7

Object Detection and Instance Segmentation with CNN

Until now, in this book, we have been mostly using **convolutional neural networks** (**CNNs**) for classification. Classification classifies the whole image into one of the classes with respect to the entity having the maximum probability of detection in the image. But what if there is not one, but multiple entities of interest and we want to have the image associated with all of them? One way to do this is to use tags instead of classes, where these tags are all classes of the penultimate Softmax classification layer with probability above a given threshold. However, the probability of detection here varies widely by size and placement of entity, and from the following image, we can actually say, *How confident is the model that the identified entity is the one that is claimed?* What if we are very confident that there is an entity, say a dog, in the image, but its scale and position in the image is not as prominent as that of its owner, a *Person* entity? So, a *Multi-Class Tag* is a valid way but not the best for this purpose:

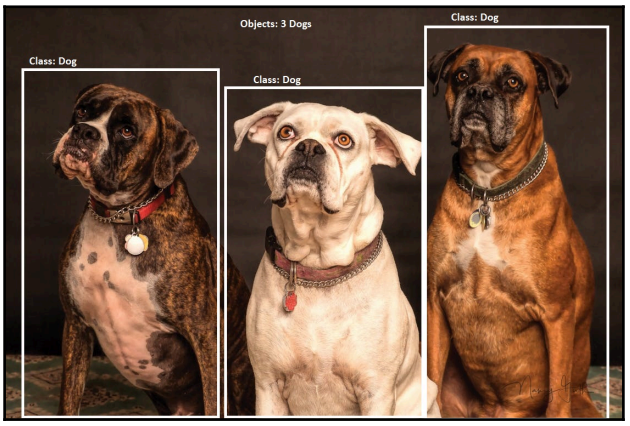
In this chapter, we will cover the following topics:

- The differences between object detection and image classification
- Traditional, non-CNN approaches for object detection
- Region-based CNN and its features
- Fast R-CNN
- Faster R-CNN
- Mask R-CNN

The differences between object detection and image classification

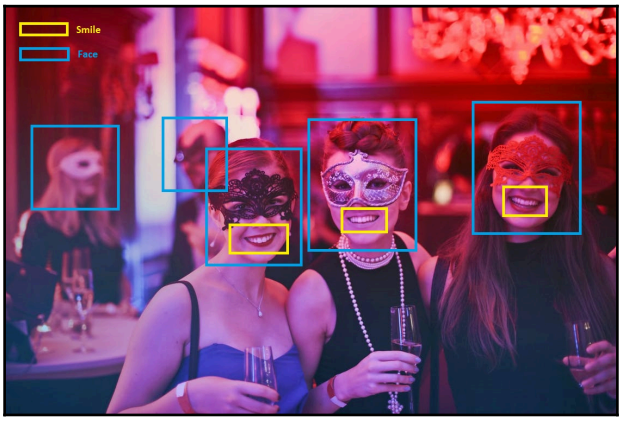
Let's take another example. You are watching the movie *101 Dalmatians*, and you want to know how many Dalmatians you can actually count in a given movie scene from that movie. Image Classification could, at best, tell you that there is at least one dog or one *Dalmatian* (depending upon which level you have trained your classifier for), but not exactly how many of them there are.

Another issue with classification-based models is that they do not tell you where the identified entity in the image is. Many times, this is very important. Say, for example, you saw your neighbor's dog playing with him (*Person*) and his cat. You took a snap of them and wanted to extract the image of the dog from there to search on the web for its breed or similar dogs like it. The only problem here is that searching the whole image might not work, and without identifying individual objects from the image, you have to do the cut-extract-search job manually for this task, as shown in the following image:



So, you essentially need a technique that not only identifies the entities in an image but also tells you their placement in the image. This is what is called **object detection**. Object detection gives you bounding boxes and class labels (along with the probability of detection) of all the entities identified in an image. The output of this system can be used to empower multiple advanced use cases that work on the specific class of the objects detected.

Take, for example, the Facial Recognition feature that you have in Facebook, Google Photos, and many other similar apps. In it, before you identify *who is* there in an image taken in at a party, you need to detect all the faces in that image; then you can pass these faces through your face recognition/classification module to get/classify their names. So, the Object nomenclature in object detection is not limited to linguistic entities but includes anything that has specific boundaries and enough data to train the system, as shown in the following image:



Now, if you want to find out how many of the guests present at your party were actually **enjoying** it, you can even run an object detection for **Smiling Faces** or a **Smile Detector**. There are very powerful and efficient trained models of object detectors available for most of the detectable human body parts (eye, face, upper body, and so on), popular human expressions (such as a smile), and many other general objects as well. So, the next time you use the **Smile Shutter** on your smartphone (a feature made to automatically click the image when most of the faces in the scene are detected as smiling), you know what is powering this feature.

Why is object detection much more challenging than image classification?

From our understanding of CNN and image classification so far, let's try to understand how we can approach the object detection problem, and that should logically lead us to the discovery of the underlying complexity and challenges. Assume we are dealing with monochromatic images for simplicity.

Any object detection at a high level may be considered a combination of two tasks (we will refute this later):

- Getting the right bounding boxes (or as many of them to filter later)
- Classifying the object in that bounding box (while returning the classification effectiveness for filtering)

So, object detection not only has to cater to all the challenges of image classification (second objective), but also faces new challenges of finding the right, or as many as possible, bounding boxes. As we already know how to use CNNs for the purpose of image classification, and the associated challenges, we can now concentrate on our first task and explore how effective (classification accuracy) and efficient (computational complexity) our approach is—or rather how challenging this task is going to be.

So, we start with randomly generating bounding boxes from the image. Even if we do not worry about the computational load of generating so many candidate boxes, technically termed as **Region Proposals** (regions that we send as proposals for classifying objects), we still need to have some mechanism for finding the best values for the following parameters:

- Starting (or center) coordinates to extract/draw the candidate bounding box
- Length of the candidate bounding box
- Width of the candidate bounding box
- Stride across each axis (distance from one starting location to another in the x-horizontal axis and y-vertical axis)

Let's assume that we can generate such an algorithm that can give us the most optimal value of these parameters. Still, will one value for these parameters work in most of the cases, or in fact, in some general cases? From our experience, we know that each object will have a different scale, so we know that one fixed value for L and W for these boxes will not work. Also, we can understand that the same object, say Dog, may be present in varying proportions/scales and positions in different images, as in some of our earlier examples. So this confirms our belief that we need boxes of not only different scales but also different sizes.

Let's assume that, correcting from the previous analogy, we want to extract *N* number of candidate boxes per starting coordinate in the image, where *N* encompasses most of the sizes/scales that may fit our classification problem. Although that seems to be a rather challenging job in itself, let's assume we have that magic number and it is far from a combination of *L[1,l-image] x W[1,w-image]* (all combinations of *L* and *W* where length is a set of all integers between 1 and the length of the actual image and breadth is from 1 to the breadth of the image); that will lead us to *l*w* boxes per coordinate:

Then, another question is about how many starting coordinates we need to visit in our image from where we will extract these *N* boxes each, or the Stride. Using a very big stride will lead us to extract sub-images in themselves, instead of a single homogeneous object that can be classified effectively and used for the purpose of achieving some of the objectives in our earlier examples. Conversely, too short a stride (say, 1 pixel in each direction) may mean a lot of candidate boxes.

From the preceding illustration, we can understand that even after hypothetically relaxing most of the constraints, we are nowhere close to making a system that we can fit in our Smartphones to detect smiling selfies or even bright faces in real time (even after an hour in fact). Nor can it have our robots and self-driving cars identify objects as they move (and navigate their way by avoiding them). This intuition should help us appreciate the advancements in the field of object detection and why it is such an impactful area of work.

Traditional, nonCNN approaches to object detection

Libraries such as OpenCV and some others saw rapid inclusion in the software bundles for Smartphones, Robotic projects, and many others, to provide detection capabilities of specific objects (face, smile, and so on), and Computer Vision like benefits, though with some constraints even before the prolific adoption of CNN.

CNN-based research in this area of object detection and Instance Segmentation provided many advancements and performance enhancements to this field, not only enabling large-scale deployment of these systems but also opening avenues for many new solutions. But before we plan to jump into CNN based advancements, it will be a good idea to understand how the challenges cited in the earlier section were answered to make object detection possible in the first place (even with all the constraints), and then we will logically start our discussion about the different researchers and the application of CNN to solve other problems that still persist with the use of traditional approaches.

Haar features, cascading classifiers, and the Viola-Jones algorithm

Unlike CNN, or the deepest learning for that matter, which is known for its capability of generating higher conceptual features automatically, which in-turn gives a major boost to the classifier, in case of traditional machine learning applications, such features need to be hand crafted by SMEs.

As we may also understand from our experience working on CPU-based machine learning classifiers, their performance is affected by high dimensionality in data and the availability of too many features to apply to the model, especially with some of the very popular and sophisticated classifiers such as **Support Vector Machines** (**SVM**), which used to be considered state-of-the-art until some time ago.

In this section, we will understand some of the innovative ideas drawing inspirations from different fields of science and mathematics that led to the resolution of some of the cited challenges above, to fructify the concept of real-time object detection in non-CNN systems.

Haar Features

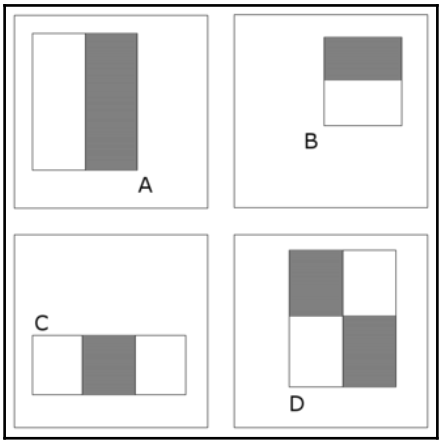
Haar or Haar-like features are formations of rectangles with varying pixel density. Haar features sum up the pixel intensity in the adjacent rectangular regions at specific locations in the detection region. Based on the difference between the sums of pixel intensities across regions, they categorize the different subsections of the image.

Haar-like features have their name attributed to the mathematics term of Haar wavelet, which is a sequence of rescaled square-shaped functions that together form a wavelet family or basis.

Because Haar-like features work on the difference between pixel intensities across regions, they work best with monochrome images. This is also the reason the images used earlier and in also this section are monochrome for better intuition.

These categories can be grouped into three major groups, as follows:

- Two rectangle features
- Three rectangle features
- Four rectangle features



Haar-like Features

With some easy tricks, the computation of varying intensities across the image becomes very efficient and can be processed at a very high rate in real time.

Cascading classifiers

Even if we can extract Haar features from a particular region very quickly, it does not solve the problem of extracting such features from a lot of different places in the image; this is where the concept of cascading features comes in to help. It was observed that only 1 in 10,000 sub-regions turns positive for faces in classification, but we have to extract all features and run the whole classifier across all regions. Further, it was observed that by using just a few of the features (two in the first layer of the cascade), the classifier could eliminate a very high proportion of the regions (50% in the first region of the cascade). Also, if the sample consists of just these reduced region samples, then only slightly more features (10 features in the second layer of the cascade) are required for a classifier that can weed out a lot more cases, and so on. So we do classification in layers, starting with a classifier that requires very low computational power to weed out most of the subregions, gradually increasing the computation load required for the remaining subset, and so on.

The Viola-Jones algorithm

In 2001, Paul Viola and Michael Jones proposed a solution that could work well to answer some of the preceding challenges, but with some constraints. Though it is an almost two decades old algorithm, some of the most popular computer vision software to date, or at least till recently, used to embed it in some form or another. This fact makes it very important to understand this very simple, yet powerful, algorithm before we move on to CNN-based approaches for Region Proposal.

OpenCV, one of the most popular software libraries for computer vision, uses cascading classifiers as the predominant mode for object detection, and Haar-featuring-like Cascade classifier is very popular with OpenCV. A lot of pretrained Haar classifiers are available for this for multiple types of general objects.

This algorithm is not only capable of delivering detections with high **TPRs** (**True Positive Rates**) and low **FPRs** (**False Positive Rates**), it can also work in real time (process at least two frames per second).

High TPR combined with Low FPR is a very important criterion for determining the robustness of an algorithm.

The constraints of their proposed algorithm were the following:

- It could work only for detecting, not recognizing faces (they proposed the algorithm for faces, though the same could be used for many other objects).
- The faces had to be present in the image as a frontal view. No other view could be detected.

At the heart of this algorithm are the Haar (like) Features and Cascading Classifiers. Haar Features are described later in a subsection. The Viola-Jones algorithm uses a subset of Haar features to determine general features on a face such as:

- Eyes (determined by a two-rectangle feature (horizontal), with a dark horizontal rectangle above the eye forming the brow, followed by a lighter rectangle below)
- Nose (three-rectangle feature (vertical), with the nose as the center light rectangle and one darker rectangle on either side on the nose, forming the temple), and so on

These fast-to-extract features can then be used to make a classifier to detect (distinguish) faces (from non-faces).

Haar features, with some tricks, are very fast to compute.

Viola-Jones algorithm and Haar-like Features for detecting faces

These Haar-like features are then used in the cascading classifiers to expedite the detection problem without losing the robustness of detection.

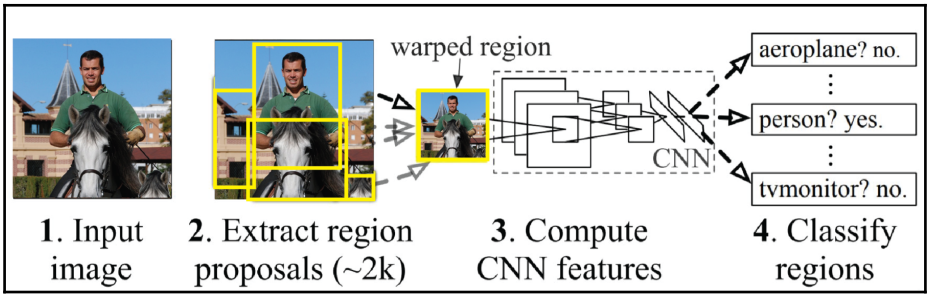
The Haar Features and cascading classifiers thus led to some of the very robust, effective, and fast individual object detectors of the previous generation. But still, the training of these cascades for a new object was very time consuming, and they had a lot of constraints, as mentioned before. That is where the new generation CNN-based object detectors come to the rescue.

In this chapter, we have covered only the basis of Haar-Cascades or Haar features (in the non-CNN category) as they remained predominant for a long time and were the basis of many new types. Readers are encouraged to also explore some of the later and much effective SIFT and HOG-based features/cascades (associated papers are given in the *References* section).

R-CNN – Regions with CNN features

In the 'Why is object detection much more challenging than image classification?' section, we used a non-CNN method to draw region proposals and CNN for classification, and we realized that this is not going to work well because the regions generated and fed into CNN were not optimal. R-CNN or regions with CNN features, as the name suggests, flips that example completely and use CNN to generate features that are classified using a (non-CNN) technique called **SVM** (**Support Vector Machines**)

R-CNN uses the sliding window method (much like we discussed earlier, taking some *L x W* and stride) to generate around 2,000 regions of interest, and then it converts them into features for classification using CNN. Remember what we discussed in the transfer learning chapter—the last flattened layer (before the classification or softmax layer) can be extracted to transfer learning from models trained on generalistic data, and further train them (often requiring much less data as compared to a model with similar performance that has been trained from scratch using domain-specific data) to model domain-specific models. R-CNNs also use a similar mechanism to improve their effectiveness on specific object detection:



R-CNN – Working

The original paper on R-CNN claims that on a PASCAL VOC 2012 dataset, it has improved the **mean average precision** (**mAP**) by more than 30% relative to the previous best result on that data while achieving a mAP of 53.3%.

We saw very high precision figures for the image classification exercise (using CNN) over the ImageNet data. Do not use that figure with the comparison statistics given here, as not only are the datasets used different (and hence not comparable), but also the tasks in hand (classification versus object detection) are quite different, and object detection is much more challenging a task than image classification.

PASCAL **VOC** (**Visual Object Challenge**): Every area of research requires some sort of standardized dataset and standard KPIs to compare results across different studies and algorithms. Imagenet, the dataset we used for image classification, cannot be used as a standardized dataset for object detection, as object-detection requires (train, test, and validation set) data labeled with not only the object class but also its position. ImageNet does not provide this. Therefore, in most object detection studies, we may see the use of a standardized object-detection dataset, such as PASCAL VOC. The PASCAL VOC dataset has 4 variants so far, VOC2007, VOC2009, VOC2010, and VOC2012. VOC2012 is the latest (and richest) of them all.

Another place we stumbled at was the differing scales (and location) of the regions of interest, *recognition using region*. This is what is called the **localization** challenge; it is solved in R-CNN by using a varying range of receptive fields, starting from as high a region with 195 x 195 pixels and 32 x 32 strides, to lesser downwards.

This approach is called **recognition using region**.

Wait a minute! Does that ring a bell? We said that we will use CNN to generate features from this region, but CNN uses a constant-size input to produce a fixed-size flattened layer. We do require fixed-size features (flattened vector size) as input to our SVMs, but here the input region size is changing. So how does that work? R-CNN uses a popular technique called **Affine Image Warping** to compute a fixed-size CNN input from each region proposal, regardless of the region's shape.

In geometry, an affine transformation is the name given to a transformation function between affine spaces that preserves points, straight lines, and planes. Affine spaces are structures that generalize the properties of Euclidian spaces while preserving only the properties related to parallelism and respective scale.

Besides the challenges that we have covered, there exists another challenge that is worth mentioning. The candidate regions that we generated in the first step (on which we performed classification in the second step) were not very accurate, or they were lacking tight boundaries around the object identified. So we include a third stage in this method, which improves the accuracy of the bounding boxes by running a regression function (called **bounding-box regressors**) to identify the boundaries of separation.

R-CNN proved to be very successful when compared to the earlier end-to-end non-CNN approaches. But it uses CNN only for converting regions to features. As we understand, CNNs are very powerful for image classifications as well, but because our CNN will work only on input region images and not on flattened region features, we cannot use it here directly. In the next section, we will see how to overcome this obstacle.

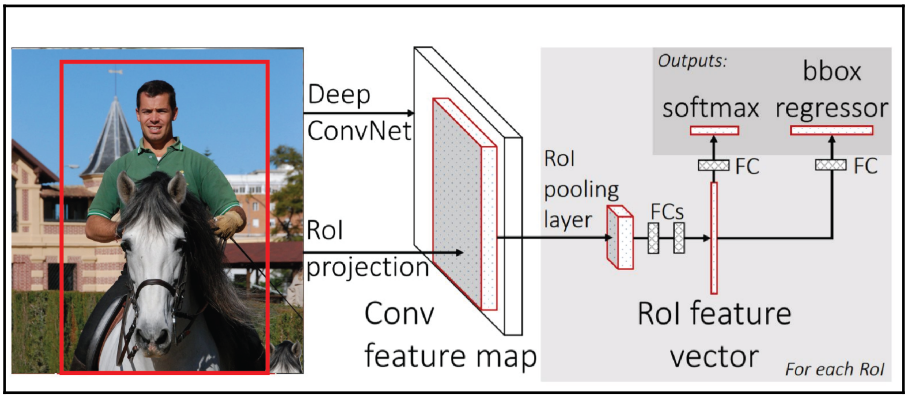
R-CNN is very important to cover from the perspective of understanding the background use of CNN in object detection as it has been a giant leap from all non-CNN-based approaches. But because of further improvements in CNN-based object detection, as we will discuss next, R-CNN is not actively worked upon now and the code is not maintained any longer.

Fast R-CNN – fast region-based CNN

Fast R-CNN, or Fast Region-based CNN method, is an improvement over the previously covered R-CNN. To be precise about the improvement statistics, as compared to R-CNN, it is:

- 9x faster in training
- 213x faster at scoring/servicing/testing (0.3s per image processing), ignoring the time spent on region proposals
- Has higher mAP of 66% on the PASCAL VOC 2012 dataset

Where R-CNN uses a smaller (five-layer) CNN, Fast R-CNN uses the deeper VGG16 network, which accounts for its improved accuracy. Also, R-CNN is slow because it performs a ConvNet forward pass for each object proposal without sharing computation:



Fast R-CNN: Working

In Fast R-CNN, the deep VGG16 CNN provides essential computations for all the stages, namely:

- **Region of Interest** (**RoI**) computation
- Classification Objects (or background) for the region contents
- Regression for enhancing the bounding box

The input to the CNN, in this case, is not raw (candidate) regions from the image, but the (complete) actual image itself; the output is not the last flattened layer but the convolution (map) layer before that. From the so-generated convolution map, a the RoI pooling layer (a variant of max-pooling) is used to generate the flattened fixed-length RoI corresponding to each object proposal are generated, which are then passed through some **fully connected** (**FC**) layers.

The RoI pooling is a variant of max pooling (that we used in our initial chapters in this book), in which output size is fixed and input rectangle is a parameter.

The RoI pooling layer uses max pooling to convert the features inside any valid region of interest into a small feature map with a fixed spatial extent.

The output from the penultimate FC layer is then used for both:

- Classification (SoftMax layer) with as many classes as object proposals, +1 additional class for the background (none of the classes found in the region)
- Sets of regressors that produce the four numbers (two numbers denoting the x, y coordinates of the upper-left corner for the box for that object, and the next two numbers corresponding to the height and width of that object found in that region) for each object-proposal that is required to make bounding boxes precise for that particular object

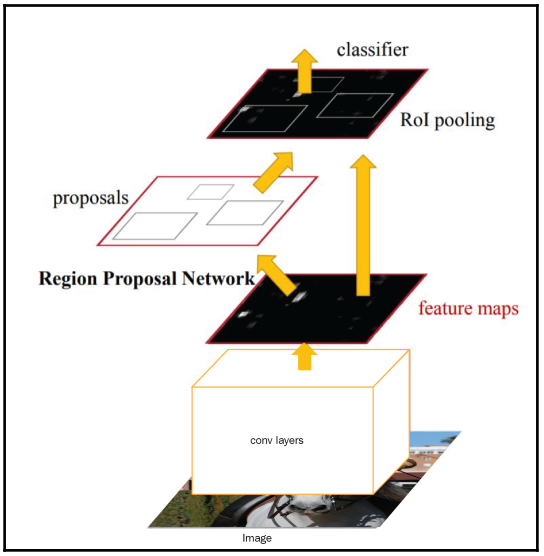
The result achieved with Fast R-CNN is great. What is even greater is the use of a powerful CNN network to provide very effective features for all three challenges that we need to overcome. But there are still some drawbacks, and there is scope for further improvements as we will understand in our next section on Faster R-CNN.

Faster R-CNN – faster region proposal network-based CNN

We saw in the earlier section that Fast R-CNN brought down the time required for scoring (testing) images drastically, but the reduction ignored the time required for generating Region Proposals, which use a separate mechanism (though pulling from the convolution map from CNN) and continue proving a bottleneck. Also, we observed that though all three challenges were resolved using the common features from convolution-map in Fast R-CNN, they were using different mechanisms/models.

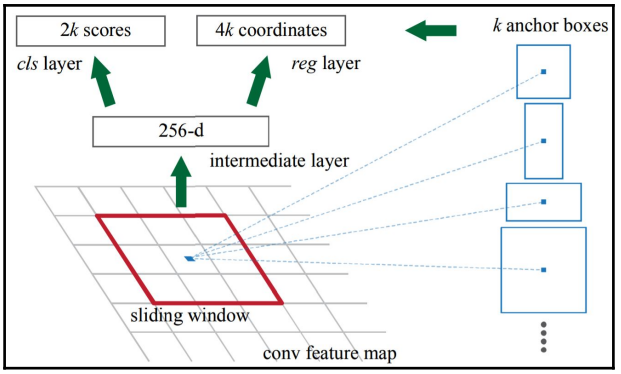
Faster R-CNN improves upon these drawbacks and proposes the concept of **Region Proposal Networks** (**RPNs**), bringing down the scoring (testing) time to 0.2 seconds per image, even including time for Region Proposals.

Fast R-CNN was doing the scoring (testing) in 0.3 seconds per image, that too excluding the time required for the process equivalent to Region Proposal.



Faster R-CNN: Working - The Region Proposal Networking acting as Attention Mechanism

As shown in the earlier figure, a VGG16 (or another) CNN works directly on the image, producing a convolutional map (similar to what was done in Fast R-CNN). Things differ from here, where now there are two branches, one feeding into the RPN and the other into the detection Network. This is again an extension of the same CNN for prediction, leading to a **Fully Convolutional Network** (**FCN**). The RPN acts as an Attention Mechanism and also shares full-image convolutional features with the detection network. Also, now because all the parts in the network can use efficient GPU-based computation, it thus reduces the overall time required:

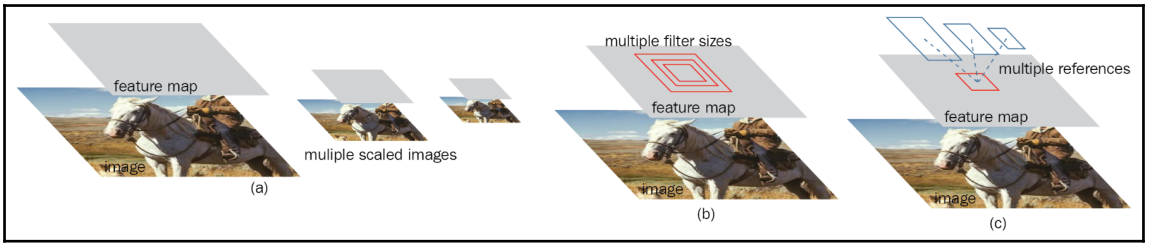


Faster R-CNN: Working - The Region Proposal Networking acting as Attention Mechanism

For a greater understanding of the Attention Mechanism, refer to the chapter on Attention Mechanisms for CNN in this book.

The RPN works in a sliding window mechanism, where a window slides (much like CNN filters) across the last convolution map from the shared convolutional layer. With each slide, the sliding window produces *k* ($k=N_{Scale} \times N_{Size}$) number of Anchor Boxes (similar to Candidate Boxes), where N_{Scale} is the number of (pyramid like) scales per *size* of the N_{Size} sized (aspect ratio) box extracted from the center of the sliding window, much like the following figure.

The RPN leads into a flattened, FC layer. This, in turn, leads into two networks, one for predicting the four numbers for each of the *k* boxes (determining the coordinates, length and width of the box as in Fast R-CNN), and another into a binomial classification model that determines the objectness or probability of finding any of the given objects in that box. The output from the RPN leads into the detection network, which detects which particular class of object is in each of the k boxes given the position of the box and its objectness.



Faster R-CNN: Working - extracting different scales and sizes

One problem in this architecture is the training of the two networks, namely the Region Proposal and detection network. We learned that CNN is trained using backpropagating across all layers while reducing the losses layers with every iteration. But because of the split into two different networks, we could at a time backpropagate across only one network. To resolve this issue, the training is done iteratively across each network, while keeping the weights of the other network constant. This helps in converging both the networks quickly.

An important feature of the RPN architecture is that it has translation invariance with respect to both the functions, one that is producing the anchors, and another that is producing the attributes (its coordinate and objectness) for the anchors. Because of translation invariance, a reverse operation, or producing the portion of the image given a vector map of an anchor map is feasible.

Owing to Translational Invariance, we can move in either direction in a CNN, that is from image to (region) proposals, and from the proposals to the corresponding portion of the image.

Mask R-CNN – Instance segmentation with CNN

Faster R-CNN is state-of-the-art stuff in object detection today. But there are problems overlapping the area of object detection that Faster R-CNN cannot solve effectively, which is where Mask R-CNN, an evolution of Faster R-CNN can help.

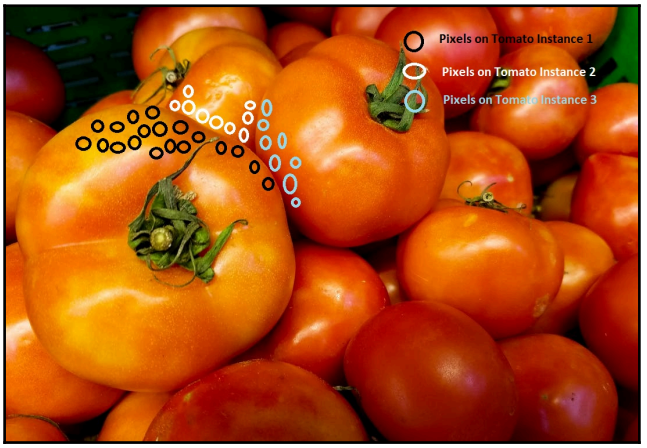
This section introduces the concept of instance segmentation, which is a combination of the standard object detection problem as described in this chapter, and the challenge of semantic segmentation.

In semantic segmentation, as applied to images, the goal is to classify each pixel into a fixed set of categories without differentiating object instances.

Remember our example of counting the number of dogs in the image in the intuition section? We were able to count the number of dogs easily, because they were very much apart, with no overlap, so essentially just counting the number of objects did the job. Now, take the following image, for instance, and count the number of tomatoes using object detection. It will be a daunting task because the Bounding Boxes will have so much of an overlap that it will be difficult to distinguish the Instances of tomatoes from the boxes.

So, essentially, we need to go further, beyond bounding boxes and into pixels to get that level separation and identification. Like we use to classify bounding boxes with object names in object detection, in Instance Segment, we segment/ classify, each pixel with not only the specific object name but also the object-instance.

The object detection and Instance Segmentation could be treated as two different tasks, one logically leading to another, much like we discovered the tasks of finding Region Proposals and Classification in the case of object detection. But as in the case of object detection, and especially with techniques like Fast/Faster R-CNN, we discovered that it would be much effective if we have a mechanism to do them simultaneously, while also utilizing much of the computation and network to do so, to make the tasks seamless.



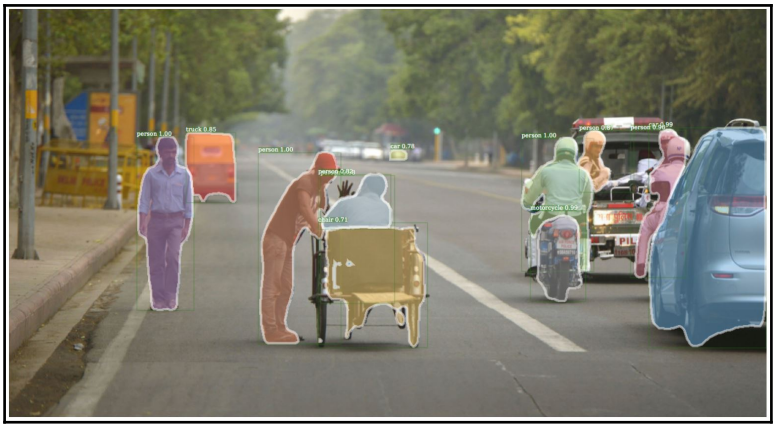
Instance Segmentation – Intuition

Mask R-CNN is an extension of Faster R-CNN covered in the earlier network, and uses all the techniques used in Faster R-CNN, with one addition—an additional path in the network to generate a Segmentation Mask (or Object Mask) for each detected Object Instance in parallel. Also, because of this approach of using most of the existing network, it adds only a minimal overhead to the entire processing and has a scoring (test) time almost equivalent to that of Faster R-CNN. It has one of the best accuracies across all single-model solutions as applied to the COCO2016 challenge (using the COCO2015 dataset).

Like, PASCAL VOC, COCO is another large-scale standard (series of) dataset (from Microsoft). Besides object detection, COCO is also used for segmentation and captioning. COCO is more extensive than many other datasets and much of the recent comparison on object detection is done on this for comparison purposes. The COCO dataset comes in three variants, namely COCO 2014, COCO 2015, and COCO 2017.

In Mask R-CNN, besides having the two branches that generate the objectness and localization for each anchor box or RoI, there also exists a third FCN that takes in the RoI and predicts a segmentation mask in a pixel-to-pixel manner for the given anchor box.

But there still remain some challenges. Though Faster R-CNN does demonstrate transformational invariance (that is, we could trace from the convolutional map of the RPN to the pixel map of the actual image), the convolutional map has a different structure from that of the actual image pixels. So, there is no pixel-to-pixel alignment between network inputs and outputs, which is important for our purpose of providing pixel-to-pixel masking using this network. To solve this challenge, Mask R-CNN uses a quantization-free layer (named RoIAlign in the original paper) that helps align the exact spatial locations. This layer not only provides exact alignment but also helps in improving the accuracy to a great extent, because of which Mask R-CNN is able to outperform many other networks:



Mask R-CNN – Instance Segmentation Mask (illustrative output)

The concept of instance segmentation is very powerful and can lead to realizing a lot of very impactful use cases that were not possible with object detection alone.

We can even use instance segmentation to estimate human poses in the same framework and eliminate them.

Instance segmentation in code

It's now time to put the things that we've learned into practice. We'll use the COCO dataset and its API for the data, and use Facebook Research's Detectron project (link in References), which provides the Python implementation of many of the previously discussed techniques under an Apache 2.0 license. The code works with Python2 and Caffe2, so we'll need a virtual environment with the given configuration.

Creating the environment

The virtual environment, with Caffe2 installation, can be created as per the `caffe2` installation instructions on the Caffe2 repository link in the *References* Section. Next, we will install the dependencies.

Installing Python dependencies (Python2 environment)

We can install the Python dependencies as shown in the following code block:

Python 2X and Python 3X are two different flavors of Python (or more precisely CPython), and not a conventional upgrade of version, therefore the libraries for one variant might not be compatible with another. Use Python 2X for this section.

When we refer to the (interpreted) programming language Python, we need to refer to it with the specific interpreter (since it is an interpreted language as opposed to a compiled one like Java). The interpreter that we implicitly refer to as the Python interpreter (like the one you download from Python.org or the one that comes bundled with Anaconda) is technically called CPython, on which is the default byte-code interpreter of Python, which is written in C. But there are other Python interpreters also like Jython (build on Java), PyPy (written in Python itself - not so intuitive, right?), IronPython (.NET implementation of Python).

```
pip install numpy>=1.13 pyyaml>=3.12 matplotlib opencv-python>=3.2
setuptools Cython mock scipy
```

Downloading and installing the COCO API and detectron library (OS shell commands)

We will then download and install the Python dependencies as shown in the following code block:

```
# COCO API download and install
# COCOAPI=/path/to/clone/cocoapi
git clone https://github.com/cocodataset/cocoapi.git $COCOAPI
cd $COCOAPI/PythonAPI
make install

# Detectron library download and install
# DETECTRON=/path/to/clone/detectron
git clone https://github.com/facebookresearch/detectron $DETECTRON
cd $DETECTRON/lib && make
```

Alternatively, we can download and use the Docker image of the environment (requires Nvidia GPU support):

```
# DOCKER image build
cd $DETECTRON/docker docker build -t detectron:c2-cuda9-cudnn7.
nvidia-docker run --rm -it detectron:c2-cuda9-cudnn7 python2
tests/test_batch_permutation_op.py
```

Preparing the COCO dataset folder structure

Now we will see the code to prepare the COCO dataset folder structure as follows:

```
# We need the following Folder structure: coco [coco_train2014,
coco_val2014, annotations]
mkdir -p $DETECTRON/lib/datasets/data/coco
ln -s /path/to/coco_train2014 $DETECTRON/lib/datasets/data/coco/
ln -s /path/to/coco_val2014 $DETECTRON/lib/datasets/data/coco/
ln -s /path/to/json/annotations
$DETECTRON/lib/datasets/data/coco/annotations
```

Running the pre-trained model on the COCO dataset

We can now implement the pre-trained model on the COCO dataset as shown in the following code snippet:

```
python2 tools/test_net.py \
    --cfg configs/12_2017_baselines/e2e_mask_rcnn_R-101-FPN_2x.yaml \
    TEST.WEIGHTS
https://s3-us-west-2.amazonaws.com/detectron/35861858/12_2017_baselines/e2e
_mask_rcnn_R-101-
FPN_2x.yaml.02_32_51.SgT4y1cO/output/train/coco_2014_train:coco_2014_valmin
usminival/generalized_rcnn/model_final.pkl \
    NUM_GPUS 1
```

References

1. Paul Viola and Michael Jones, Rapid object detection using a boosted cascade of simple features, *Conference on Computer Vision and Pattern Recognition*, 2001.

2. Paul Viola and Michael Jones, Robust Real-time object detection, *International Journal of Computer Vision*, 2001.

3. Itseez2015opencv, OpenCV, *Open Source Computer Vision Library*, Itseez, 2015.

4. Ross B. Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, CoRR, arXiv:1311.2524, 2013.

5. Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, Computer Vision and Pattern Recognition, 2014.

6. M. Everingham, L. VanGool, C. K. I. Williams, J. Winn, A. Zisserman, *The PASCAL Visual Object Classes Challenge 2012*, VOC2012, Results.

7. D. Lowe. *Distinctive image features from scale-invariant keypoints*, IJCV, 2004.

8. N. Dalal and B. Triggs. *Histograms of oriented gradients for human detection*. In CVPR, 2005.

9. Ross B. Girshick, Fast R-CNN, CoRR, arXiv:1504.08083, 2015.

10. Rbgirshick, fast-rcnn, GitHub, `https://github.com/rbgirshick/fast-rcnn`, Feb-2018.

11. Shaoqing Ren, Kaiming He, Ross B. Girshick, Jian Sun, Faster R-CNN: *Towards Real-Time Object Detection with Region Proposal Networks*, CoRR, arXiv:1506.01497, 2015.

12. Shaoqing Ren and Kaiming He and Ross Girshick and Jian Sun, Faster R-CNN: *Towards Real-Time Object Detection with Region Proposal Networks*, Advances in **Neural Information Processing Systems** (**NIPS**), 2015.

13. Rbgirshick, py-faster-rcnn, GitHub, `https://github.com/rbgirshick/py-faster-rcnn`, Feb-2018.

14. Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollar, Kaiming He,

    Detectron, GitHub, `https://github.com/facebookresearch/Detectron`, Feb-2018.

15. Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollar, C. Lawrence Zitnick, *Microsoft COCO: Common Objects in Context*, CoRR, arXiv:1405.0312, 2014.

16. Kaiming He, Georgia Gkioxari, Piotr Dollar, Ross B. Girshick, Mask R-CNN, CoRR, arXiv:1703.06870, 2017.

17. Liang-Chieh Chen, Alexander Hermans, George Papandreou, Florian Schroff, Peng Wang, Hartwig Adam, MaskLab: *Instance Segmentation by Refining Object Detection with Semantic and Direction Features*, CoRR, arXiv:1712.04837, 2017.

18. Anurag Arnab, Philip H. S. Torr, *Pixelwise Instance Segmentation with a Dynamically Instantiated Network*, CoRR, arXiv:1704.02386, 2017.

19. Matterport, Mask_RCNN, GitHub, `https://github.com/matterport/Mask_RCNN`, Feb-2018.

20. CharlesShang, FastMaskRCNN, GitHub, `https://github.com/CharlesShang/FastMaskRCNN`, Feb-2018.

21. Caffe2, Caffe2, GitHub, `https://github.com/caffe2/caffe2`, Feb-2018.

Summary

In this chapter, we started from the very simple intuition behind the task of object detection and then proceeded to very advanced concepts, such as Instance Segmentation, which is a contemporary research area. Object detection is at the heart of a lot of innovation in the field of Retail, Media, Social Media, Mobility, and Security; there is a lot of potential for using these technologies to create very impactful and profitable features for both enterprise and social consumption.

From the Algorithms perspective, this chapter started with the legendary Viola-Jones algorithm and its underlying mechanisms, such as Haar Features and Cascading Classifiers. Using that intuition, we started exploring the world of CNN for object detection with algorithms, such as R-CNN, Fast R-CNN, up to the very state-of-the-art Faster R-CNN.

In this chapter, we also laid the foundations and introduced a very recent and impactful field of research called **instance segmentation**. We also covered some state-of-the-art Deep CNNs based on methods, such as Mask R-CNN, for easy and performant implementation of instance segmentation.