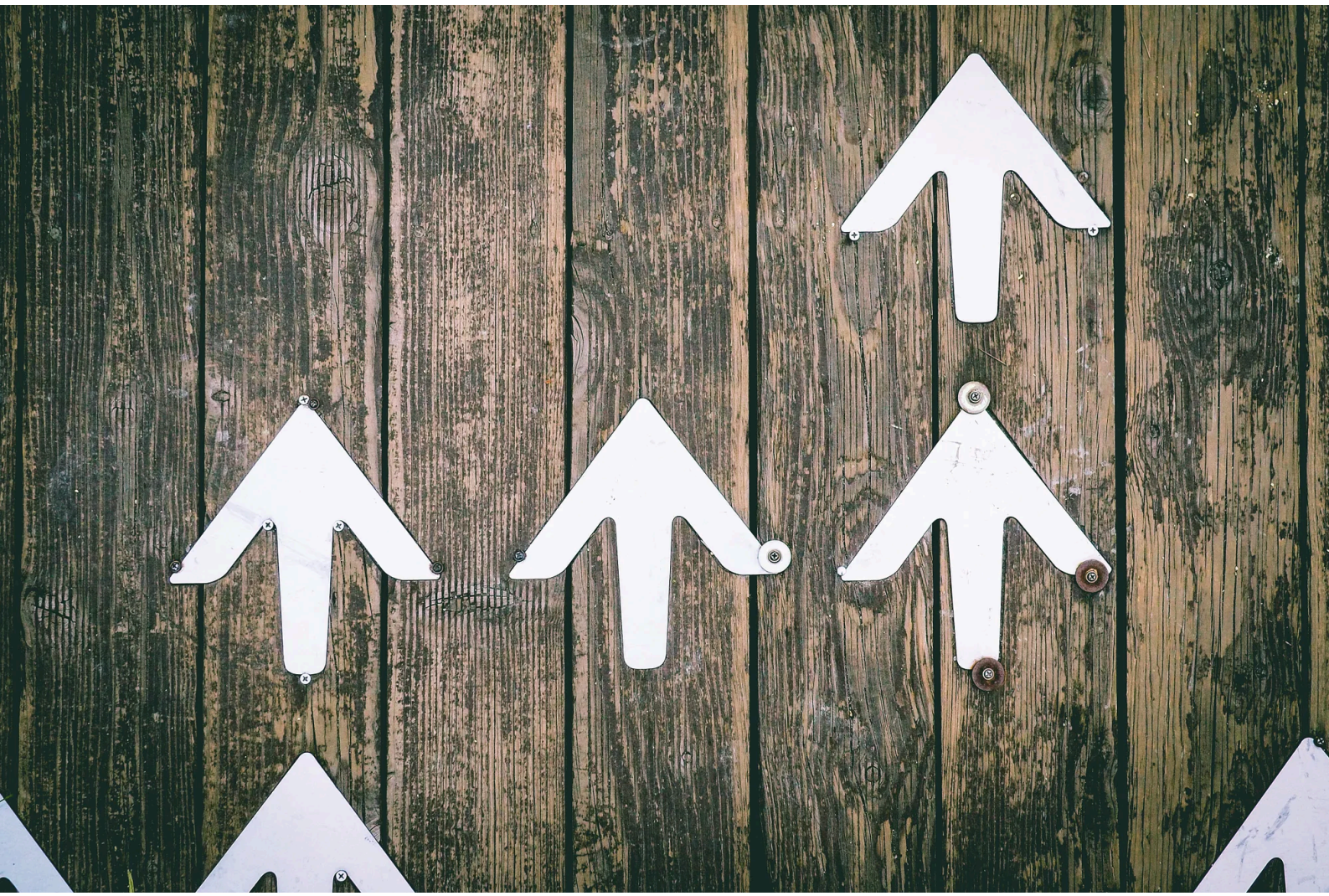


What do we learn from region based object detectors (Faster R-CNN, R-FCN, FPN)?

Jonathan Hui · Follow

11 min read · Mar 28, 2018

In this series, we will take a comprehensive journey on object detection. In Part 1 here, we cover the region based object detectors including Fast R-CNN, Faster R-CNN, R-FCN and FPN. In part 2, we will study the single shoot detectors. In part 3, we cover the performance and some implementation issues. By studying them in one context, we study what is working, what matters and where can be improved. Hopefully, by studying how we get here, it will give us more insights on where we are heading.

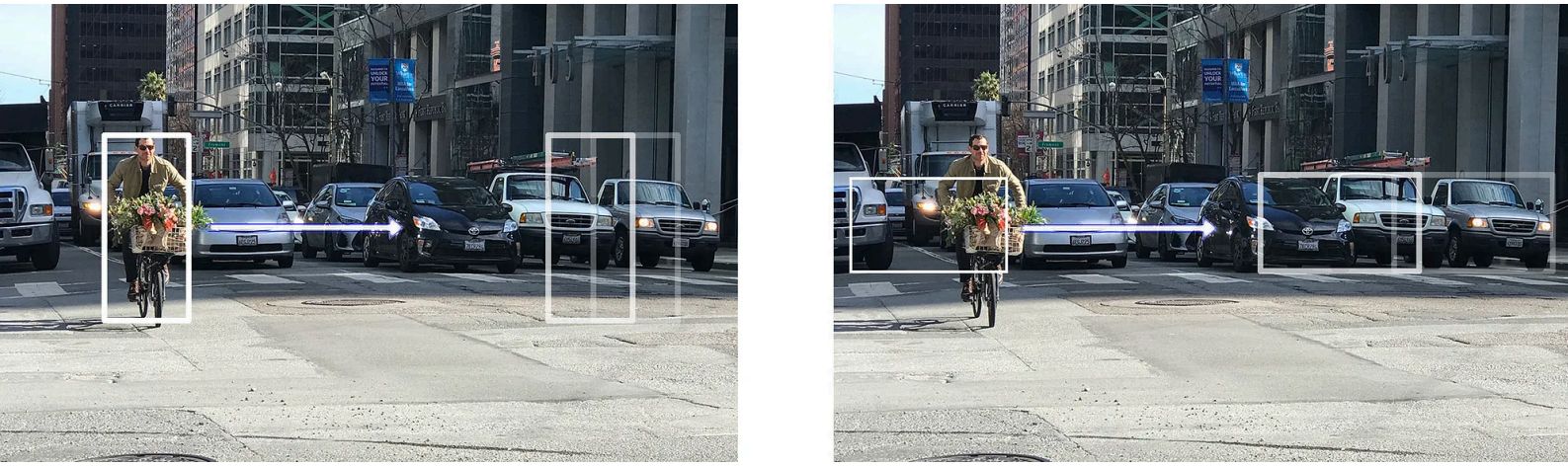
Part 1: What do we learn from region based object detectors (Faster R-CNN, R-FCN, FPN)?

Part 2: What do we learn from single shot object detectors (SSD, YOLO), FPN & Focal loss?

Part 3: Design choices, lessons learned and trends for object detections?

Sliding-window detectors

Since AlexNet won the 2012 ILSVRC challenge, the use of the CNN for classification has dominated the field. One brute force approach for object detection is to slide windows from left and right, and from up to down to identify objects using classification. To detect different object types at various viewing distances, we use windows of varied sizes and aspect ratios.

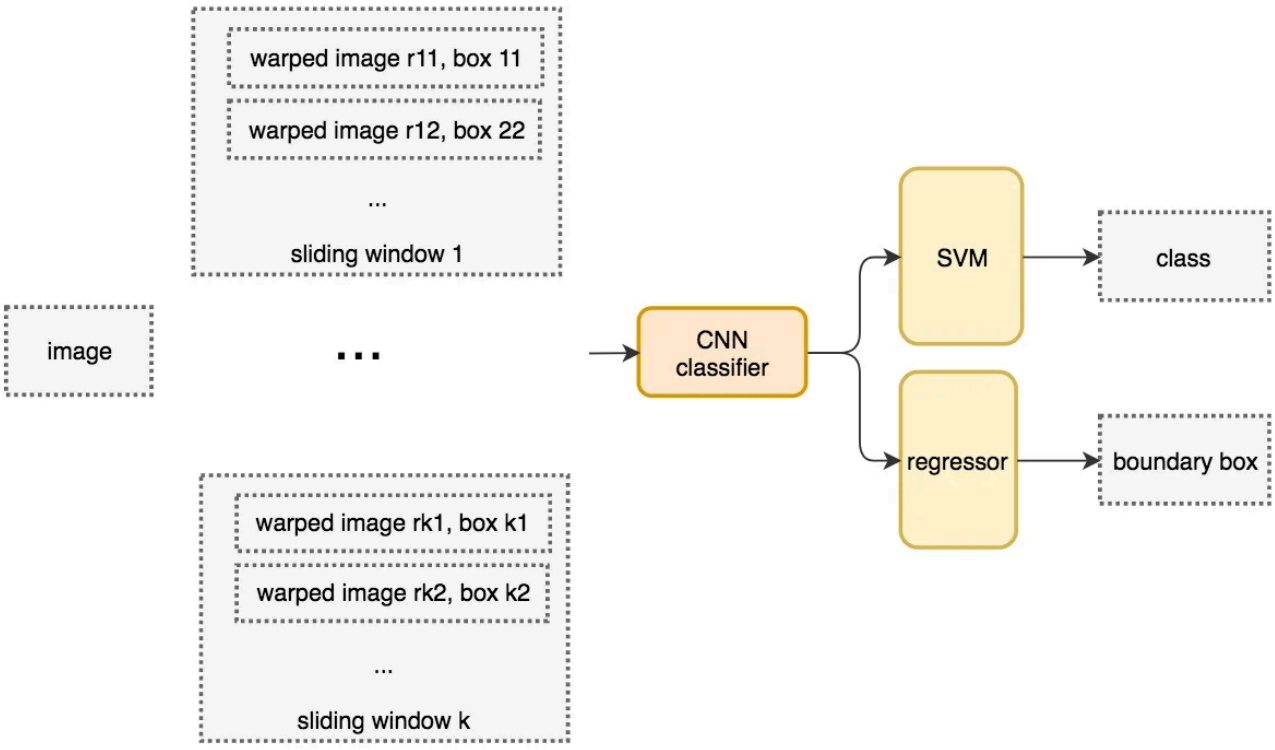


Sliding-windows (From right to left, up and down)

We cut out patches from the picture according to the sliding windows. The patches are warped since many classifiers take fixed size images only. However, this should not impact the classification accuracy since the classifier are trained to handle warped images.

Warp an image to a fixed size image.

The warped image patch is fed into a CNN classifier to extract 4096 features. Then we apply a SVM classifier to identify the class and another linear regressor for the boundary box.



System flow for the sliding-window detector.

Below is the pseudo code. We create many windows to detect different object shapes at different locations. To improve performance, one obvious solution is to reduce the number of *windows*.

```
for window in windows
    patch = get_patch(image, window)
    results = detector(patch)
```

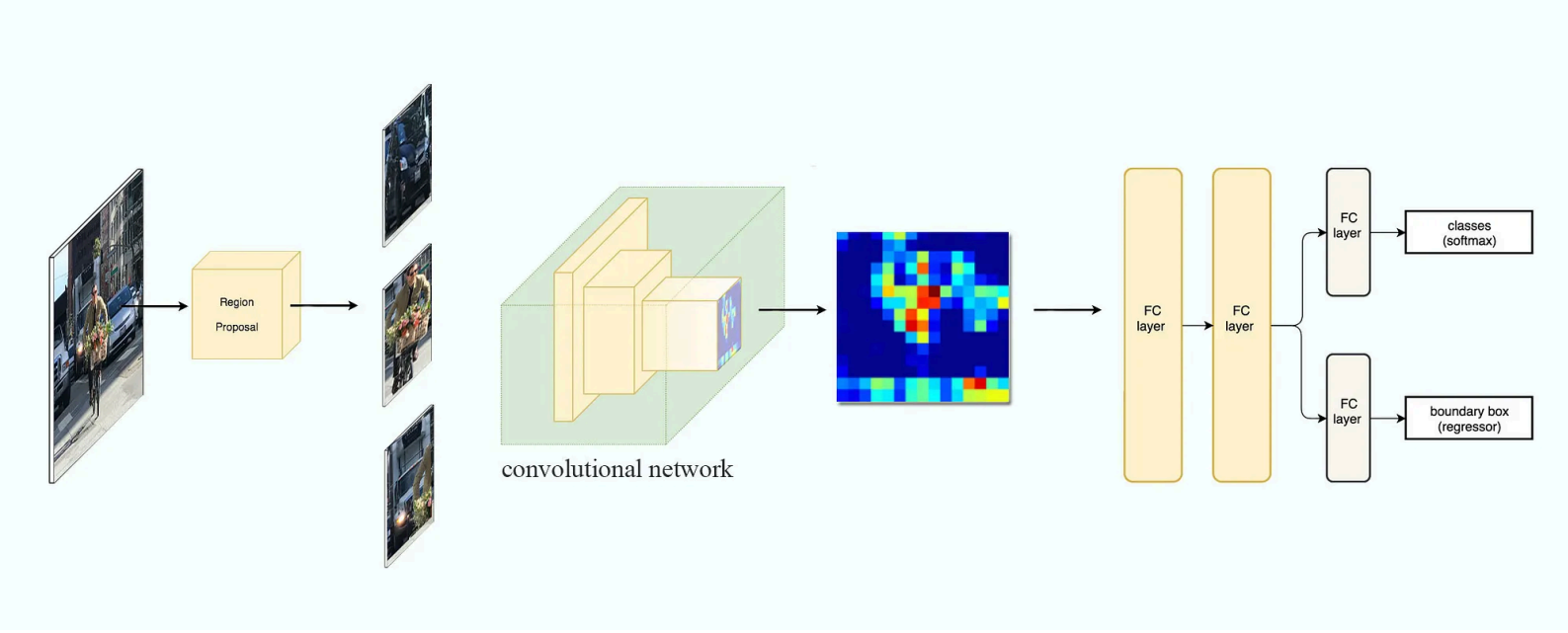
Selective Search

Instead of a brute force approach, we use a region proposal method to create **regions of interest (ROIs)** for object detection. In **selective search** (**SS**), we start with each individual pixel as its own group. Next, we calculate the texture for each group and combine two that are the closest. But to avoid a single region in gobbling others, we prefer grouping smaller group first. We continue merging regions until everything is combined together. In the first row below, we show how we grow the regions, and the blue rectangles in the second rows show all possible ROIs we made during the merging.

(Image source: van de Sande et al. IC

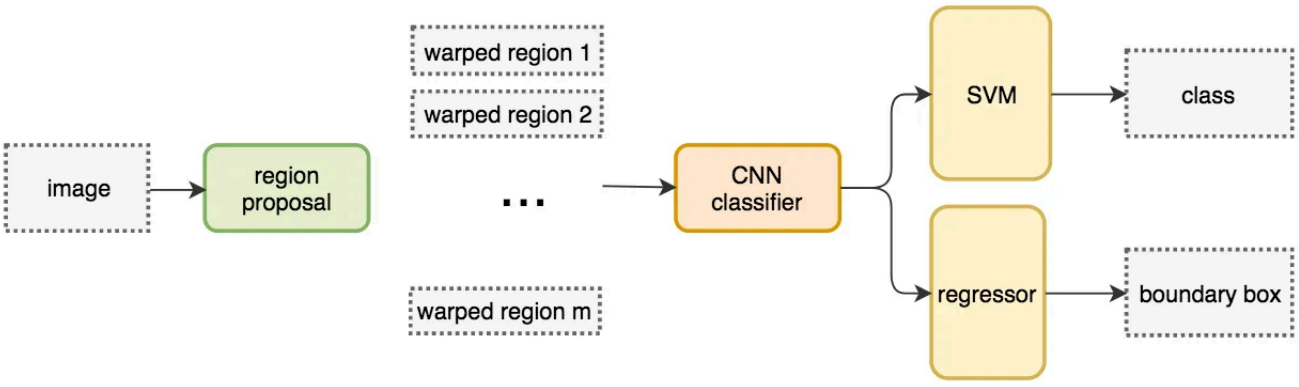
R-CNN

R-CNN makes use of a region proposal method to create about 2000 **ROI**s (regions of interest). The regions are warped into fixed size images and feed into a CNN network individually. It is then followed by fully connected layers to classify the object and to refine the boundary box.



Use region proposals, CNN, affine layers to locate objects.

Here is the system flow.



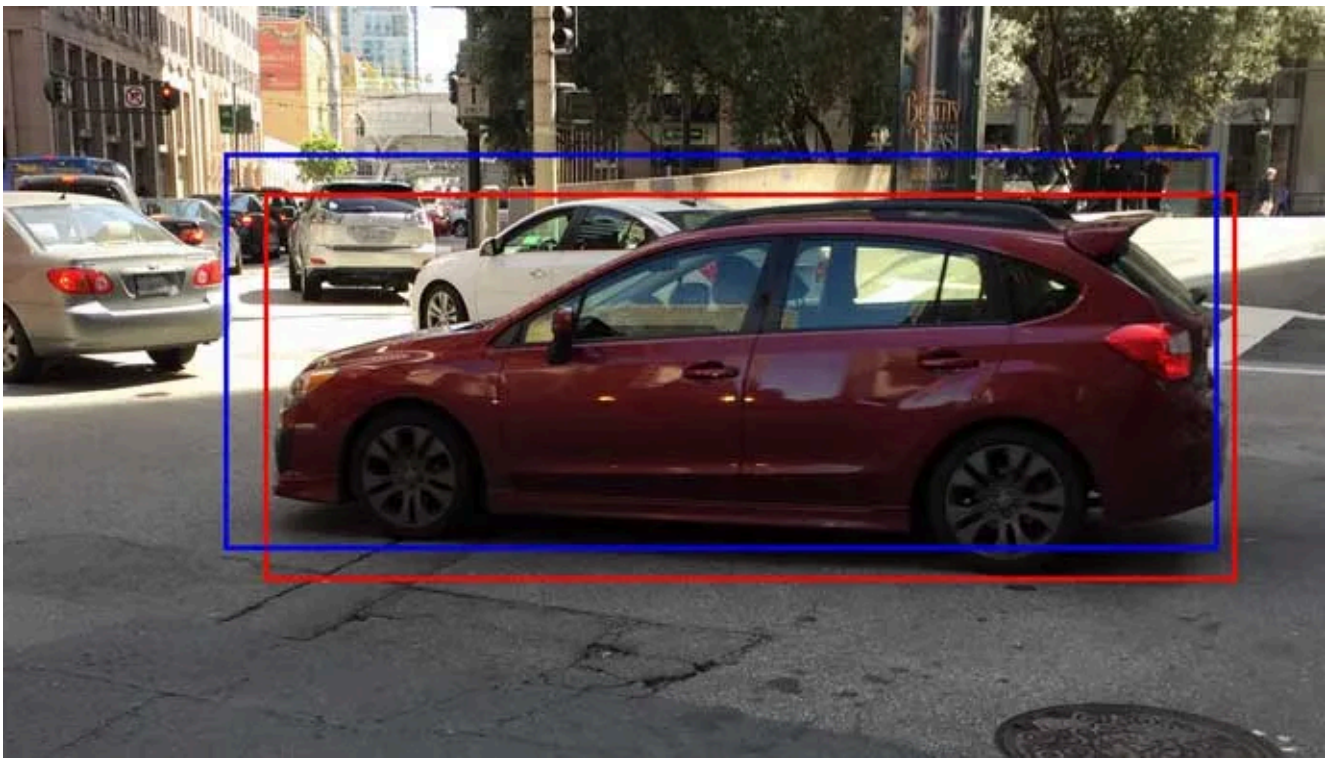
System flow for R-CNN

With far fewer but higher quality ROIs, R-CNN run faster and more accurate than the sliding windows.

```
ROIs = region_proposal(image)
for ROI in ROIs
    patch = get_patch(image, ROI)
    results = detector(patch)
```

Boundary box regressor

Region proposal methods are computation intense. To speed up the process, we often pick a less expensive region proposal method to create ROIs followed by a linear regressor (using fully connected layers) to refine the boundary box further.



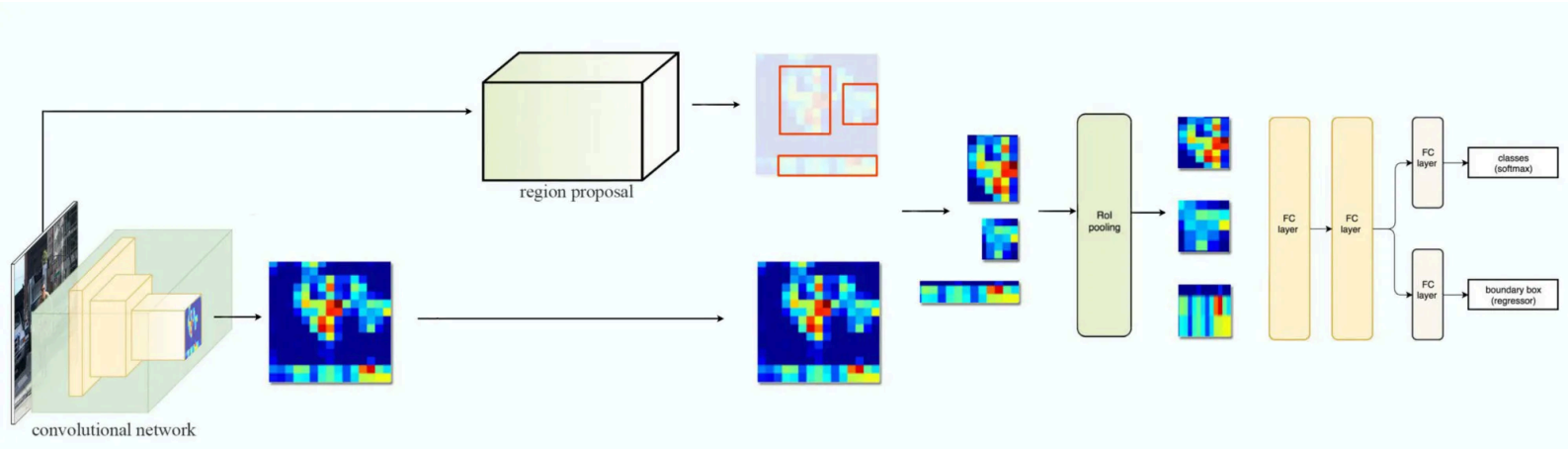
Use regression to refine the original ROI in blue to the red one.

Fast R-CNN

R-CNN needs many proposals to be accurate and many regions overlap with each other. **R-CNN is slow in training & inference.** If we have 2,000 proposals, each of them is processed by a CNN separately, i.e. we repeat feature extractions 2,000 times for different ROIs.

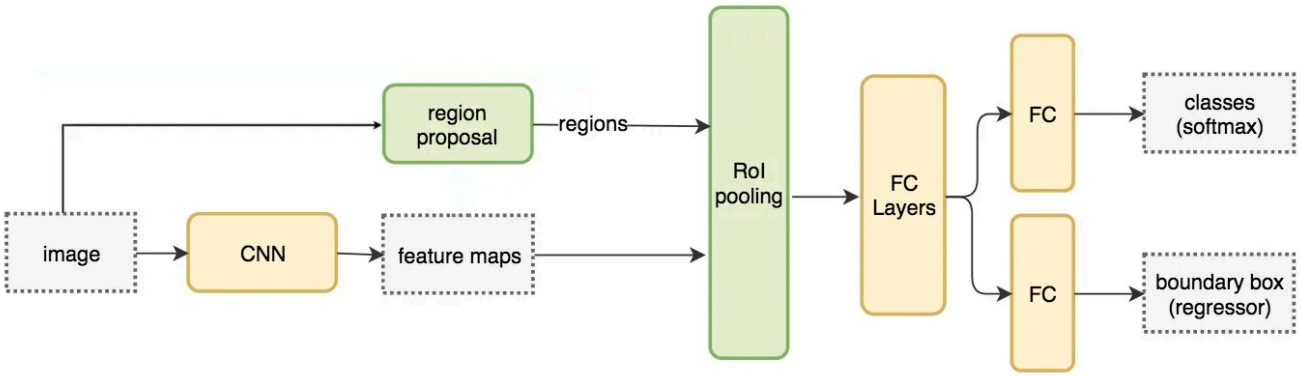
Instead of extracting features for each image patch from scratch, we use a **feature extractor** (a CNN) to extract features for th

also use an external region proposal method, like the selective search, to create ROIs which later combine with the corresponding feature maps to form patches for object detection. We warp the patches to a fixed size using **ROI pooling** and feed them to fully connected layers for classification and **localization** (detecting the location of the object). By not repeating the feature extractions, Fast R-CNN cuts down the process time significantly.



Apply region proposal on feature maps and form fixed size patches using ROI pooling.

Here is the network flow:



In the pseudo-code below, the expensive feature extraction is moving out of the for-loop, a significant speed improvement since it was executed for all 2000 ROIs. Fast R-CNN is 10x faster than R-CNN in training and 150x faster in inferencing.

```
feature_maps = process(image)
ROIs = region_proposal(image)
```

```
for ROI in ROIs
    patch = roi_pooling(feature_maps, ROI)
    results = detector2(patch)
```

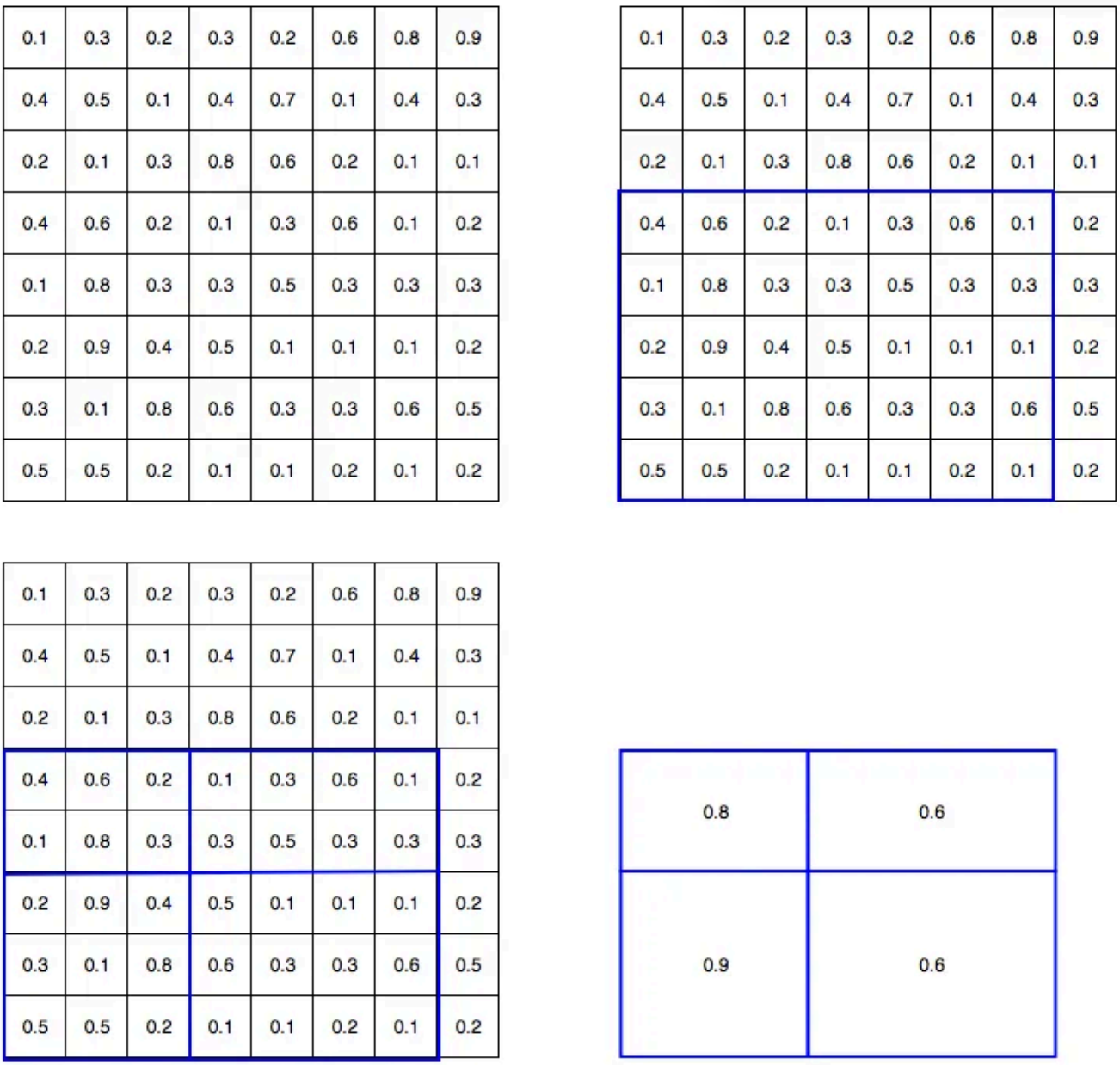
One major takeaway for Fast R-CNN is that the whole network (the feature extractor, the classifier, and the boundary box regressor) are trained end-to-end with **multi-task losses** (classification loss and localization loss). This improves accuracy.

ROI Pooling

Because Fast R-CNN uses fully connected layers, we apply **ROI pooling** to warp the variable size ROIs into in a predefined size shape.

Let's simplify the discussion by transforming 8 × 8 feature maps into a predefined 2 × 2 shape.

- Top left below: our feature maps.
- Top right: we overlap the ROI (blue) with the feature maps.
- Bottom left: we split ROIs into the target dimension. For example, with our 2×2 target, we split the ROIs into 4 sections with similar or equal sizes.
- Bottom right: find the maximum for each section and the result is our warped feature maps.



Input feature map (top left), output feature map (bottom right), blue box is the ROI (top right).

So we get a 2 × 2 feature patch that we can feed into the classifier and box regressor.

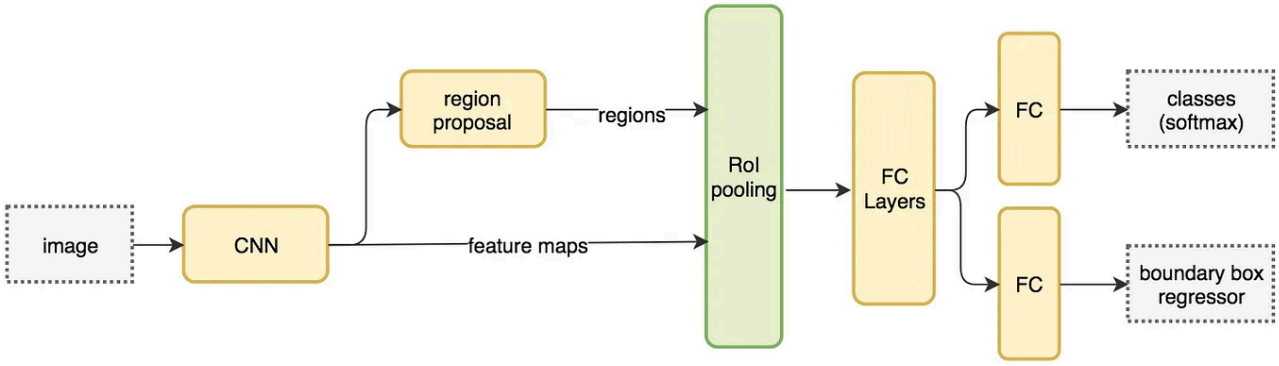
Faster R-CNN

Fast R-CNN depends on an external region proposal method like selective search. However, those algorithms run on CPU and they are slow. In testing, Fast R-CNN takes 2.3 seconds to make a prediction in which 2 seconds are for generating 2000 ROIs.

```
feature_maps = process(image)
ROIs = region_proposal(image)          # Expen
for ROI in ROIs
```

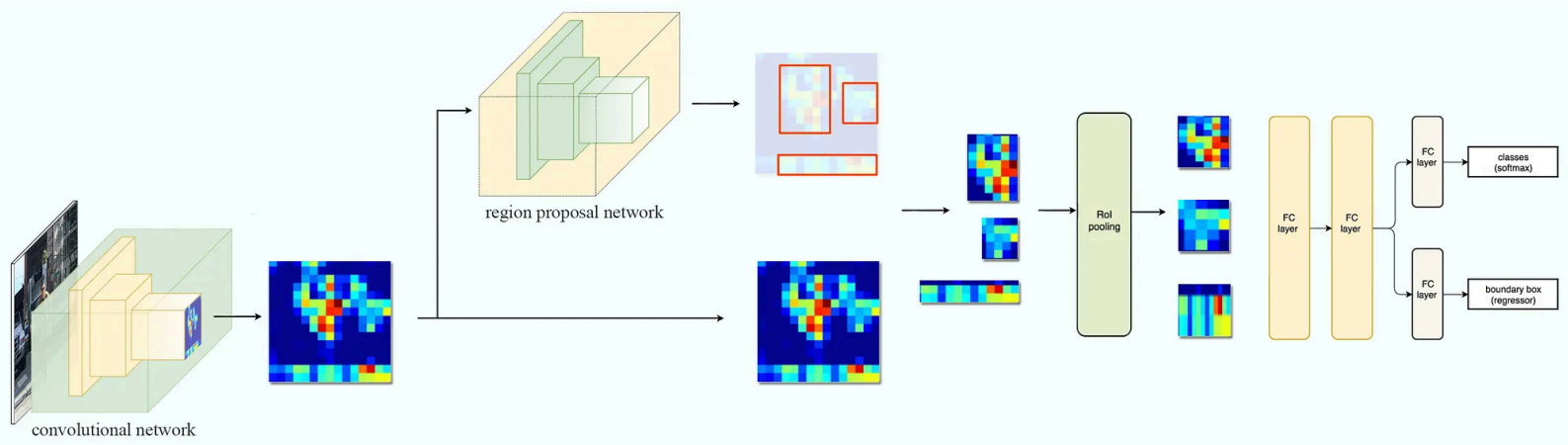
```
    patch = roi_pooling(feature_maps, ROI)
    results = detector2(patch)
```

Faster R-CNN adopts similar design as the Fast R-CNN except it replaces the region proposal method by an internal deep network and the ROIs are derived from the feature maps instead. The new region proposal network (**RPN**) is more efficient and run at 10 ms per image in generating ROIs.



Network flow is the same as the Fast R-CNN.

The network flow is similar but the region proposal is now replaced by a convolutional network (RPN).

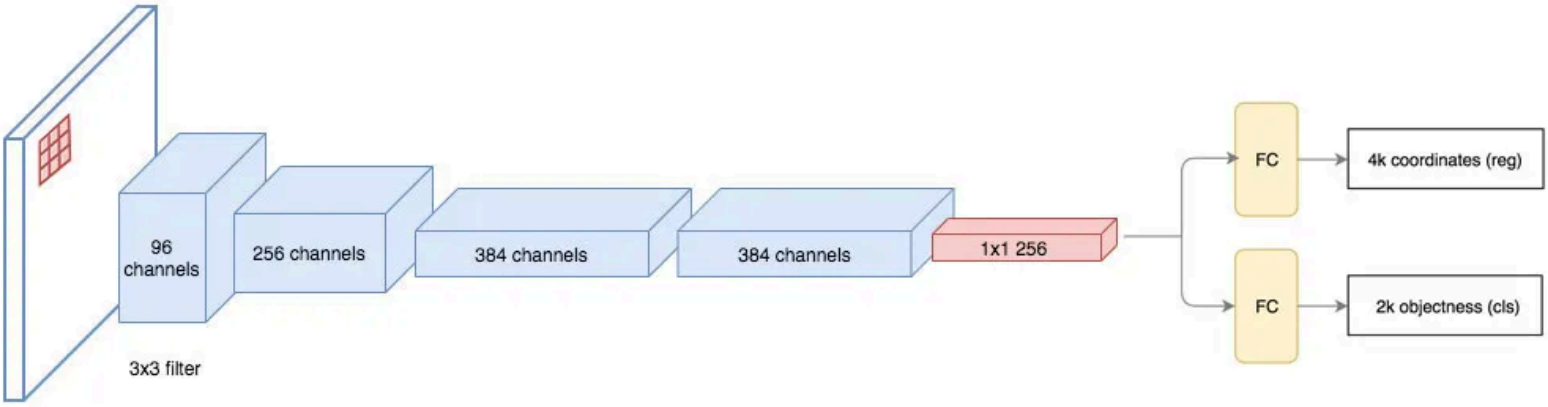


The external region proposal is replaced by an internal deep network.

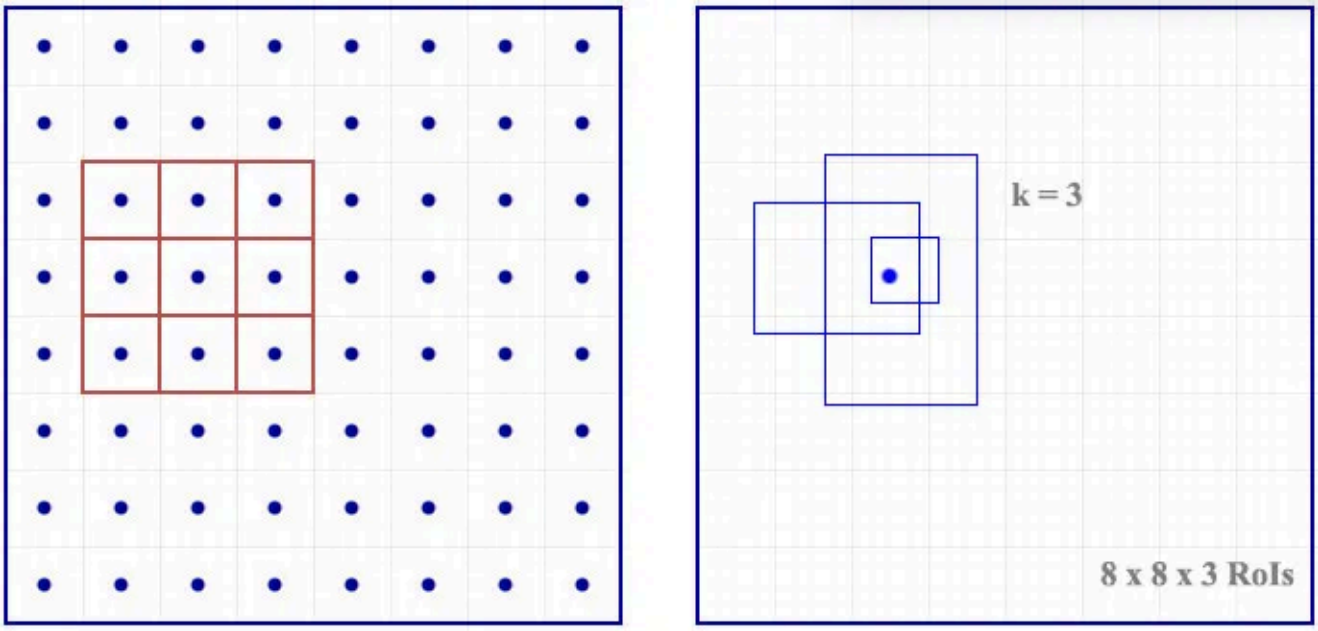
Region proposal network

The region proposal network (**RPN**) takes the output feature maps from the first convolutional network as input. It slides 3 × 3

maps to make class-agnostic region proposals using a convolutional network like ZF network (below). Other deep network likes VGG or ResNet can be used for more comprehensive feature extraction at the cost of speed. The ZF network outputs 256 values, which is feed into 2 separate fully connected layers to predict a boundary box and 2 objectness scores. The **objectness** measures whether the box contains an object. We can use a regressor to compute a single objectness score but for simplicity, Faster R-CNN uses a classifier with 2 possible classes: one for the "have an object" category and one without (i.e. the background class).



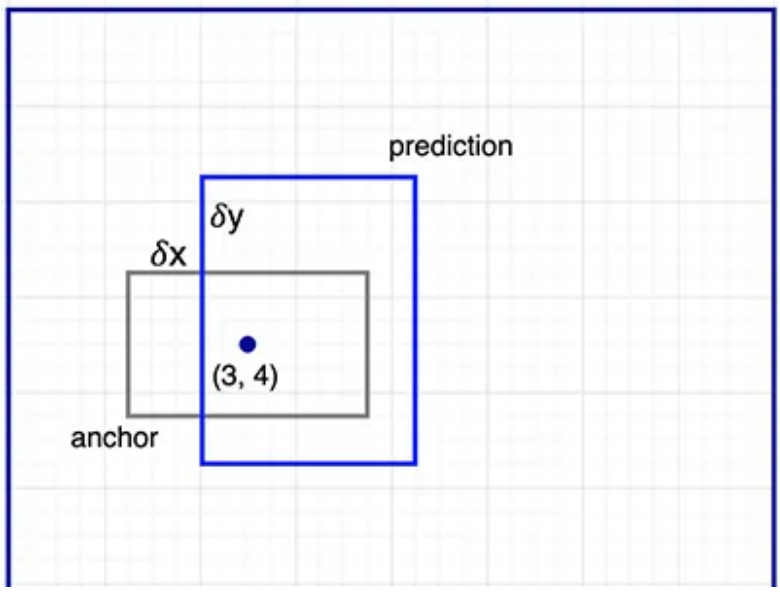
For each location in the feature maps, RPN makes **k** guesses. Therefore RPN outputs 4×k coordinates and 2×k scores per location. The diagram below shows the 8 × 8 feature maps with a 3× 3 filter, and it outputs a total of 8 × 8 × 3 ROIs (for k = 3). The right side diagram demonstrates the 3 proposals made by a single location.



8 x 8 feature maps

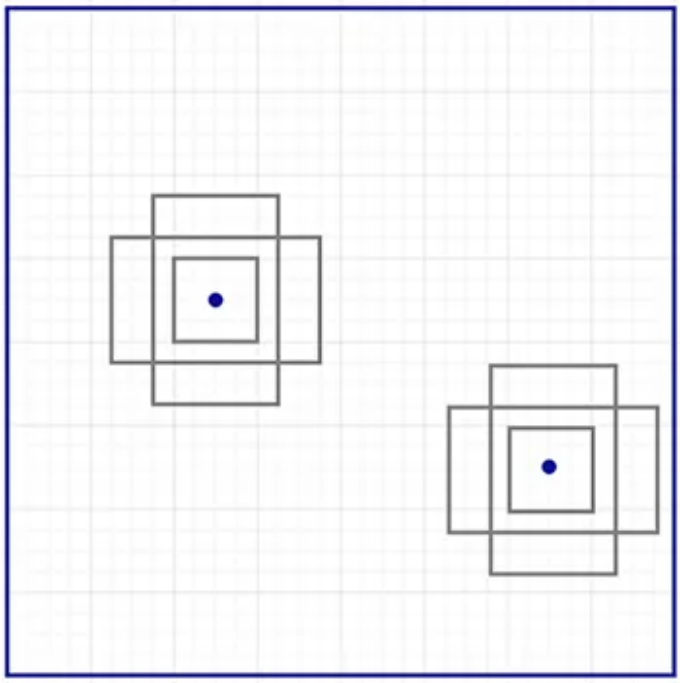
3 proposals for each location

Here, we get 3 guesses and we will refine our guesses later. Since we just need one to be correct, we will be better off if our initial guesses have different shapes and size. Therefore, Faster R-CNN does not make random boundary box proposals. Instead, it predicts offsets like δx, δy that are relative to the top left corner of some reference boxes called **anchors**. We constraints the value of those offsets so our guesses still resemble the anchors.



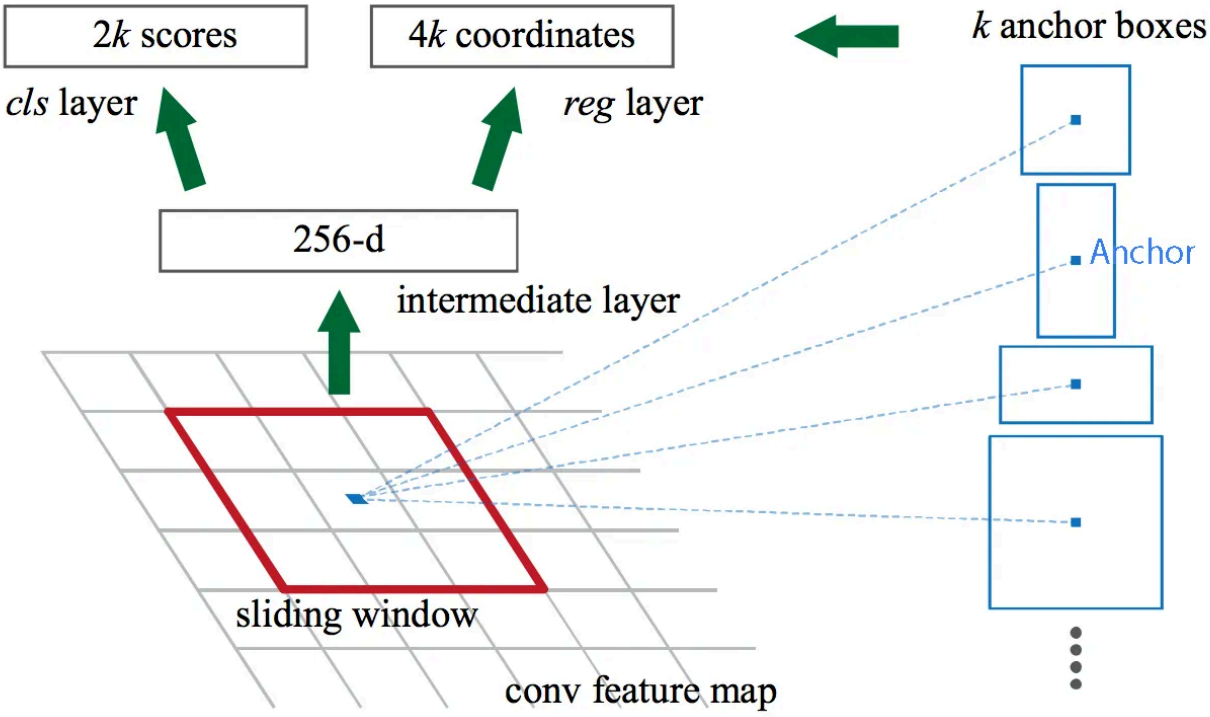
To make k predictions per location, we need k anchors centered at each location. Each prediction is associated with a spe

locations share the same anchor shapes.



Those anchors are carefully pre-selected so they are diverse and cover real-life objects at different scales and aspect ratios reasonable well. This guides the initial training with better guesses and allows each prediction to specialize in a certain shape. This strategy makes early training more stable and easier.

Faster R-CNN uses far more anchors. It deploys 9 anchor boxes: 3 different scales at 3 different aspect ratio. Using 9 anchors per location, it generates 2×9 objectness scores and 4×9 coordinates per location.

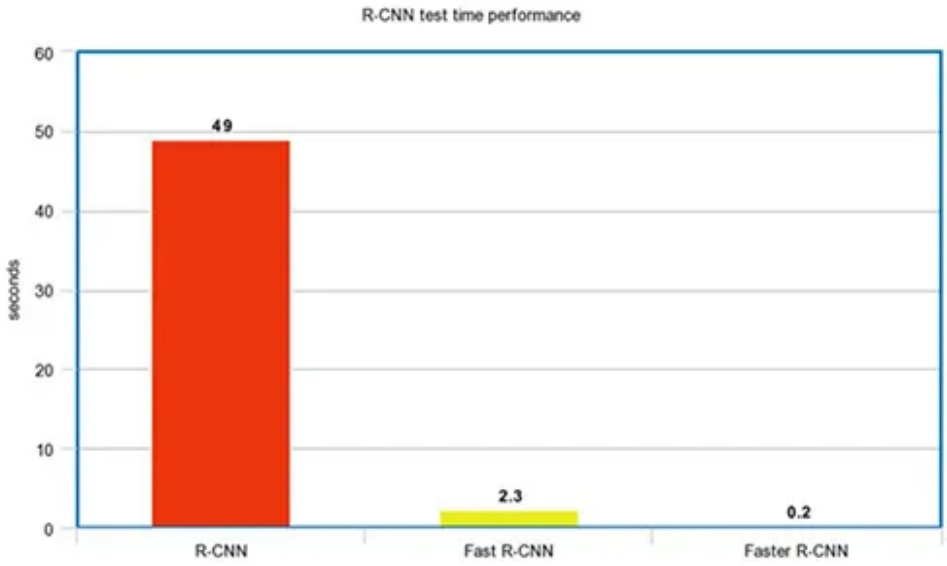


Source

> *Anchors are also called **priors** or **default boundary boxes** in different papers.*

Performance for R-CNN methods

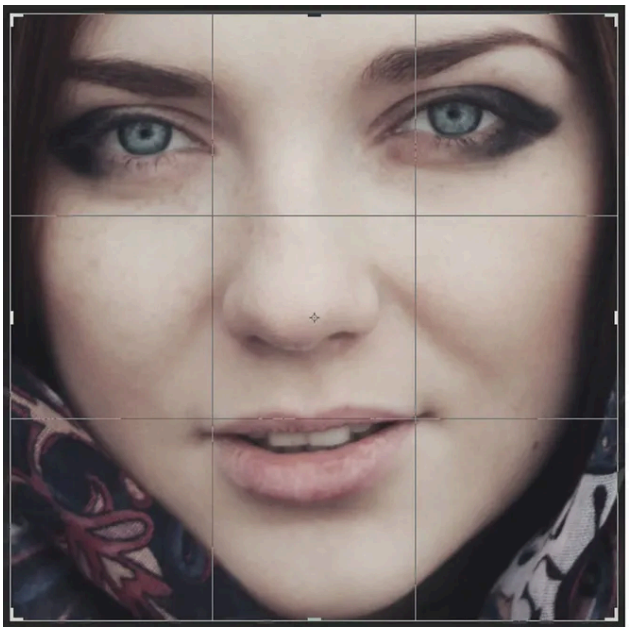
As shown below, Faster R-CNN is even much faster.

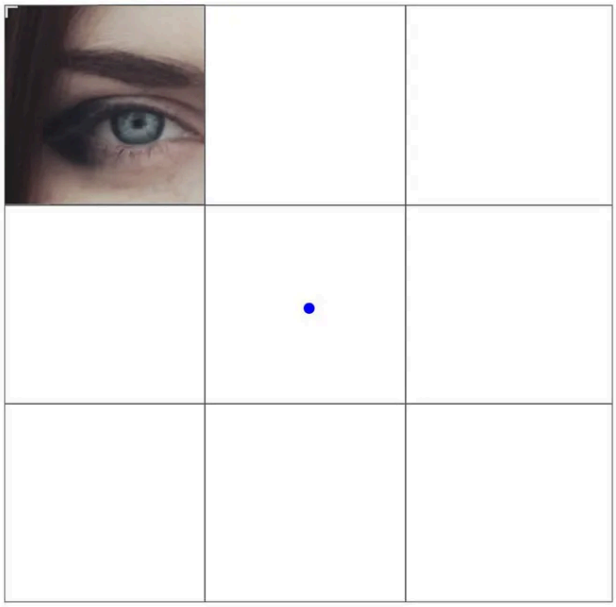


Region-based Fully Convolutional Networks (R-FCN)

Let's assume we only have a feature map detecting the right eye of a face. Can we use it to locate a face? It should. Since the

top-left corner of a facial picture, we can use that to locate the face.





If we have other feature maps specialized in detecting the left eye, the nose or the mouth, we can combine the results together to locate the face better.

So why we go through all the trouble. In Faster R-CNN, the *detector* applies multiple fully connected layers to make predictions. With 2,000 ROIs, it can be expensive.

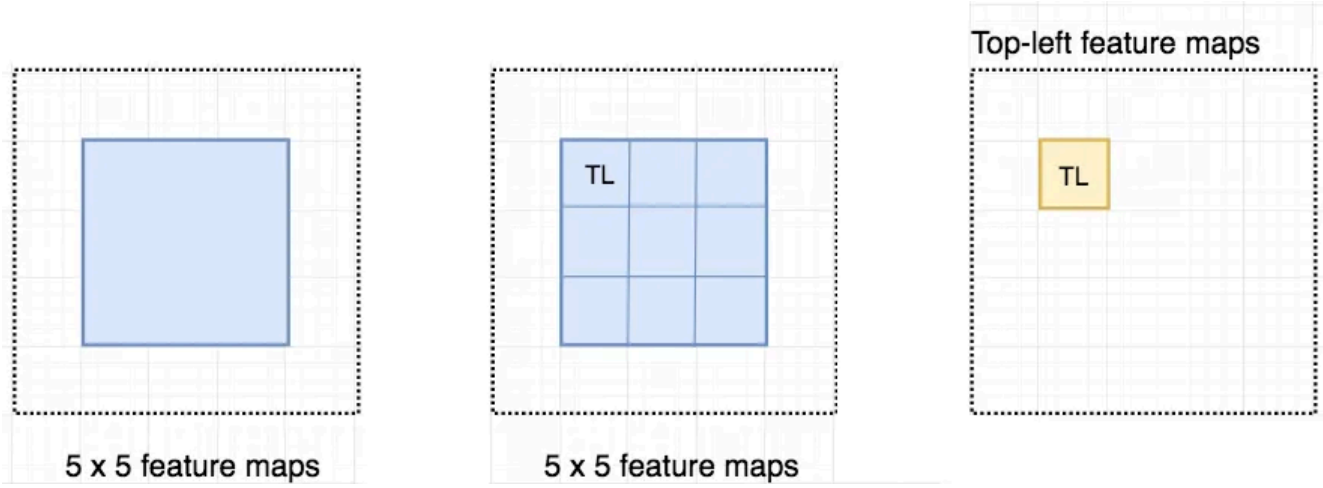
```
feature_maps = process(image)
ROIs = region_proposal(feature_maps)
for ROI in ROIs
    patch = roi_pooling(feature_maps, ROI)
    class_scores, box = detector(patch)         # Expensive!
    class_probabilities = softmax(class_scores)
```

R-FCN improves speed by reducing the amount of work needed for each ROI. The region-based feature maps above are independent of ROIs and can be computed outside each ROI. The remaining work is much simpler and therefore R-FCN is faster than Faster R-CNN.

```
feature_maps = process(image)
ROIs = region_proposal(feature_maps)
```

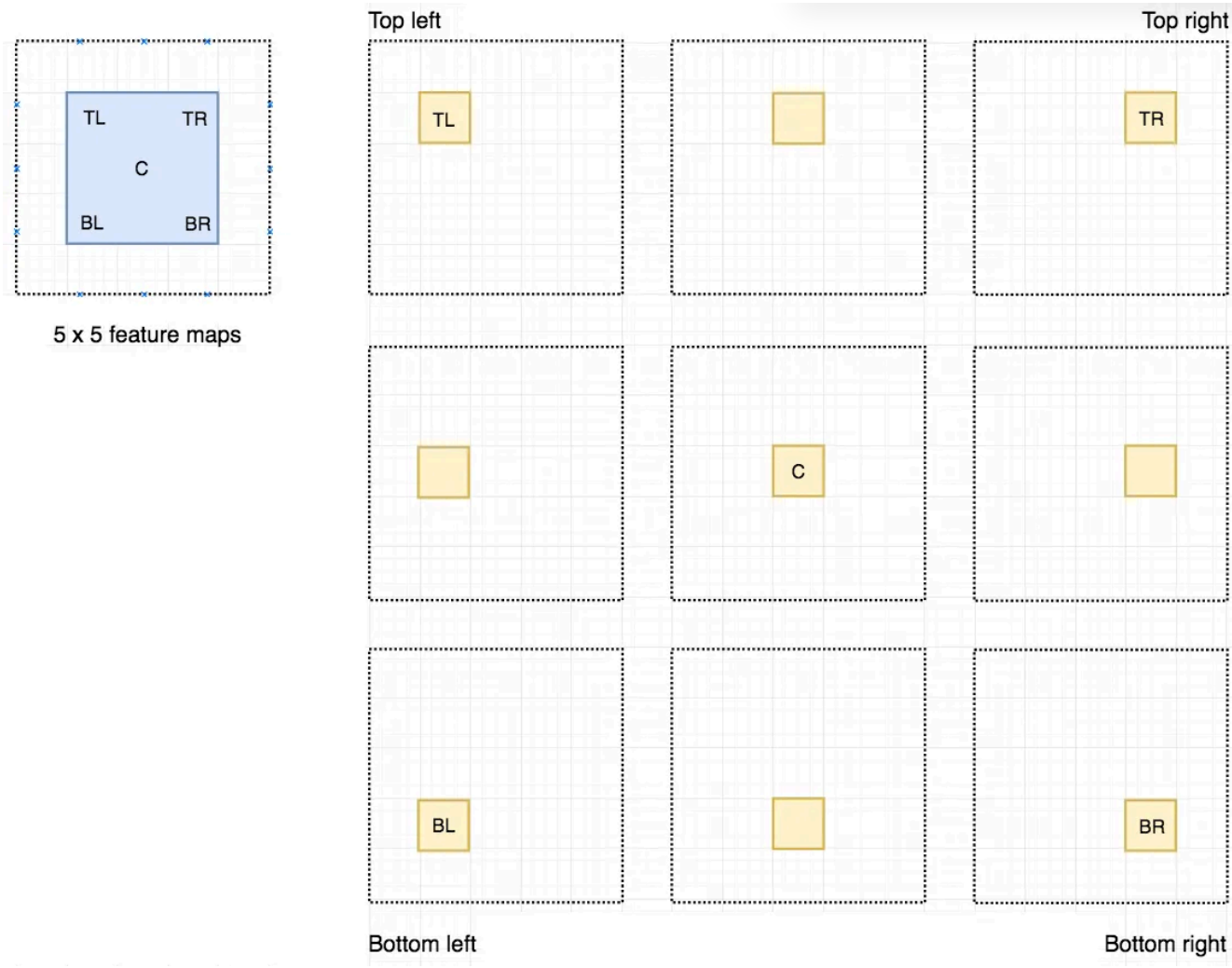
```
score_maps = compute_score_map(feature_maps)
for ROI in ROIs
    V = region_roi_pool(score_maps, ROI)
    class_scores, box = average(V)                    # Much simpler!
    class_probabilities = softmax(class_scores)
```

Let's consider a 5 × 5 feature map **M** with a blue square object inside. We divide the square object equally into 3 × 3 regions. Now, we create a new feature map from M to detect the top left (TL) corner of the square only. The new feature map looks like the one on the right below. Only the yellow **grid cell** [2, 2] is activated.



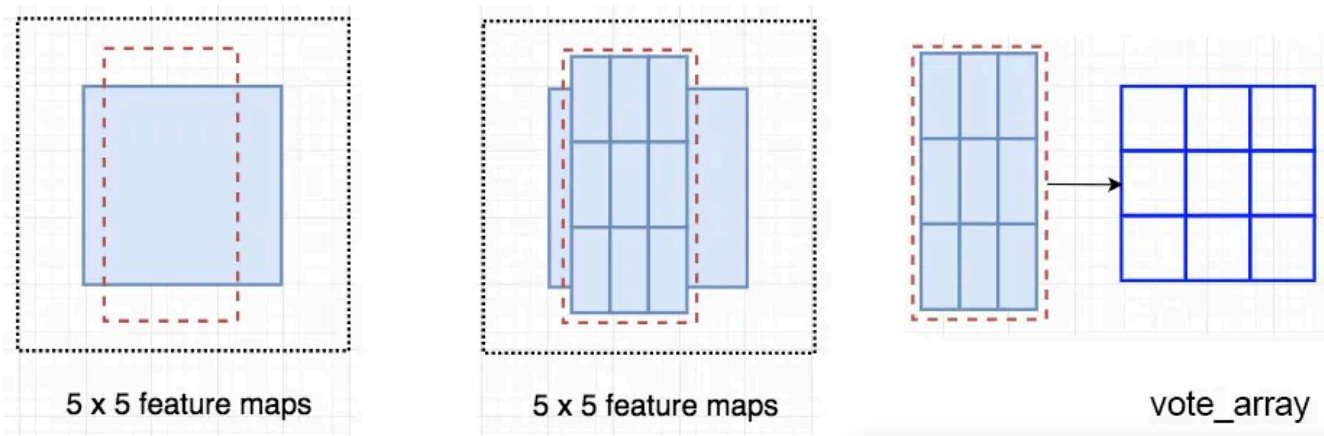
Create a new feature map from the left to detect the top left corner of an object.

Since we divide the square into 9 parts, we can create 9 feature maps each detecting the corresponding region of the object. These feature maps are called **position-sensitive score maps** because each map detects (scores) a sub-region of the object.



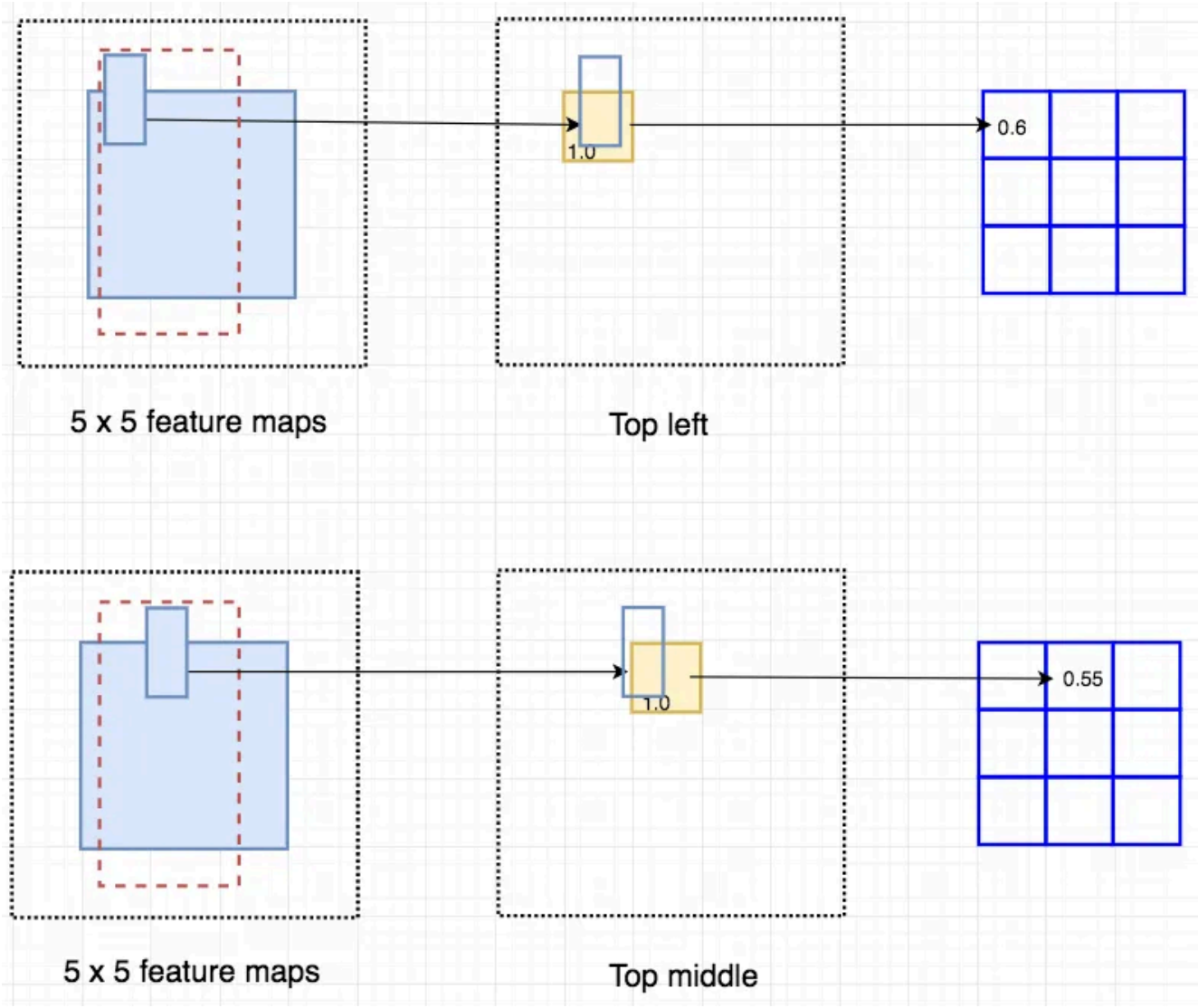
Generate 9 score maps

Let's say the dotted red rectangle below is the ROI proposed. We divide it into 3 × 3 regions and ask how likely each region contains the corresponding part of the object. For example, how likely the top-left ROI region contains the left eye. We store the results into a 3 × 3 vote array in the right diagram below. For example, vote_array[0][0] contains the score on whether we find the top-left region of the square object.



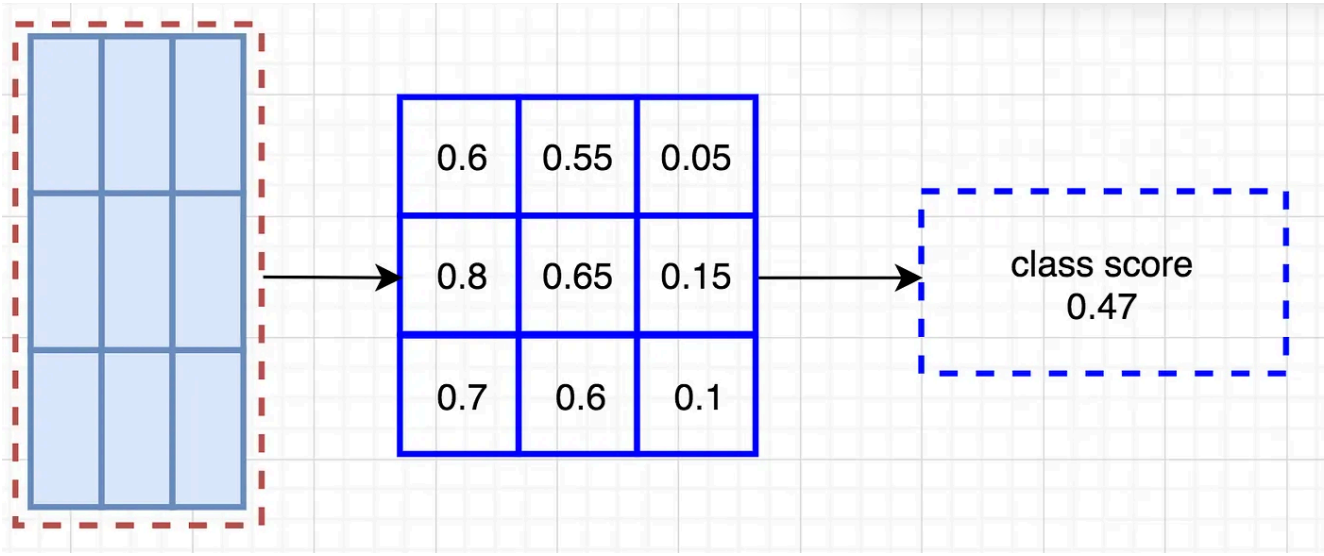
Apply ROI onto the feature maps to output a

This process to map score maps and ROIs to the vote array is called **position-sensitive ROI-pool**. The process is extremely close to the ROI pool we discussed before. We will not cover it further but you can refer to the future reading section for more information.



Overlay a portion of the ROI onto the corresponding score map to calculate V[i][j]

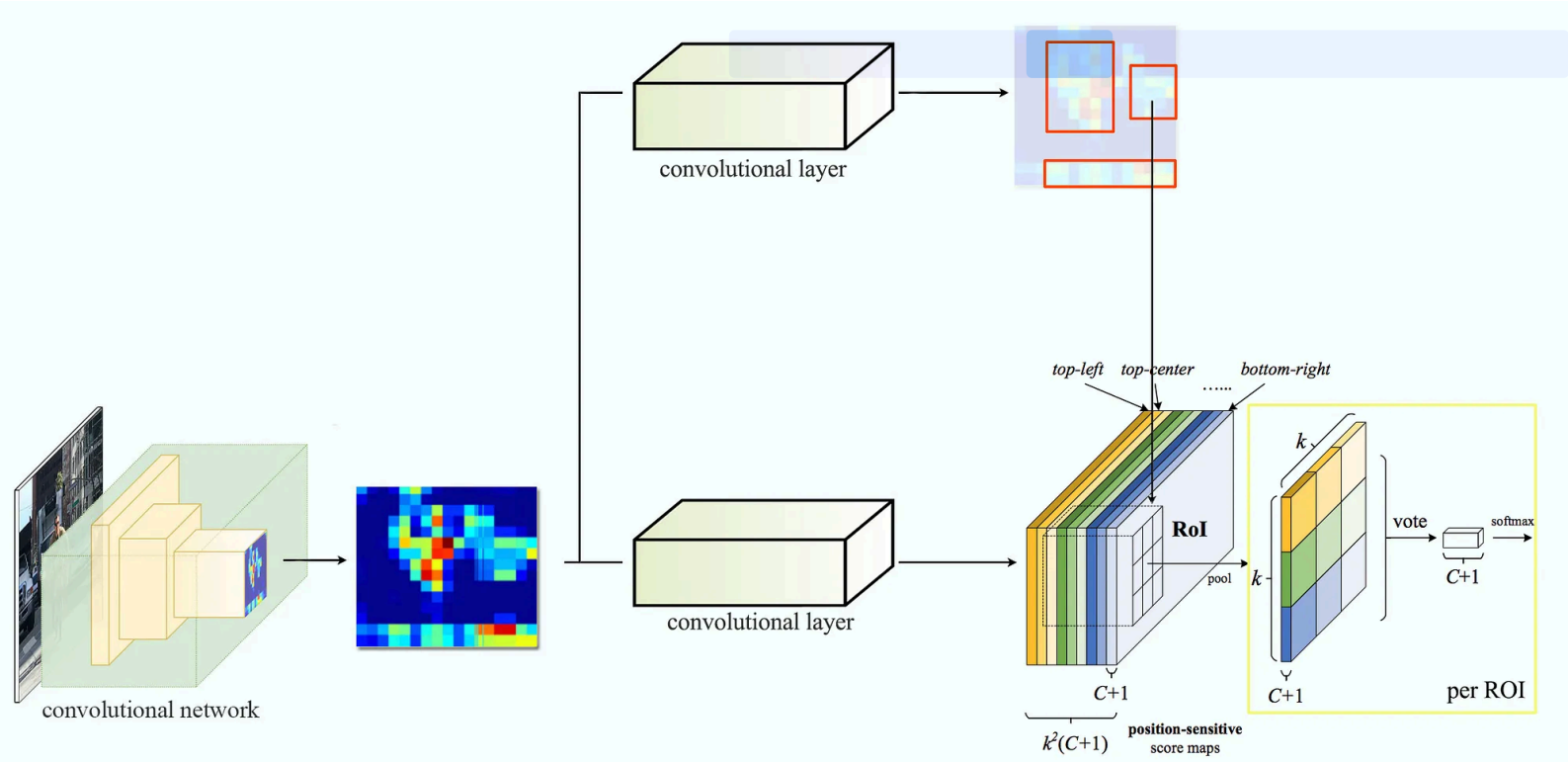
After calculating all the values for the position-sensitive ROI pool, the class score is the average of all its elements.



ROI pool

Let's say we have **C** classes to detect. We expand it to C + 1 classes so we include a new class for the background (non-object). Each class will have its own 3×3 score maps and therefore a total of $(C+1) \times 3 \times 3$ score maps. Using its own set of score maps, we predict a class score for each class. Then we apply a softmax on those scores to compute the probability for each class.

The following is the data flow. For our example, we have k=3 below.



Our journey so far

We start with the basic sliding window algorithm.

```
for window in windows
    patch = get_patch(image, window)
    results = detector(patch)
```

Then we try to reduce the number of windows and move as much work as possible outside the for-loop.

```
ROIs = region_proposal(image)
for ROI in ROIs
    patch = get_patch(image, ROI)
    results = detector(patch)
```

In Part 2, we go even further to completely remove the for-loop. Single shot detectors make object detections in single shot without a separate step of region proposal.

Further reading on FPN, R-FCN and Mask R-CNN

Both FPN and R-FCN are more complex than we described here. For further study, please refer to:

- Feature Pyramid Networks (FPN) for object detection.
- Region-based Fully Convolutional Networks (R-FCN).

Resources

Detectron: Facebook Research's implementation of the Faster R-CNN and Mask R-CNN using Caffe2.

The official implementation for the Faster R-CNN in MATLAB.

Faster R-CNN implementation in TensorFlow.

R-FCN implementation in MXNet.

R-FCN implementation in Caffe and MATLAB.

R-FCN implementation in TensorFlow.

Machine Learning | Deep Learning | Computer Vision | Object Detection | Artificial Intelligence

Written by Jonathan Hui

40K Followers

Deep Learning

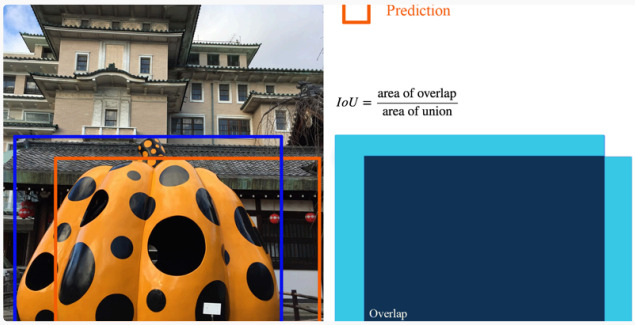
More from Jonathan Hui

Jonathan Hui

The Study of Mathematical Spaces (Machine Learning)

The terminology of mathematical spaces in AI research papers can be intimidating....

May 15 606 4

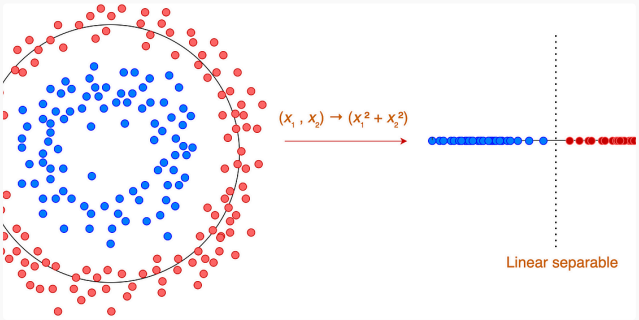


Jonathan Hui

mAP (mean Average Precision) for Object Detection

AP (Average precision) is a popular metric in measuring the accuracy of object detectors...

Mar 6, 2018 7.4K 54

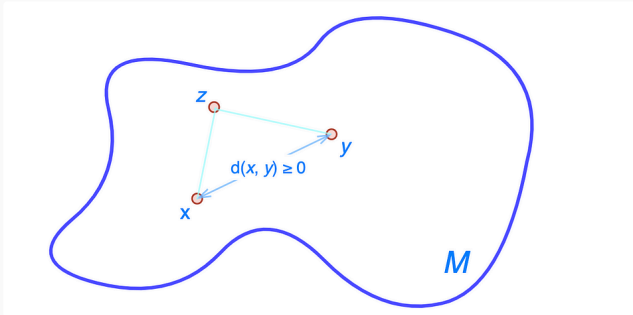


Jonathan Hui

Kernels for Machine Learning

In many machine learning problems, input data is transformed into a higher-dimension...

May 27 218 3



Jonathan Hui

Vector Space, Normed Space & Hilbert Space (Machine Learning)

Euclidean space, the familiar geometry of our everyday world, provides a useful framework...

May 20 376 4

See all from Jonathan Hui

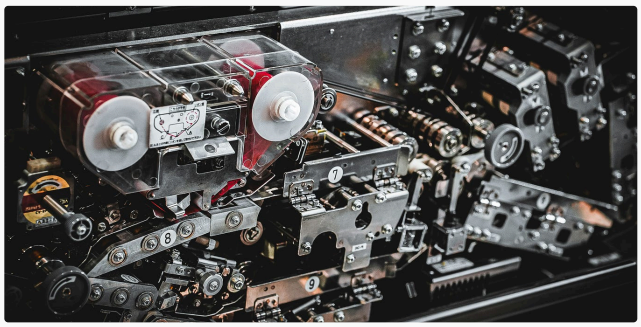
Recommended from Medium

The Tenyks Blogger

CVPR 2024: Top Highlights You Must Know— Embodied AI, GenAI...

We bring you the highlights and the key takeaways directly from CVPR 2024!

6d ago 15



Raghav Bali in Towards Data Science

Exploring Object Detection with R-CNN Models—A Comprehensive...

Object Detection Models

Feb 16 151

Lists

Predictive Modeling w/ Python
20 stories · 1337 saves

AI Regulation
6 stories · 498 saves

Natural Language Processing
1556 stories · 1093 saves

Practical Guides to Machine Learning
10 stories · 1618 saves

55 s remaining

Rajesh Katta

Object Detection part-2: Two Stage Detectors: R-CNN,Fast R-...

Object detection, the task of precisely locating and classifying objects within an...

Feb 29 111

Kavishka Abeywardana

SimCLR: A Simple Framework for Contrastive Learning of Visual...

Unsupervised contrastive learning

May 21 4

Hugman Sangkeun Jung

Intuitive Understanding of Autoencoders and Variational...

Learn about Autoencoders and Variational Autoencoders, their structures, latent space...

May 16 53

Sik-Ho Tsang

Review — YOLOv10: Real-Time End-to-End Object Detection

NMS-Free Training, and Holistic Efficiency-Accuracy Driven Model Design are Proposed

Jun 4 16

See more recommendations