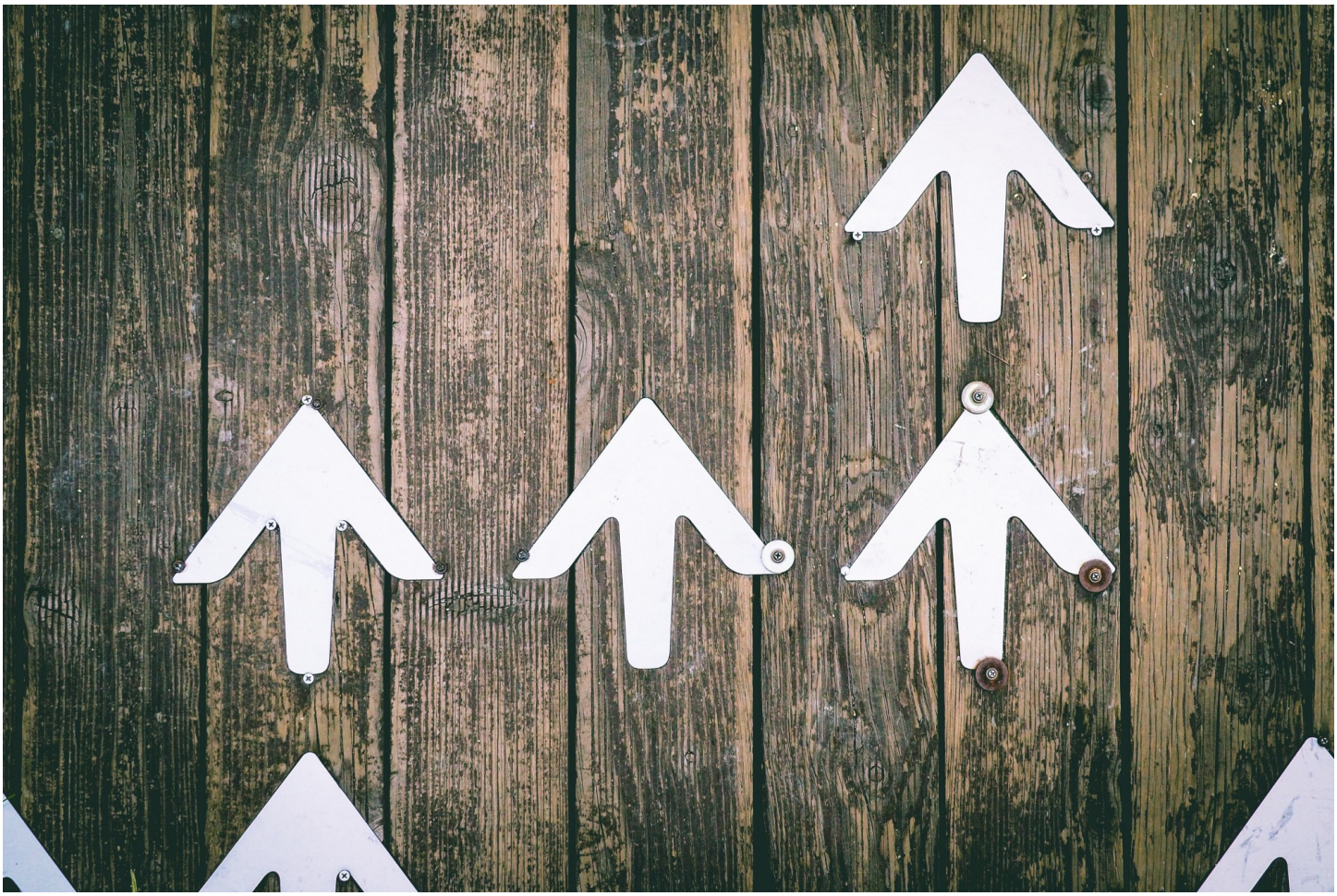
Jonathan Hui Follow
Deep Learning
Mar 28 · 13 min read



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

In this article, we will take a comprehensive journey on object detection. In Part 1 here, we will cover the region based object detectors including Fast R-CNN, Faster R-CNN, Mask R-CNN, R-FCN and FPN. In part 2, we study the single shoot detectors. In part 3, we covers 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.

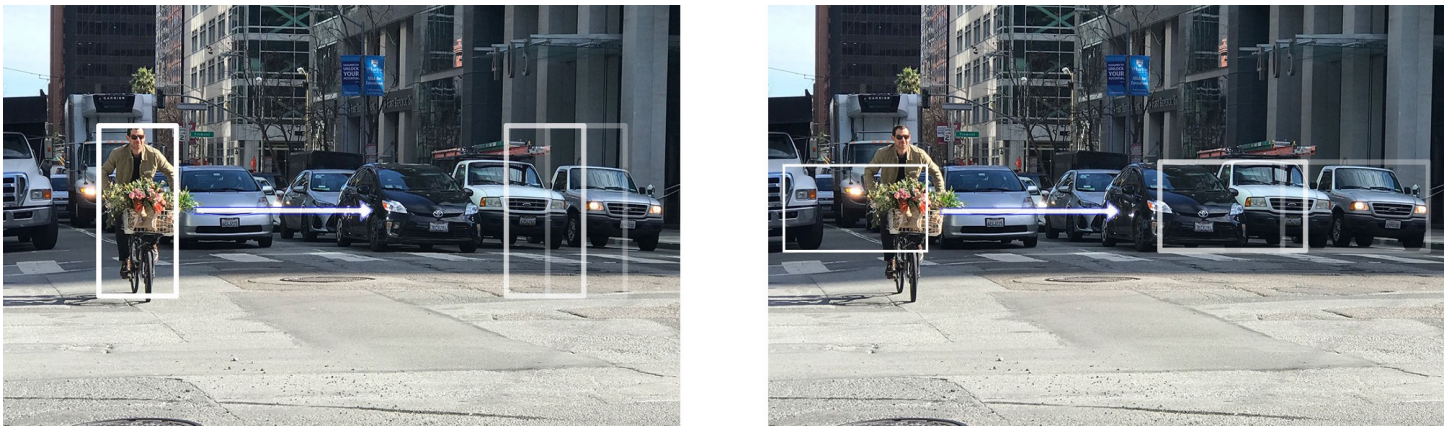
Part 1: What do we learn from region based object detectors (Faster R-CNN, R-FCN, FPN)?

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.

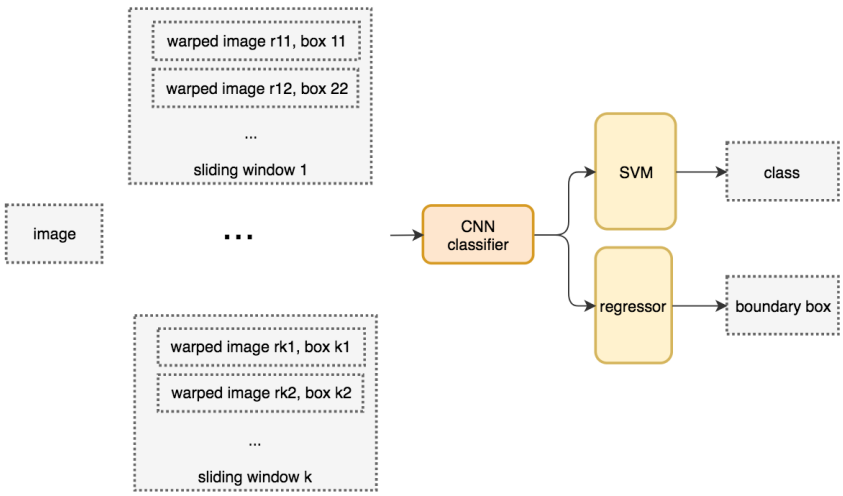


Sliding-windows (From right to left, up and down)

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 feed 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

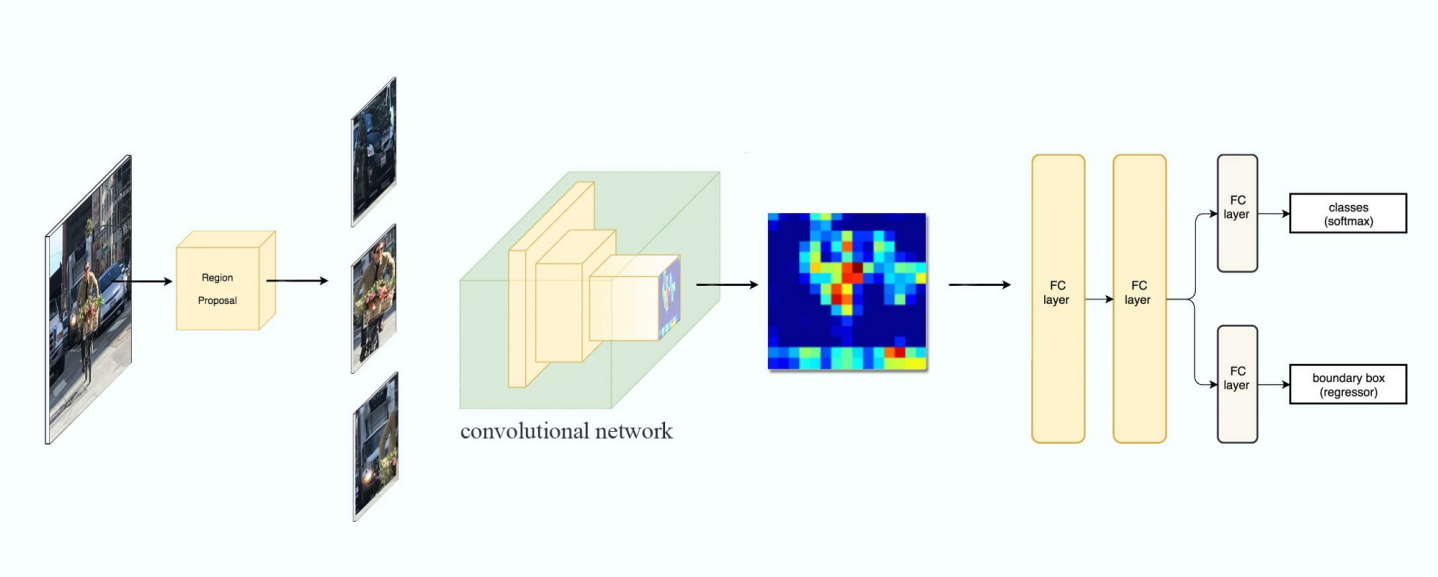
Instead of a brute force approach, we use a region proposal method to create **regions of interest (ROIs)** for object detection. In **selective**

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. The green rectangles are the target objects that we want to detect.

(Image source: van de Sande et al. ICCV'11)

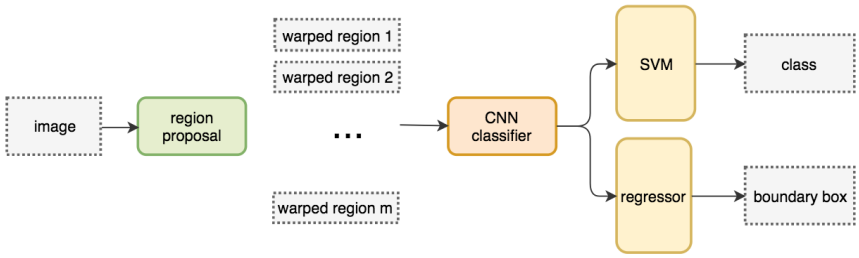
R-CNN

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.



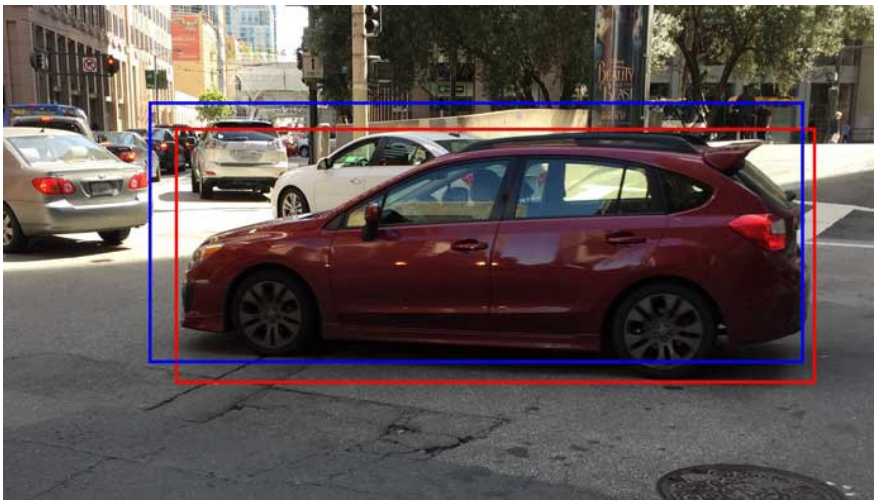
System flow for R-CNN

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.

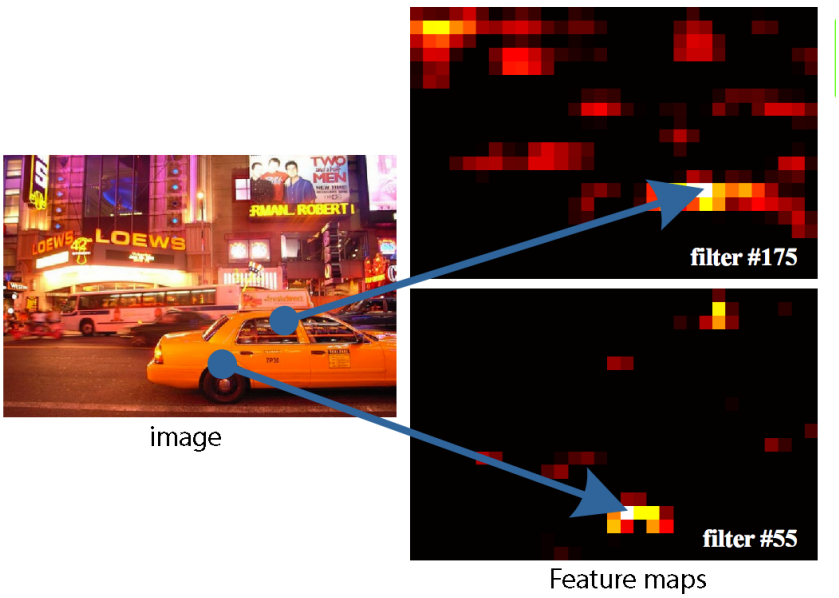


Use regression to refine the original ROI in blue to the red one.

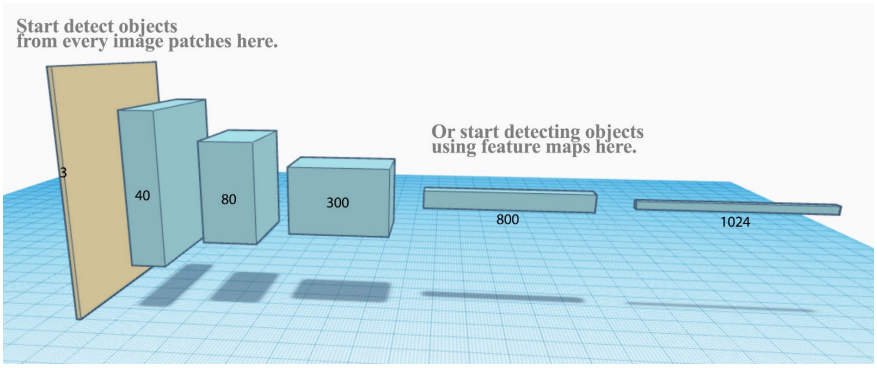
Fast R-CNN

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

The feature maps in a CNN represent spatial features in a more condensed space. Can we detect objects using the feature maps instead of the raw image?

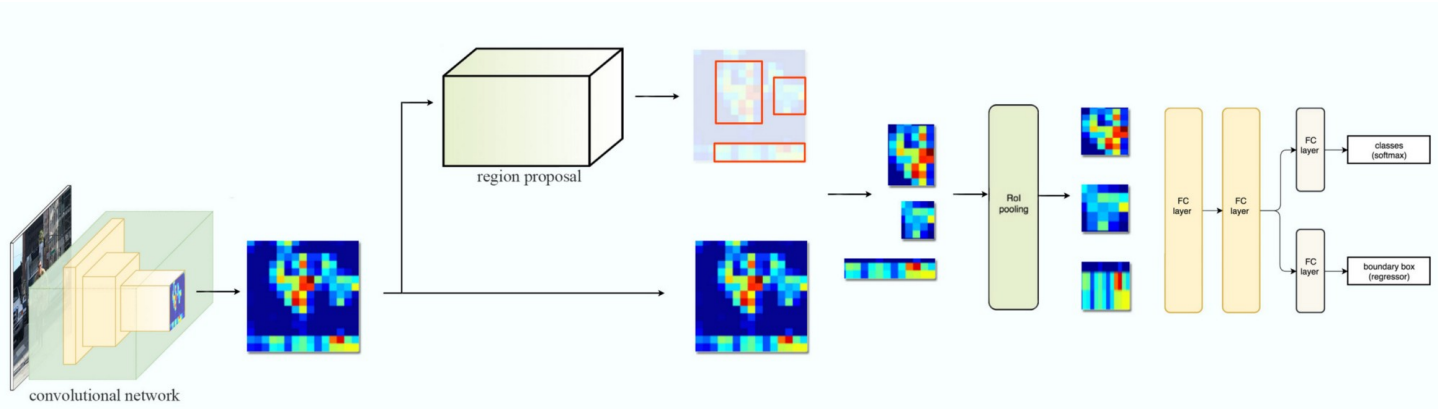


Source



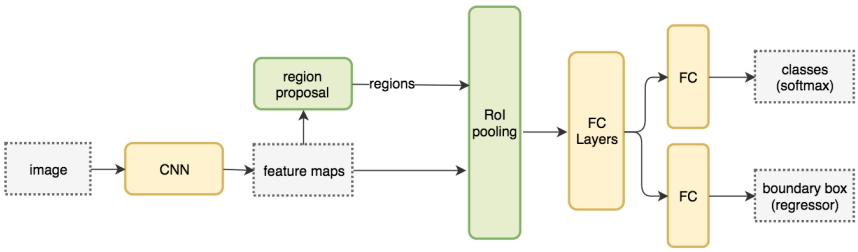
Calculate the ROIs from the feature maps instead.

Instead of extracting features for each image patch from scratch, we use a **feature extractor** (a CNN) to extract features for the whole image first. Then we apply the region proposal method on the feature maps directly. For example, Fast R-CNN selects the convolution layer **conv5** in VGG16 to generate 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:



System flow for Fast R-CNN

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

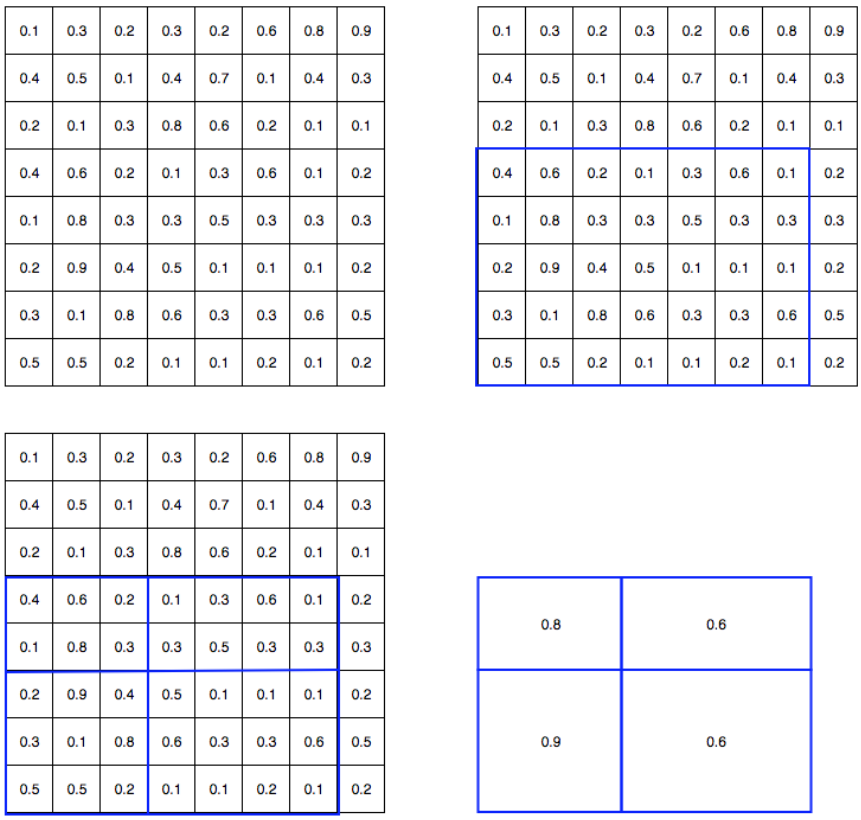
One major takeaway for Fast R-CNN is that the whole network (the feature extractor, the classifier, and the boundary box regressor) can be trained end-to-end with **multi-task losses** (classification loss and localization loss). This improves accuracy.

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 right below: 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.



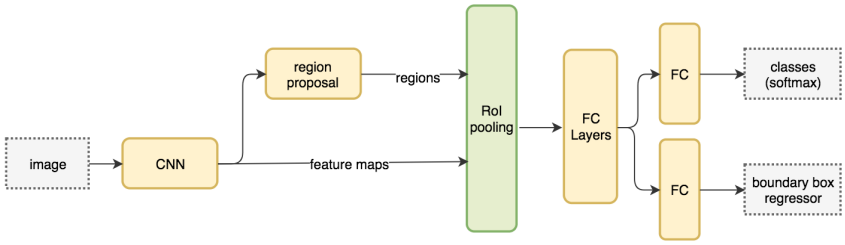
Input feature map (top left), output feature map (bottom right), blue box is the ROI (top right).

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.

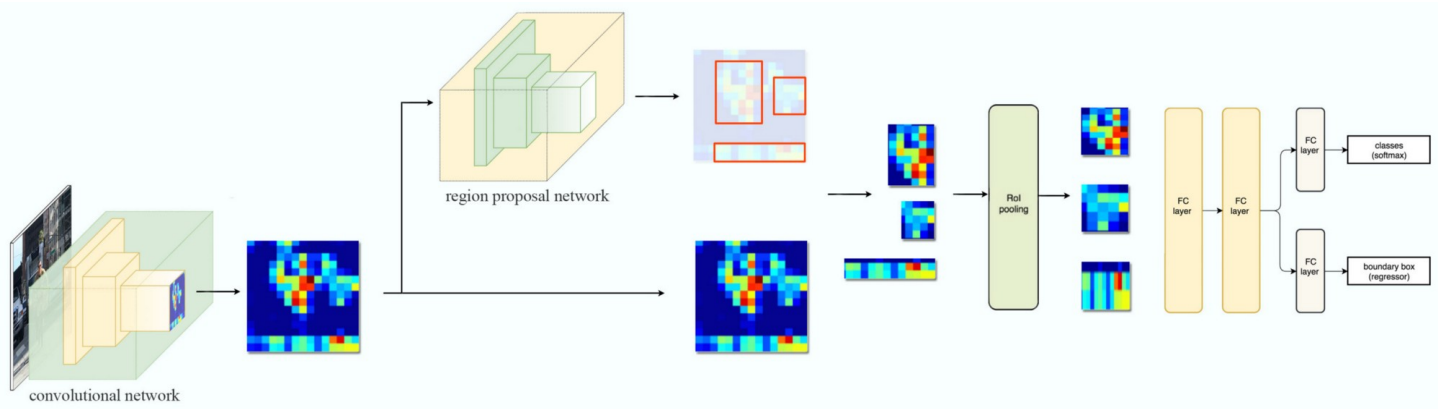
```
feature_maps = process(image)
ROIs = region_proposal(feature_maps)         # Expensive!
for ROI in ROIs
    patch = roi_pooling(feature_maps, ROI)
    results = detector2(patch)
```

Faster R-CNN adopts the same design as the Fast R-CNN except it replaces the region proposal method by an internal deep network. 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.

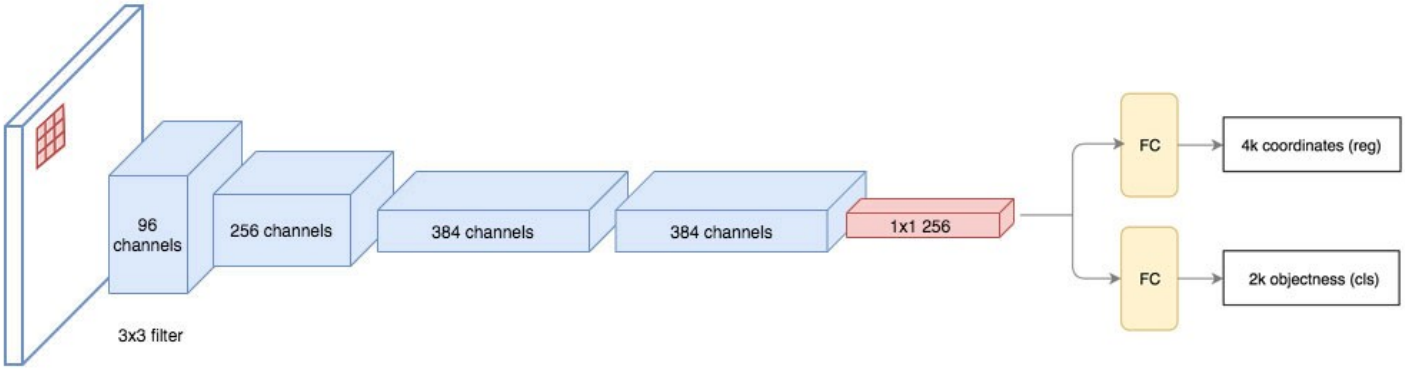
The network flow is the same but the region proposal is now replaced by a convolutional network.



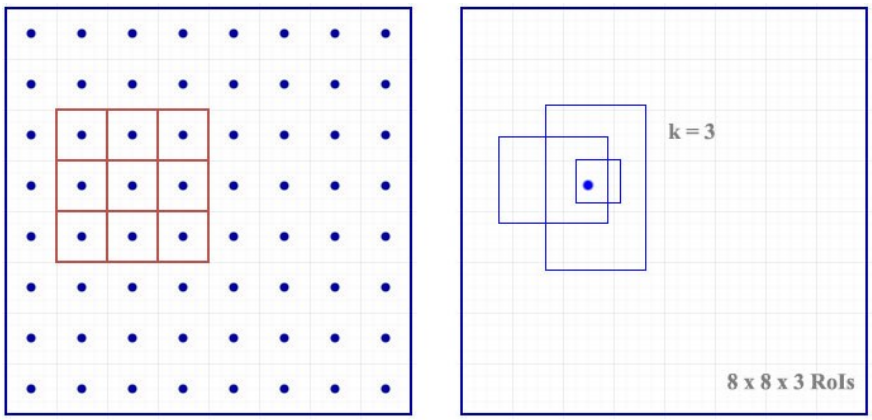
The external region proposal is replaced by an internal deep network.

Region proposal network

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

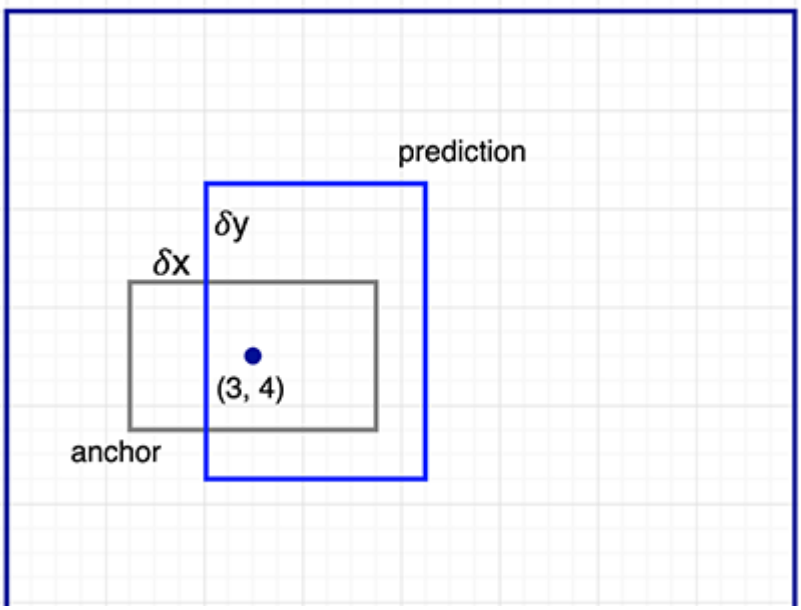


For each location in the feature maps, RPN makes **k** guesses. Therefore RPN outputs 4×k coordinates and 2×k scores per location. The diagram below shows the 8 × 8 feature maps with a 3× 3 filter, and it outputs a total of 8 × 8 × 3 ROIs (for k = 3). The right side diagram demonstrates the 3 proposals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

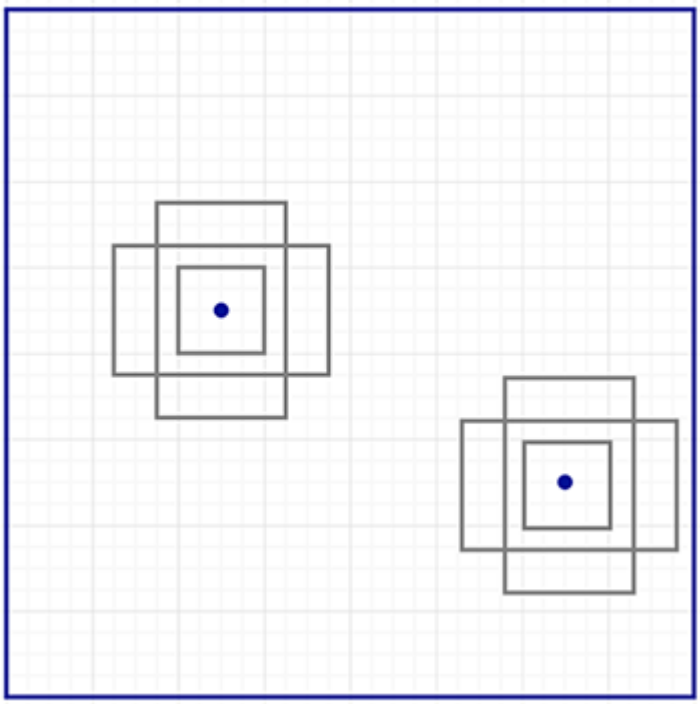


8 x 8 feature maps **3 proposals for each location**

Here, we get 3 guesses and we are allowed to refine our guesses later. Since we just need one to be correct, we will be better off if our initial guesses are very different. 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**. But constraints the value of those offsets, our guesses somehow relate to the anchors.

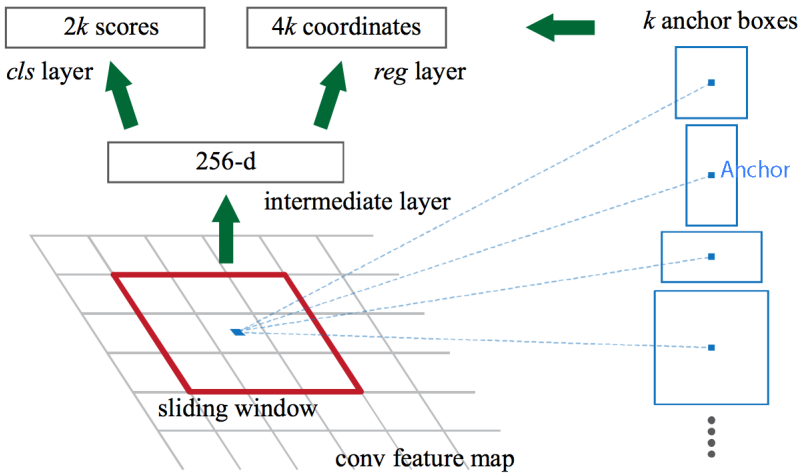


To make k predictions per location, we need k anchors centered at each location. Each prediction is associated with a specific anchor but different locations share the same anchor shapes.



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.

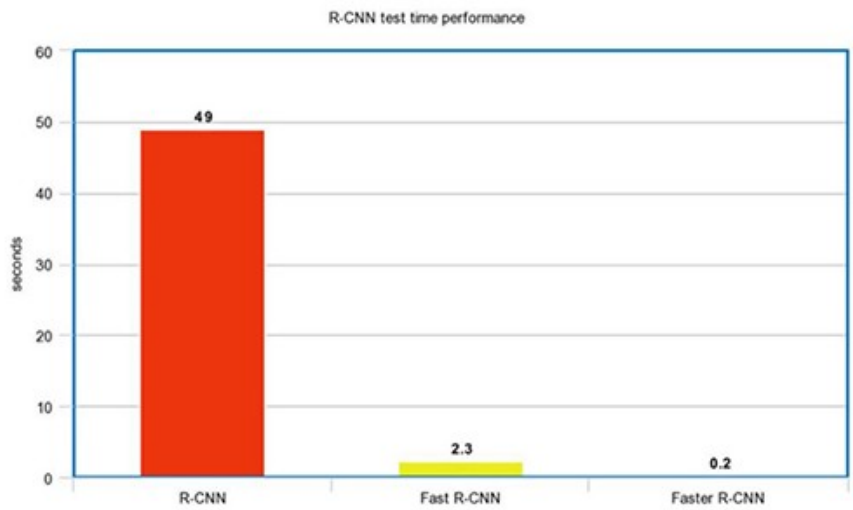
Faster R-CNN uses far more anchors. It deploys 9 anchor boxes: 3 different scales at 3 different aspect ratio. The ZF network extracts 256 features. Using 9 anchors per location, it generates 2×9 objectness scores and 4×9 coordinates per location.



Source

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

Performance for R-CNN methods

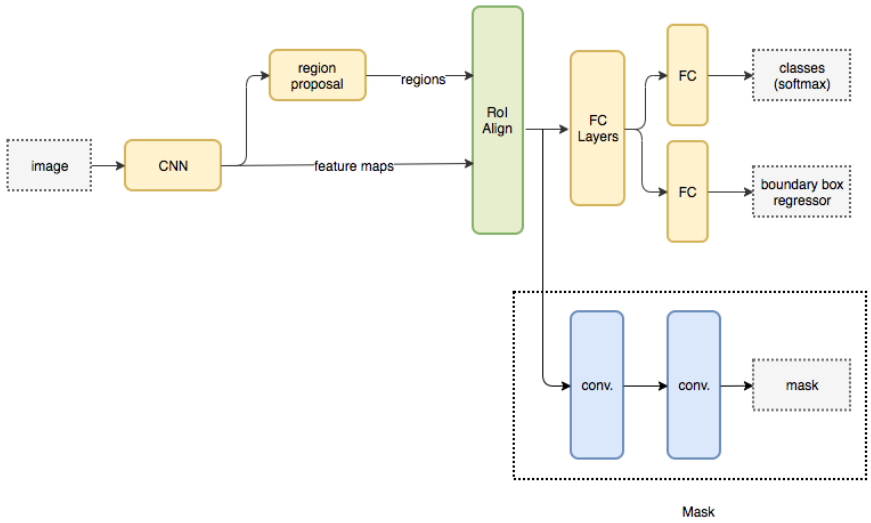


Mask R-CNN

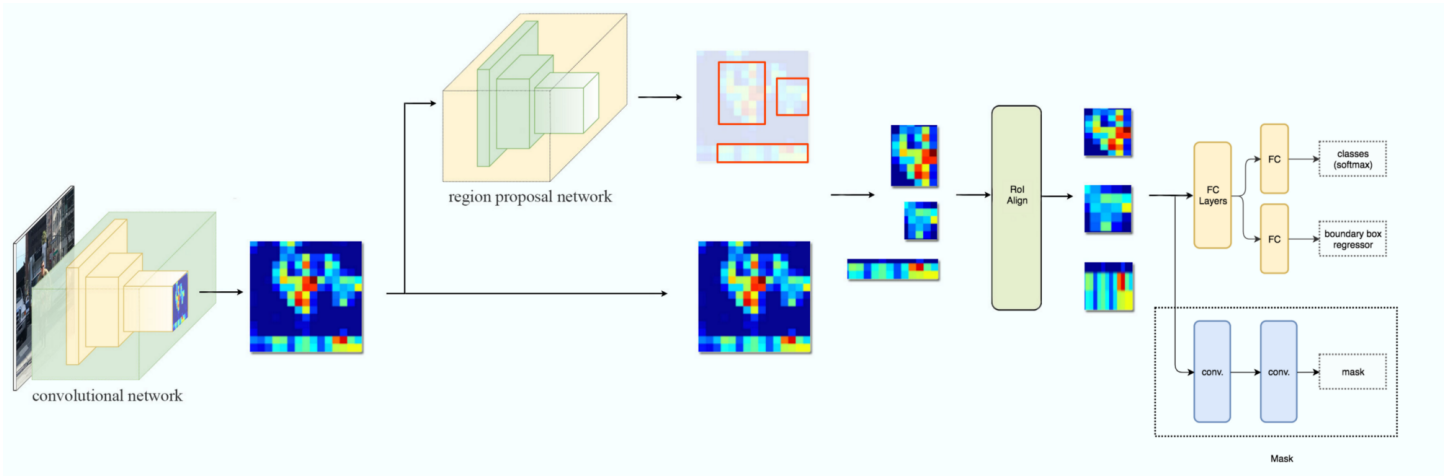
We are taking a short break into a related topic: image segmentation.

Image segmentation

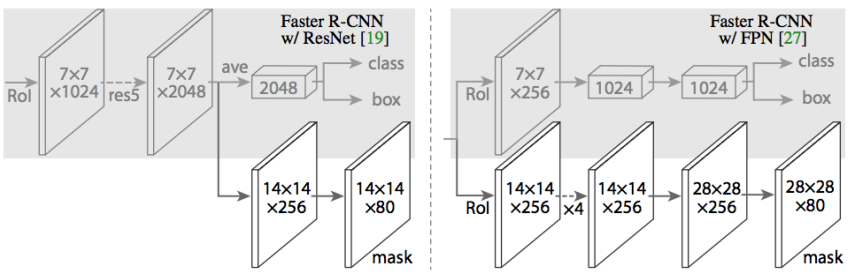
The Faster R-CNN builds all the ground works for feature extractions and ROI proposals. At first sight, performing image segmentation may require more detail analysis to colorize the image segments. By surprise, not only we can piggyback on this model, the extra work required is pretty simple. After the ROI pooling, we add 2 more convolution layers to build the mask.



Piggy back 2 convolutional layers to build the mask.



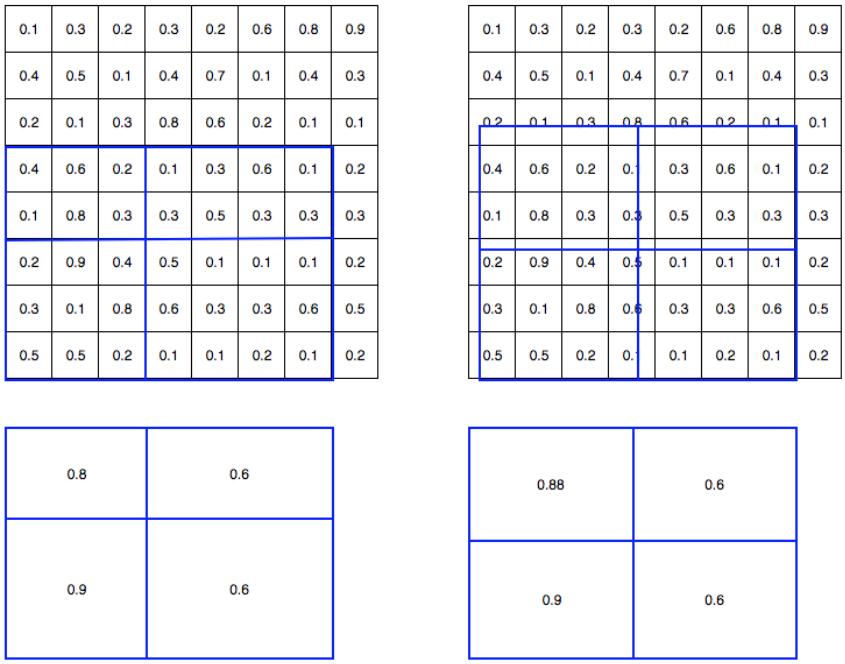
The Mask R-CNN paper provides one more variant (on the right) in building such mask. But the idea is pretty simple.



Source

ROI Align

Another major contribution of Mask R-CNN is the refinement of the ROI pooling. In ROI, the warping is digitalized (top left) as well as the warping (bottom left): the cell boundaries of the target feature map are forced to realign with the boundary of the input feature maps. Therefore, each target cells may not be in the same size. Mask R-CNN uses **ROI Align** which does not digitalize the boundary of the cells (top right) and make every target cell to have the same size (bottom right). It also applies interpolation to calculate the feature map values within the cell better. For example, by applying interpolation, the maximum feature value on the top left is changed from 0.8 to 0.88 now.



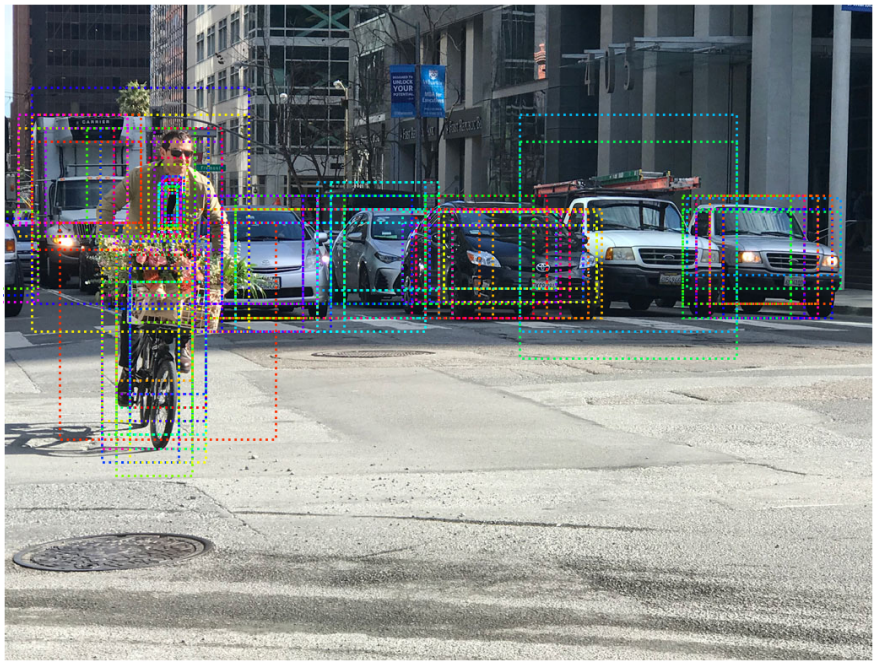
ROI Align makes significant improvements in the accuracy.

	AP	AP_{50}	AP_{75}	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}
RoIPool	23.6	46.5	21.6	28.2	52.7	26.9
RoIAlign	**30.9**	**51.8**	**32.1**	**34.0**	**55.3**	**36.4**
	+7.3	*+ 5.3*	*+10.5*	*+5.8*	*+2.6*	*+9.5*

Source

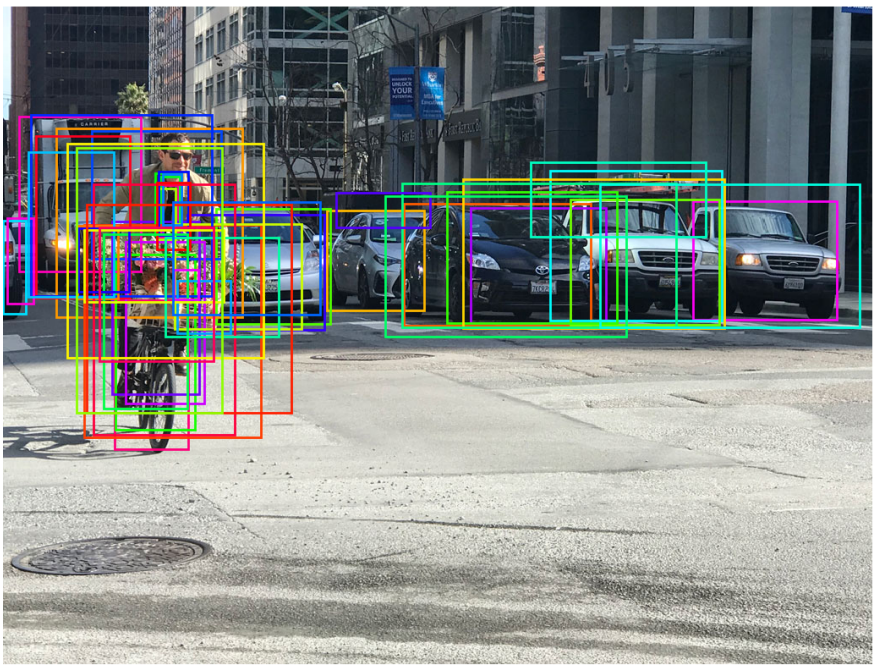
Mask R-CNN visualization

Let's visualize some of the major steps in Mask R-CNN/Faster R-CNN. Using the region proposal network, we make ROI proposals. The dotted rectangles below are those proposals but, for demonstration purpose, we decide to display those that have high final scores only.



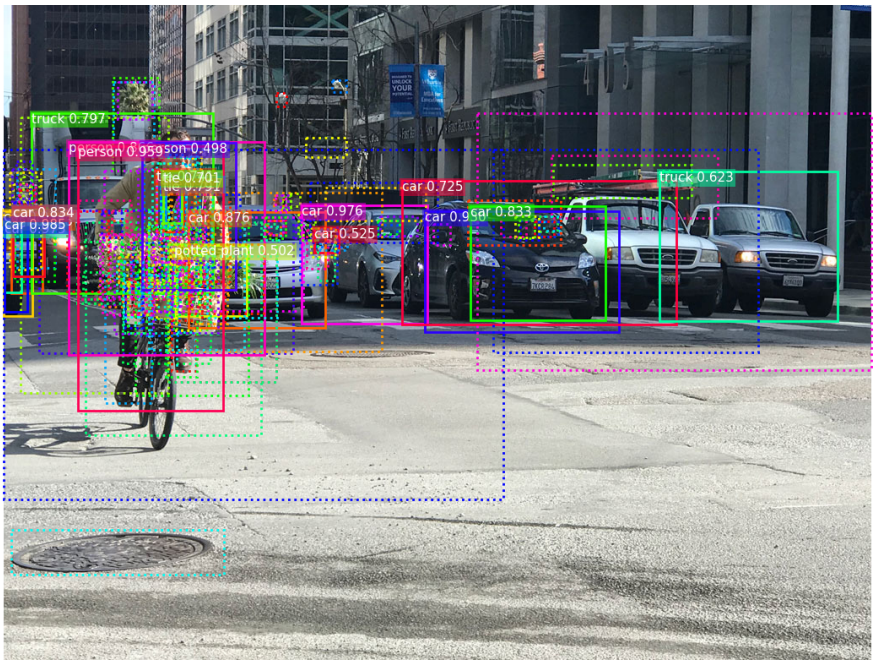
ROIs (before refinement)

Here are the boxes after boundary box refinements when we make final classification and localization predictions. The boundary box encloses the ground truth objects better.



Boundary boxes after refinement.

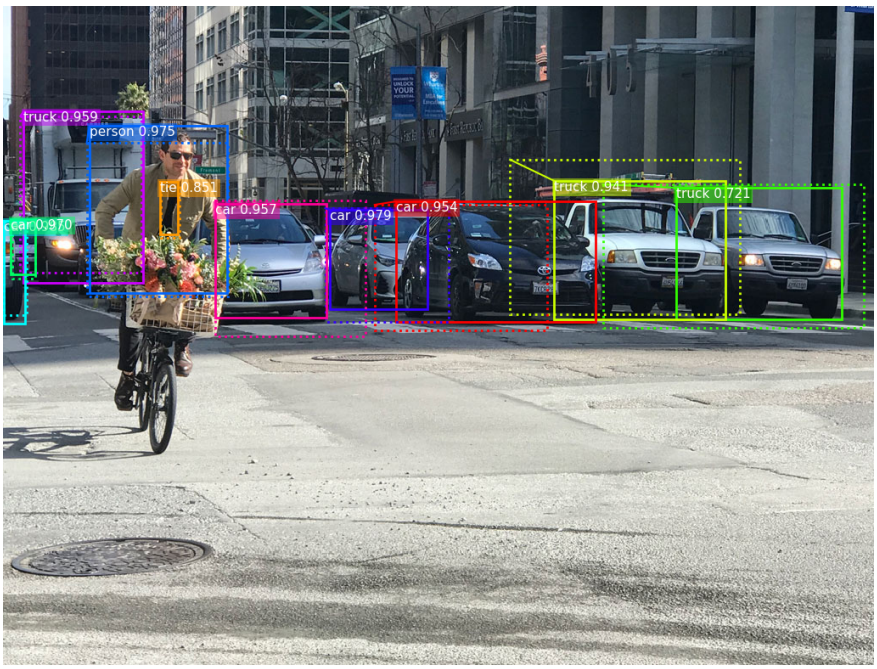
Just like Faster R-CNN, it performs object classification based on the ROIs (dotted lines) from RPN. The solid line is the boundary box refinements in the final predictions.



Classification with ROIs (dotted lines). Final refinements (solid lines).

Perform the per class non-maximum suppression (nms)

It groups highly-overlapped boxes for the same class and selects the most confidence prediction only. This avoids duplicates for the same object.

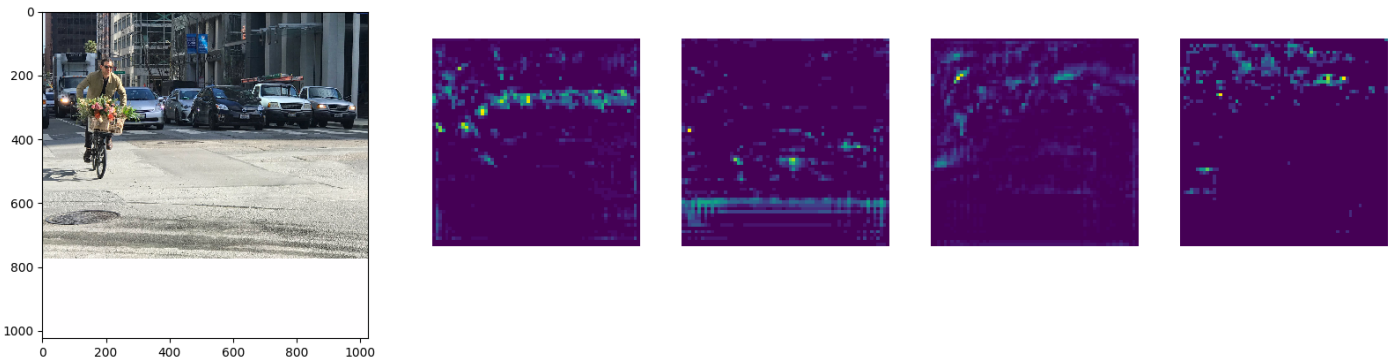


After nms. Solid line is the refined boundary box.

Here are our top final classification and boundary box predictions from the Faster R-CNN portion.

Top boundary box predictions.

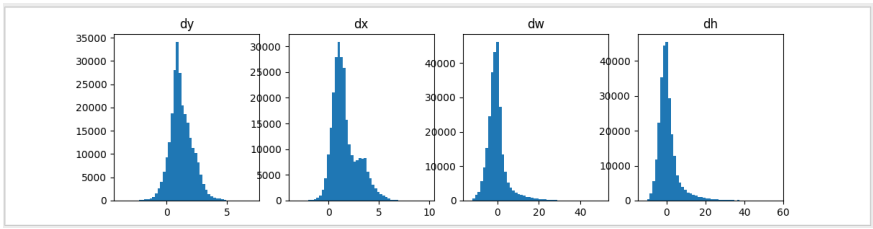
Here are the input picture and some of the feature maps used by the RPN. The first feature map shows high activations on where the cars line up.



Some feature maps for the RPN

Some of the corner locations of the boundary boxes:

And the distributions for the offsets from the anchors:

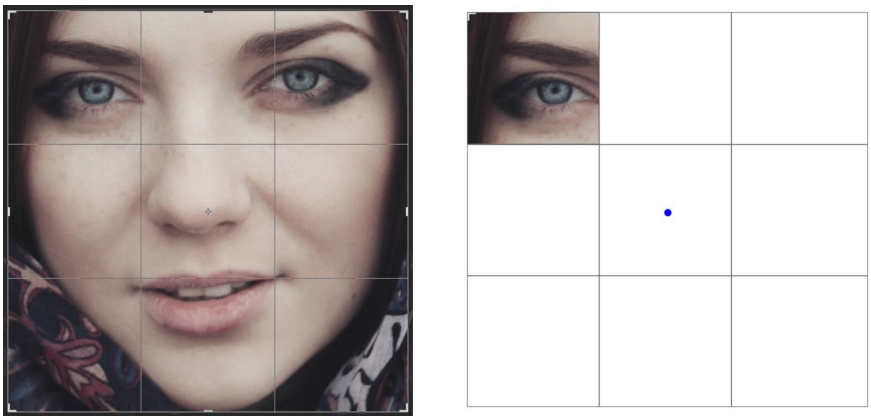


This is top final predictions from Mask R-CNN.

Final predictions from Mask R-NN.

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 right eye should be on 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.

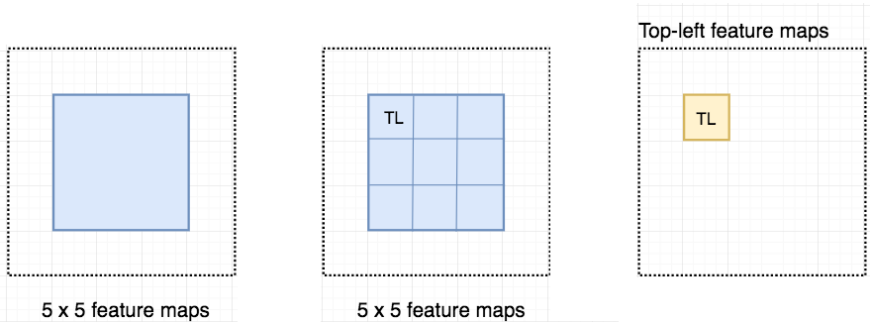
So why we go through all the trouble. In Faster R-CNN, the *detector* applies multiple fully connected layers to make predictions. With 2,000 ROIs, it can be expensive.

```
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 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.



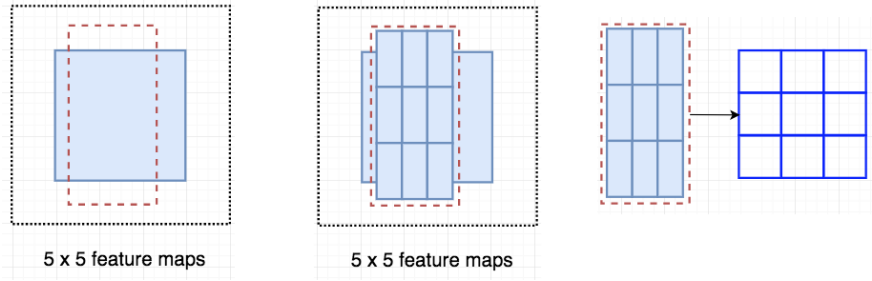
Create a new feature map from the left to detect the top left corner of an object.

Since we divide the square into 9 parts, we 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

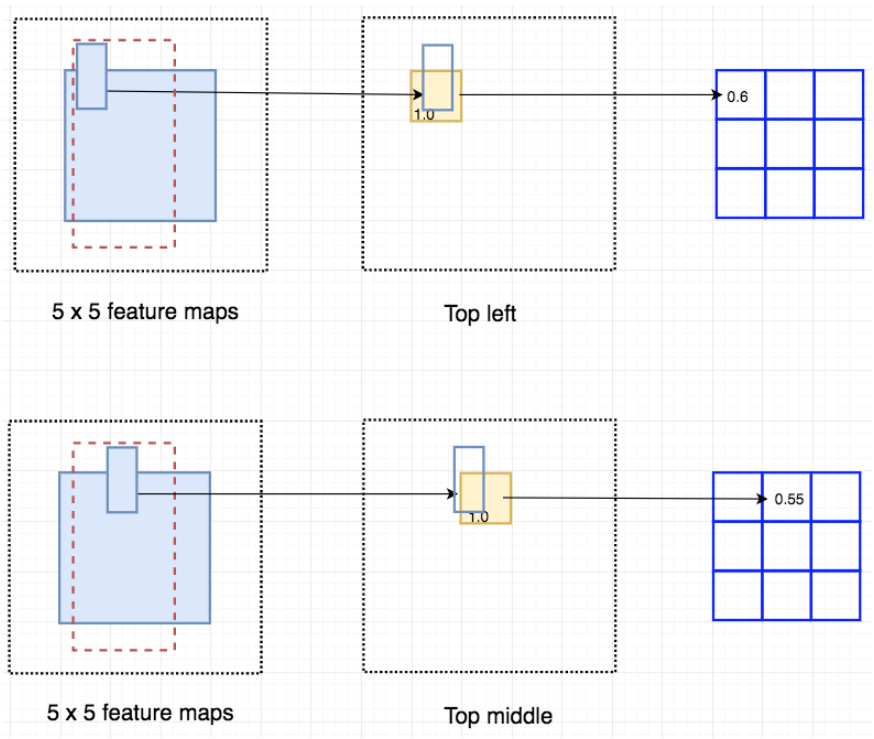
Let's say the dotted red rectangle below is the ROI proposed. We divide it into 3 × 3 regions and ask how likely each region contains the

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.



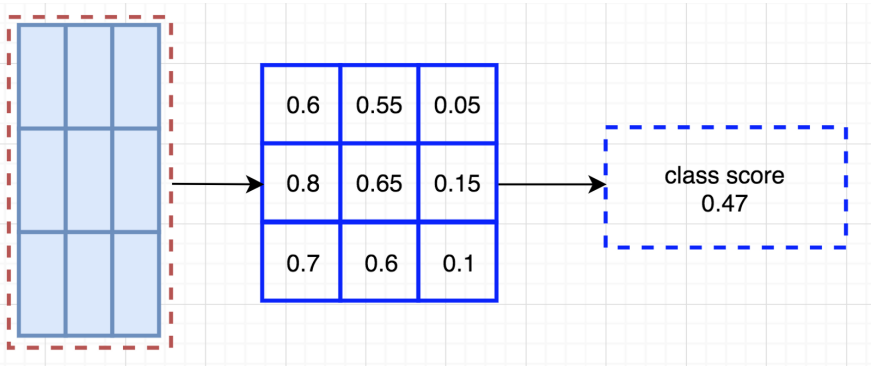
Apply ROI onto the feature maps to output a 3 x 3 array.

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]

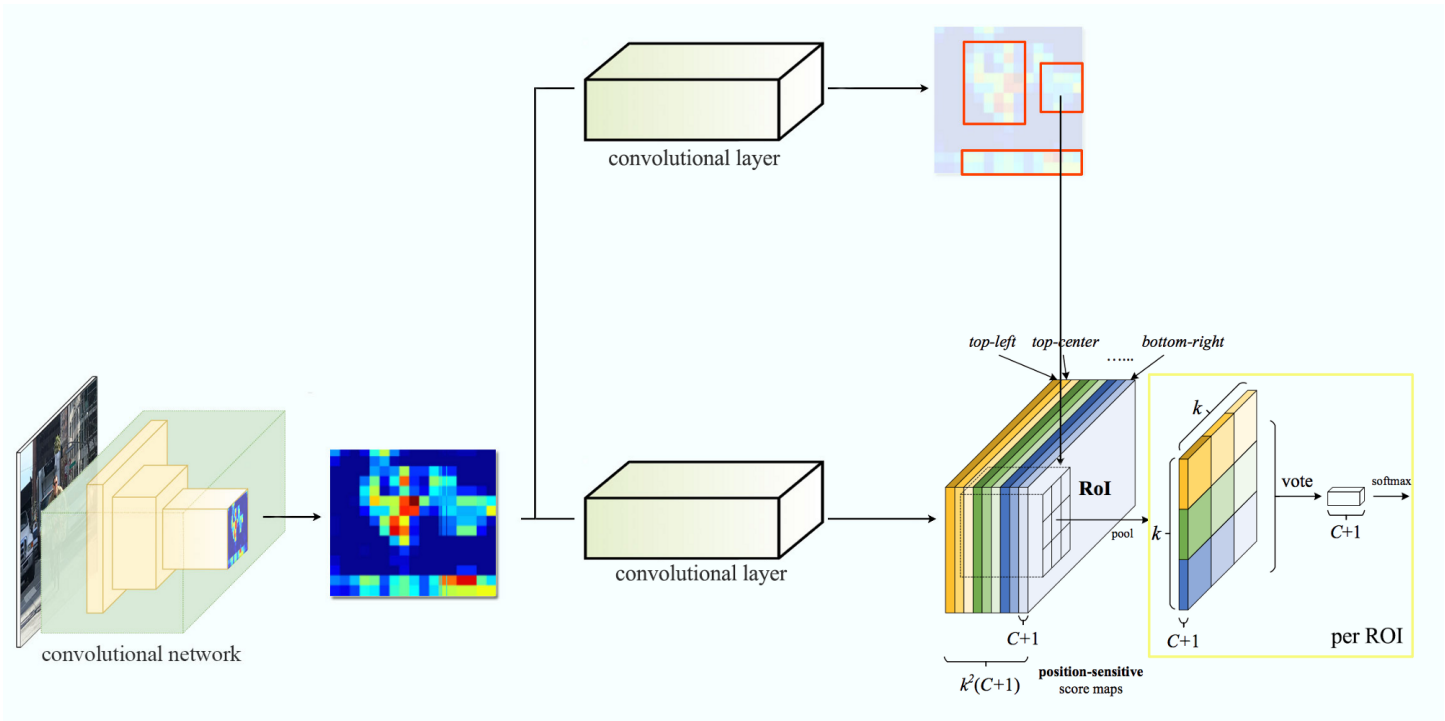
After calculating all the values for the position-sensitive ROI pool, the class score is the average of all its elements.



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.



Further reading on SSD & YOLO

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.

Mask R-CNN implementation in TensorFlow.

R-FCN implementation in MXNet.

R-FCN implementation in Caffe and MATLAB.

R-FCN implementation in TensorFlow.