(b_i, b_j) > IOU threshold, discard box with smaller confidence
- Repeat



Introduction to R-FCNs

Motive:

- Improve object detection efficiency and speed
- Address the bottleneck of fully connected layers in models like Faster R-CNN

What is R-FCN?

- A fully convolutional object detection model
- Uses **Position-Sensitive RoI (RSRoI)** pooling for efficiency region classification

Key benefits:

- Faster inference with minimal loss in accuracy
- Preserves spatial information crucial for accurate localization

Challenges with Previous Methods

Faster R-CNN Limitations:

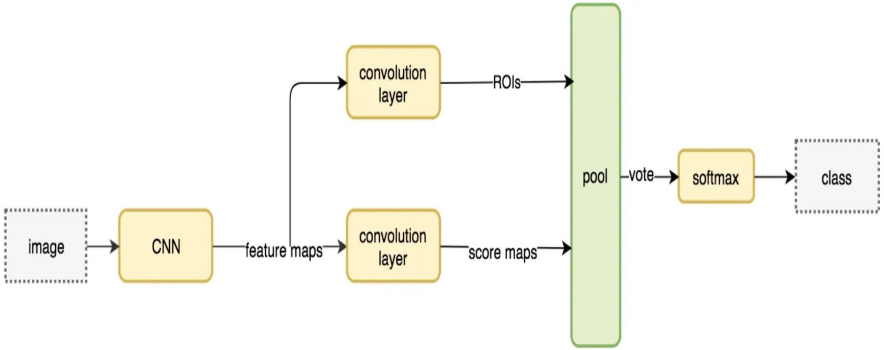
- Uses fully connected layers for each Region of Interest (RoI)
- High computational cost with many RoIs
- Redundant computation slow down detection

R-FCN Solution:

- Fully convolutional layers shared across the entire image
- **No fully connected layers** at the end of the pipeline
- Efficient RoI processing using **position-sensitive score maps**

Overview of R-FCN Architecture

- **R-FCN Goal:** Efficient object detection balancing accuracy and inference speed.
- **Main Components:**
    - **Backbone Network:** Extracts features from the input image
    - **Two parallel branches:**
        - **Branch 1:** Region Proposal Network (RPN) → Produces RoIs (Regions of Interest)
        - **Branch 2:** Position-Sensitive Score Maps → Enables accurate classification and localization
    - **Final Steps:** PSRoI pooling → Aggregation → Softmax for classification



Backbone Architecture of R-FCNs

- **Backbone Used:** Typically ResNet-101 or ResNet-50
- **Input Example:** a 600 x 800 x 3 RGB image
- **Feature Map Generation:**
    - After our initial convolutions and pooling layers, feature map dimensions reduce
    - **Example Output from Backbone:** 38 x 50 x 1024 feature map
- This feature map is used by both branches (RPN and PS Score Maps)

Region Proposal Network (RPN)

- Identifies regions in the image that might contain objects(not their classes yet)
- Instead of checking every possible location, the RPN quickly narrows down potential object areas
- How it works:
  - **Input**: 38 x 50 x 1024 feature map
  - **3 x 3 Convolution:** Captures local context ⟶ 38 x 50 x 512
  - **Two Heads:**
    - Object Classification
    - Bounding Box Regression

Position-Sensitive Score Maps (PS Score Maps)

- Traditional Methods lose information about where features come from. PS score maps retain spatial awareness, crucial for recognizing object parts
- Indicate how likely each pixel belongs to a specific object park
- **How are they made:**
  - Use a 1 x 1 convolution on the 38 x 50 x 1024 feature map
  - Create k x k x C maps ⟶ k = 3, C = 20 classes ⟶ 180 maps
- **Choosing k:**
  - k = 3 is equivalent to dividing objects into 9 parts
  - This is a hyperparameter:
    - Larger K ⟶ finer detail but more computation

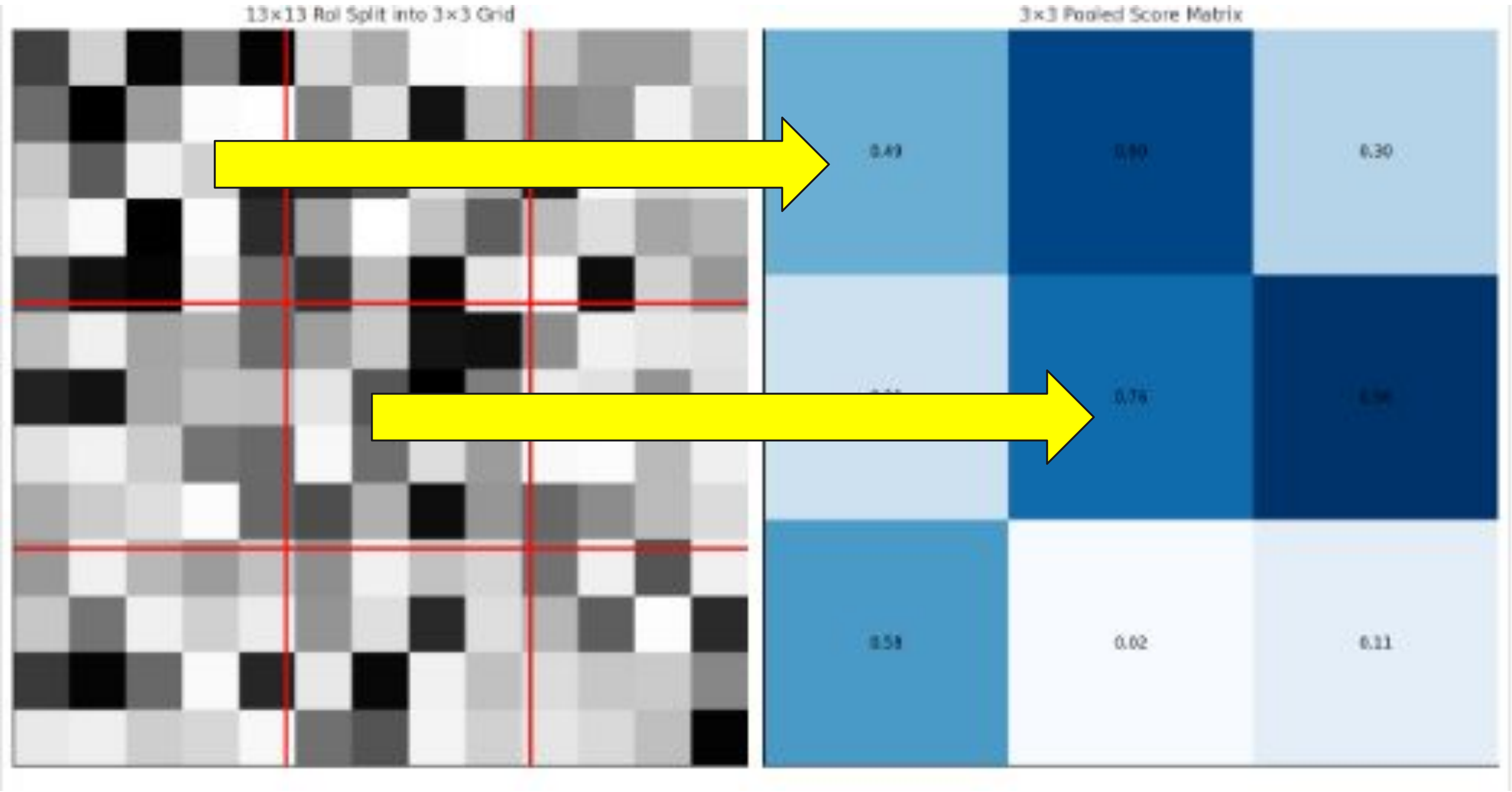
PS Score Maps Example

- **Dog Class (C=1)**: 9 maps correspond to parts like:
  - Map 1: Dog’s head (top-left)
  - Map 5: Dogs body (center)
  - Map 9: Dog’s tail (bottom-right)
- **Pixel Values:**
  - **High value pixel:** High chance that area is the dog’s head
  - **Low value pixel:** Low likelihood
- This prevents confusion between similar objects (eg dog vs cat) by checking part locations

Generate 9 score maps

Relationship between RoIs and PS score maps

- **RoIs:** Show where to look
- **PS Score Maps:** Show what is there and which part of the object it is
- **Process:**
  - RoIs are projected onto the 38 x 50 x 180 maps
  - Each RoI is split into a 3 x 3 grid if k = 3
  - Each grid cell maps to a specific PS map (top left grid $\longrightarrow$ top left PS map)



13×13 RoI Split into 3×3 Grid
3×3 Pooled Score Matrix
0.49
0.30
0.02
0.11

PSRoI Pooling

- A way to get a fixed size representation from variable-sized RoIs while keeping positional information
- Standard pooling loses location details. PSRoI pooling preserves them, crucial for accurate classification
- **Process:**
  - Select the correct PS map for each RoI grid cell
  - Extract overlapping pixels
  - Apply average pooling to get one score per cell

$$P_{ij} = \frac{1}{|R_{ij}|} \sum_{(x,y) \in R_{ij}} S_{ij}(x, y)$$

P_{ij} := ROI value pooled for cell (i,j)
S_{ij} := feature values for pixels in cell (i,j)
R_{ij} := # of pixels in ROI grid cell

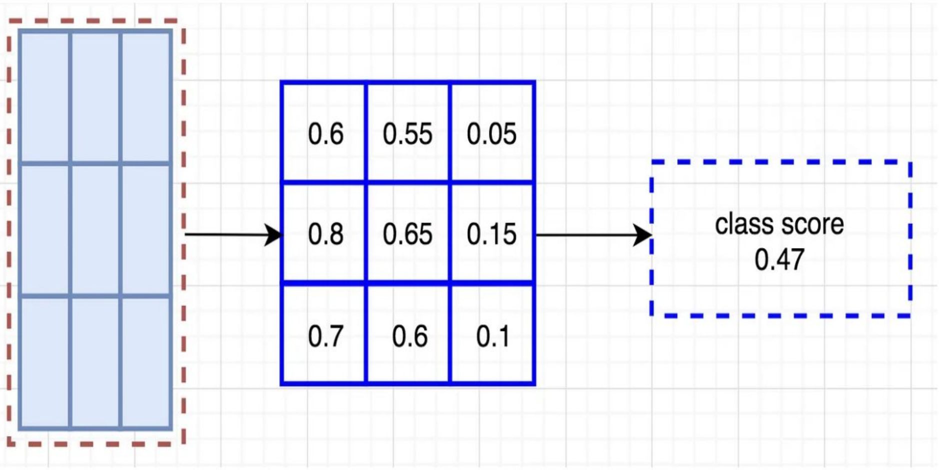
Aggregation and Softmax

- After pooling, we have 9 scores for each class:
- Aggregation: Combine them into a single class score:

$$\mathrm{P}_c = \frac{1}{k^2} \sum_{i=1}^{k} \sum_{j=1}^{k} P_{ij}^{c}$$

- Softmax: converts scores into probabilities and helps pick the correct class based on the highest probability

$$\hat{y}_c = \frac{e^{P_c}}{\sum_{c'} e^{P_{c'}}}$$



ROI pool

P_{ij} := ROI value pooled for cell (i,j)
k := ROI dimensions (cells)
P_c := score for a single class/ROI pair

```
feature_maps = process(image)
ROIs = region_proposal(feature_maps)
for ROI in ROIs
    patch = roi_pooling(feature_maps, ROI)
    class_scores, box = detector(patch)
    class_probabilities = softmax(class_scores)
```

Fast R-CNN and R-FCN pseudocode comparison

```
feature_maps = process(image)
ROIs = region_proposal(feature_maps)
score_maps = compute_score_map(feature_maps)
for ROI in ROIs
    V = region_roi_pool(score_maps, ROI)
    class_scores, box = average(V)                   # Much simpler!
    class_probabilities = softmax(class_scores)
```

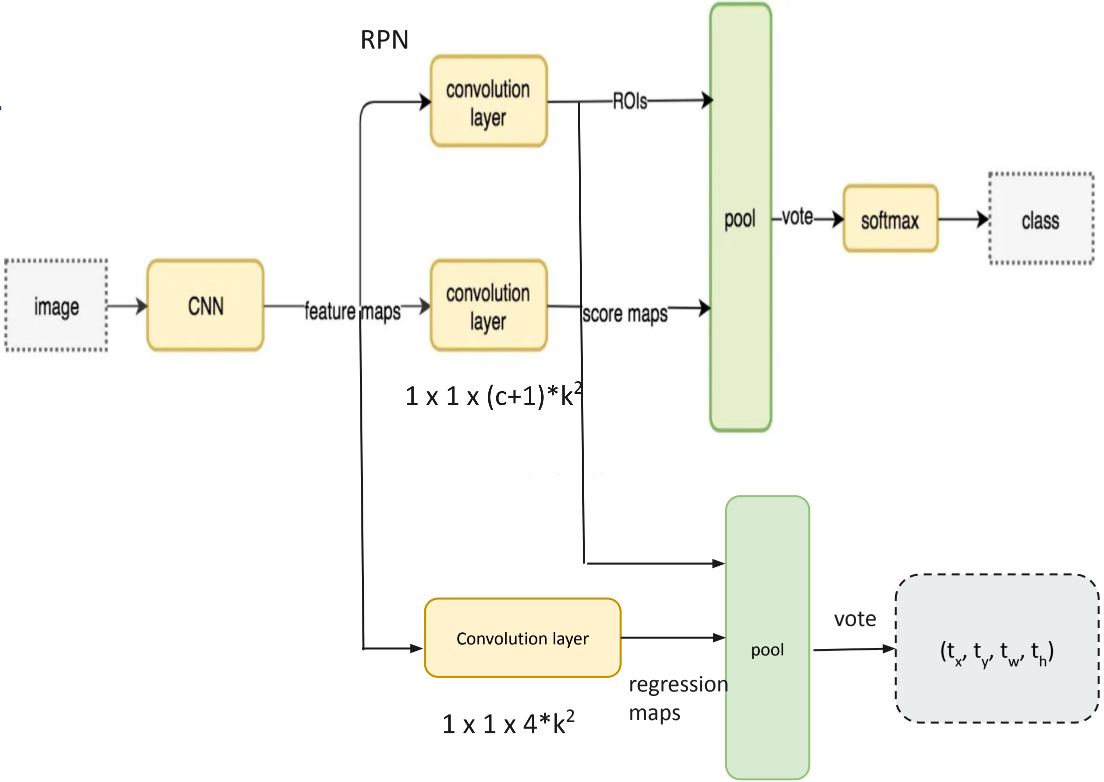
Position-Sensitive Regression Mapping

How are they made:

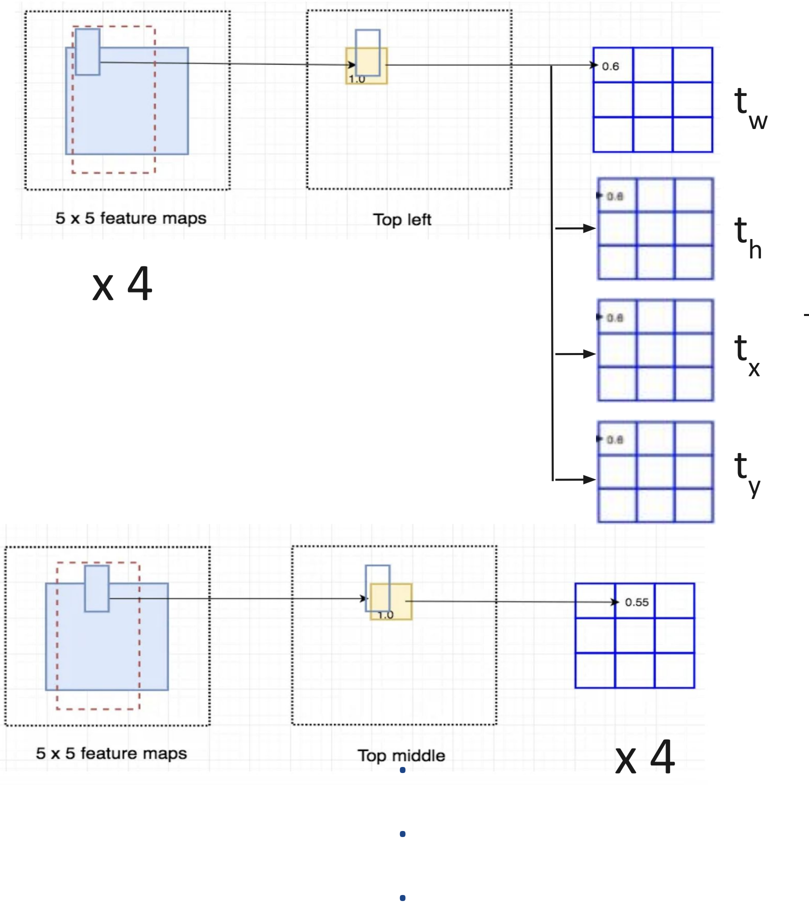
Feed same feature map from backbone's final convolution layer

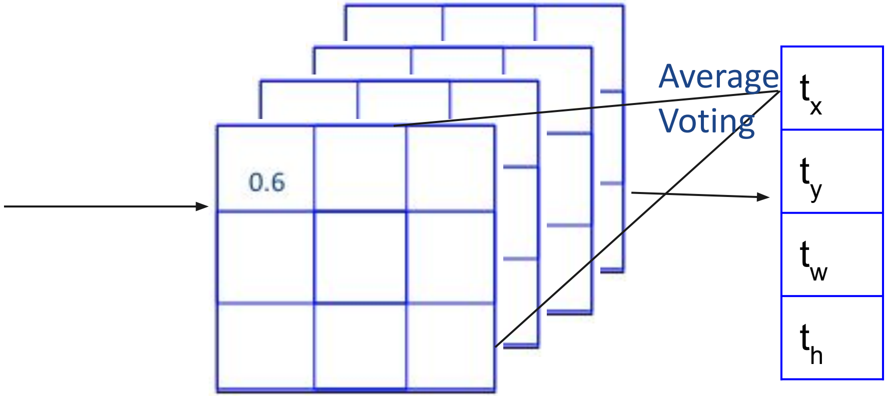
Class agnostic mapping:
(conventional, from paper)
Create $4*k^2$ with k = 3 ⟶ 36 maps

Class-specific mapping:
$4*k^2*C$ with k = 3, C=20 ⟶720 maps



Position Sensitive Regression - RoI Pooling





- Map pixels from ROI onto PS Regression Map ($4*k^2$ channels)
- 4 regression maps per feature map cell
  - → 4 pooled values per cell
  - → 4 ROI regression pool maps
- Same score calculations as positive sensitive score maps

Position Sensitive Regression ROI - The Intuition

Each cell in Regression PSRoI is learning how much the bounding box should be adjusted when crucial points of the object appear near that cell / spatial position

- Cells corresponding to corners specialize in refining bbox edges, while center cells might focus on object centers, sizes, or aspect ratios.
- Different parts of an object’s features tell you how much object boundary should shift and scale
- Eg, 9 values in the t_w channel from the 3x3x4 pooling tell you how much the width needs to be offset relative to each of the k spatial locations

Bounding Box Refinement

Infer final bounding boxes using offsets given by final 4-d vector

Regression Head: Predicts adjustments: (t_x, t_y, t_w, t_h)

- **Equations:**

$$\hat{x} = t_x \cdot w + x,$$
$$\hat{y} = t_y \cdot h + y,$$
$$\hat{w} = e^{t_w} \cdot w,$$
$$\hat{h} = e^{t_h} \cdot h.$$

Finally, use NMS to get final bounding boxes

References

https://arxiv.org/pdf/1411.4038 - FCN

https://arxiv.org/pdf/1605.06409 - R-FCN

https://arxiv.org/abs/1505.04597 - UNets

https://arxiv.org/abs/1311.2524 - R-CNN

https://arxiv.org/abs/1504.08083 - Fast R-CNN

https://d2l.ai/chapter_computer-vision/fcn.html

https://arxiv.org/abs/1506.01497 - Faster R-CNN

https://www.youtube.com/watch?v=itjQT-gFQBY

https://jonathan-hui.medium.com/understanding-region-based-fully-convolutional-networks-r-fcn-for-object-detection-828316f07c99

https://www.youtube.com/watch?v=nDPWywWRIRo