# Using FCNN Receptive Fields for Object Detection

The task of **Object Detection** is to detect and spatially identify (using bounding boxes etc.) various objects in an image, whereas **Image Classification** tells whether or not an image contains certain objects without any notion of where exactly they are located.

In this post I describe an approach to create an **Object Detection** model from a pretrained **Image Classification** model in PyTorch.

Even though the Object Detection and Image Classification are two distinct categories of Computer Vision and their methods of training are fundamentally different, we will see in this post that there is some overlap in their functionality which we will exploit for our purpose.

The idea and the implementation presented in this post are built using concepts of **CNN Receptive Fields**, **Backpropagation** and **Fully Convolutional Networks**.

In case you are unsure about some of the above terms, I recommend you to go through the following blogs on **LearnOpenCV.com** whichintroduce these concepts very well.

* [Fully Convolutional Image Classification on Arbitrary Sized Image](https://www.learnopencv.com/fully-convolutional-image-classification-on-arbitrary-sized-image/)
* [CNN Receptive Field Computation Using Backprop](https://www.learnopencv.com/cnn-receptive-field-computation-using-backprop/)

In fact, even if you are already quite familiar with these terms, I would still urge you to go through the above posts since the implementation that I am presenting extends upon the above references and for the sake of brevity, I have not captured details where a better explanation is already available.

Although you may find that the results using this approach do not quite match the performance of popular object detectors like **YOLO**, nevertheless, the purpose of this exercise is to describe an approach where a pretrained Image Classifier can be used to create an Object Detector without any explicit training on annotated bounding boxes.

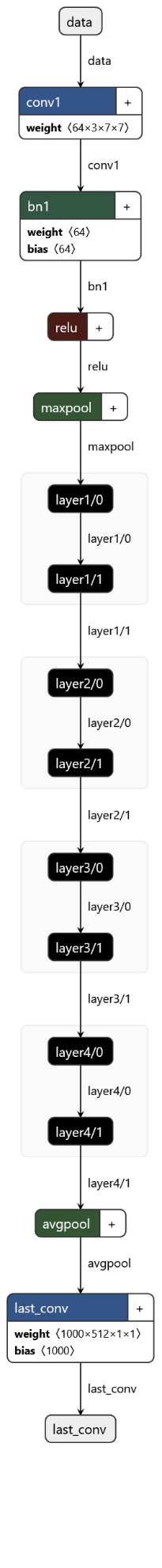
Having set the expectations, let’s get started.

## Bit of Background

For this exercise, I have used a slightly modified version of the pretrained **Resnet-18** model provided in PyTorch.

Specifically, the model used is a variant of Resnet-18, in which the final (and only) Fully Connected (or Linear) layer of the model is replaced by a 2D Convolution layer, thus converting the model into a Fully Convolutional Neural Network (FCNN).

The **Fully Convolutional Resnet18** model architecture looks like this (visualization using [Netron](https://lutzroeder.github.io/netron/)):

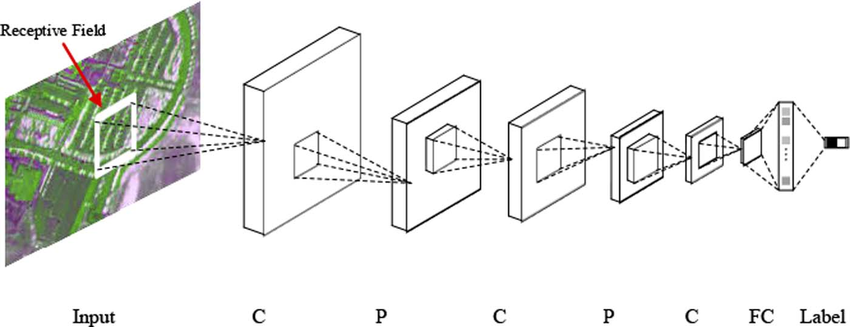


As a quick recap of the concepts presented in the blogs mentioned previously, an FCNN takes an image of arbitrary size as input and gives as output an activation/response map containing the model’s prediction of objects detected in the image.

Using the Backpropagation algorithm, we compute the Receptive Field of this response map and find out the location of the detected objects in the original image.

As you may know by now, Receptive Field of a pixel in a feature map (or layer) in a Neural Network represents all the pixels from the previous feature maps that affected its value. It is a useful debugging tool to understand what the network really saw in the input image when it gave a certain prediction.

The following figure illustrates this concept (image referenced from [this paper](https://www.researchgate.net/figure/Conventional-CNN-framework-interspersed-with-convolutional-layers-and-max-pooling-layers_fig3_300081672))



Whereas in the previous posts, we focused only on the result category with maximum score/probability (we just wanted an Image Classifier after all), this time we are going deeper to extract all of the model’s predictions and compute each of the respective receptive fields which would give us our bounding boxes.

## Let’s Get Started

Our procedure starts after we have obtained the response map predicted by the FCNN Resnet-18 model on the input image using the steps captured in [this blog](https://www.learnopencv.com/cnn-receptive-field-computation-using-backprop/) which I referenced at the beginning of this post. I am not repeating the steps in this post as that would be duplicate effort and needlessly make this post a lot lengthier.

As you will recall, the response map from the FCNN is of shape [1 x 1000 x n x m] where n and m depend on the width and height of the input image and the network itself. The 1000 corresponds to the 1000 classes of ImageNet database on which the Resnet18 is trained.

Using a max operation on this response map, we get the top [n x m] predictions or **score map** from the model which can be interpreted as the inference performed on [n x m] locations on the image by obtained sliding window of size 224×224 (input image size for the original network).

For each of these predictions in the **score map**, we have the probabilities (given by Softmax layer) and the category (from ImageNet).

Following is an example score map of shape [3 x 8] with top prediction highlighted:

Probabilities –

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0.0511 | 0.0384 | 0.0367 | 0.0304 | 0.0279 | 0.0307 | 0.0296 | 0.0194 |
| 0.5009 | 0.165 | 0.1246 | 0.1362 | 0.1477 | 0.0673 | 0.8685 | 0.3385 |
| 0.0863 | 0.1204 | 0.0786 | 0.2635 | 0.1026 | 0.1137 | 0.2548 | 0.7218 |

Categories –

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 977 | 977 | 977 | 977 | 977 | 978 | 354 | 437 |
| 978 | 977 | 980 | 977 | 858 | 970 | 354 | 461 |
| 977 | 978 | 977 | 977 | 977 | 977 | 354 | 354 |

We will now go through each of these [n x m] predictions and, if its probability is above a predefined threshold, calculate its receptive field using backpropagation.

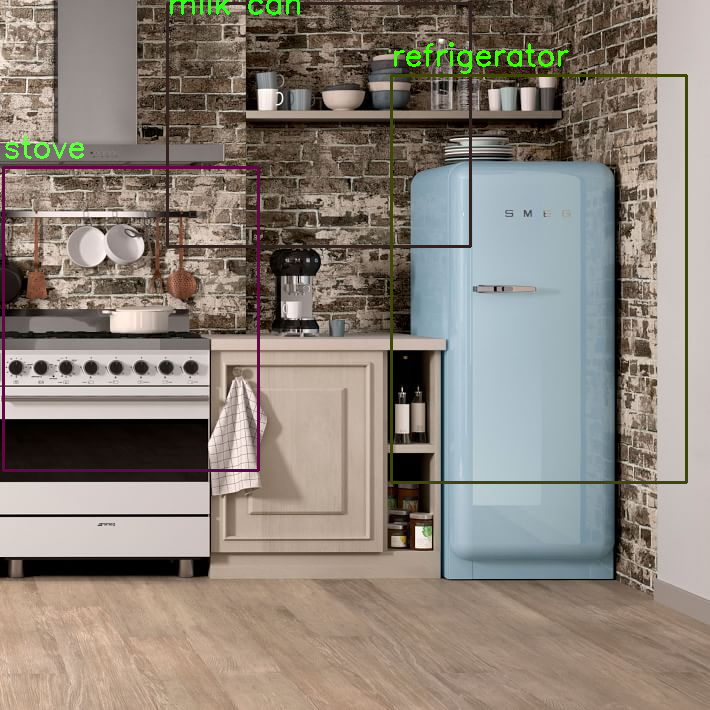
On each of these receptive fields, we will then apply OpenCV Image Threshold and Contour operations to obtain the bounding boxes.

The entire pipeline is summarized as illustrated below.

Following is a snippet of the implementation. Download the full code [from here](https://github.com/DebalB/Python_public/tree/master/fcnn_object_detector_pytorch).

<script src="https://gist.github.com/DebalB/d3895a02d8d6dbf93709493ea33d7b77.js"></script>

Following is a sample of the results that we achieve.



## Handling Overlapping Bounding Boxes

An important thing to note is that the network might detect the same object multiple times (with varying probabilities). This leads to the problem of overlapped bounding boxes which is typical even in standard object detection algorithms.

In order to fix this problem, we use Non-Maxima Suppression method. I have used the function *object\_detection.non\_max\_suppression()* provided by the **imutils** package. However, any other method may be used which achieves the same purpose.

## Receptive Field Computation Options

Now here is an interesting design choice that I would like to talk about.

For any detected object category, its corresponding receptive field can be computed using either of the following:

* Max Activated Pixel
* Net Prediction

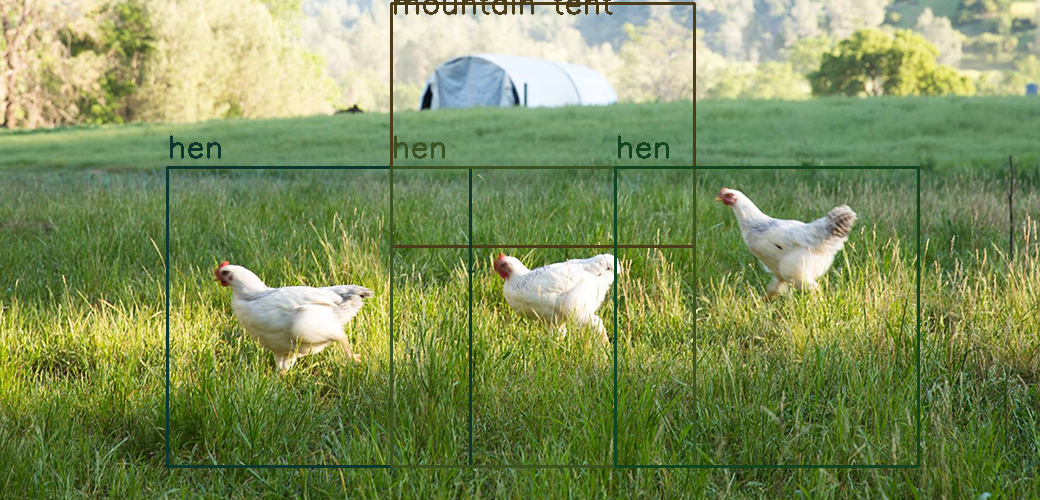
Needless to say, the resulting bounding boxes will vary a lot depending on the choice that we make above.

Again, the corresponding details are well explained [this blog](https://www.learnopencv.com/cnn-receptive-field-computation-using-backprop/).

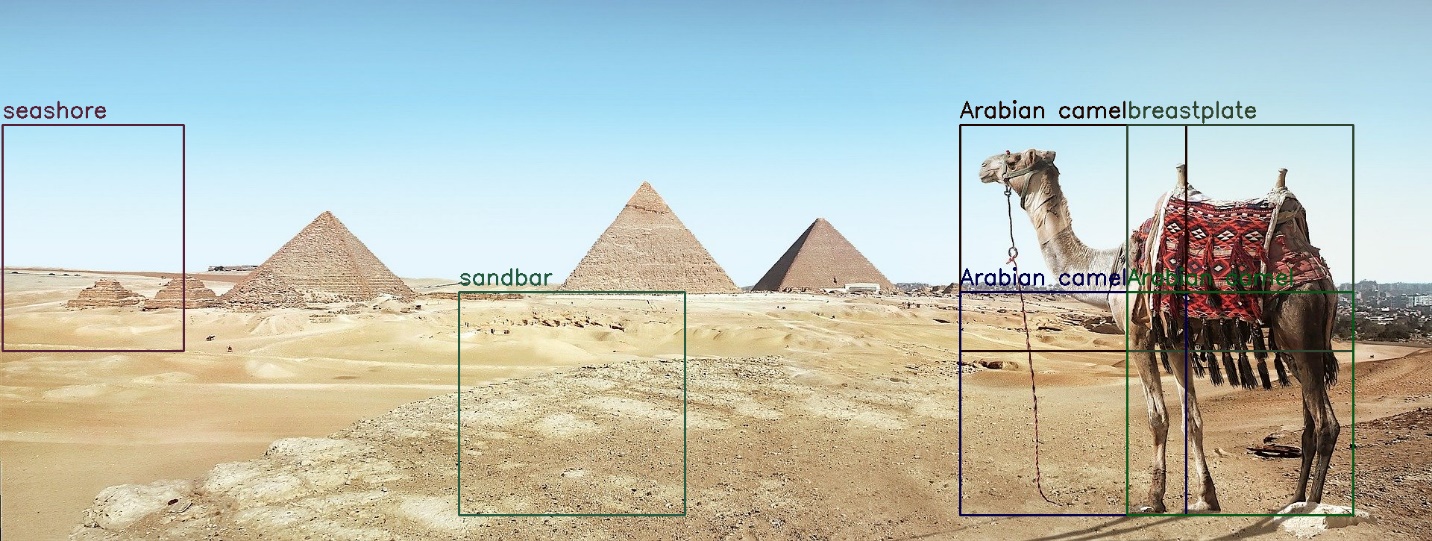
However, I will summarize them briefly and give a comparison of the two methods with respect to the results that they produce.

**Option-1:** Receptive Field computed for the Max Activated pixel of a category, looks at only one pixel in the response map and provides the bounding box of the region in the original image which maximized that pixel.

On some images (e.g. low-resolution images), this option nicely segregates each individual instance of a category, as illustrated below.



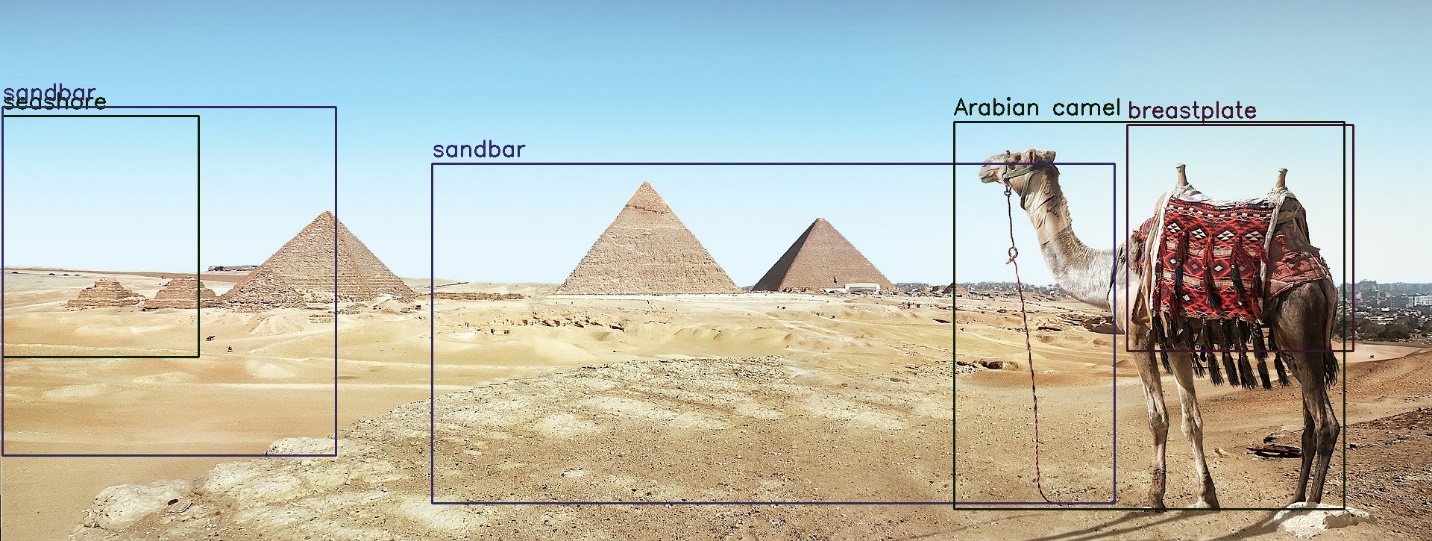
However, in certain images (e.g. high-resolution images), this may return a bounding box that encloses only a part of the object rather than the object in entirety, e.g. see the image below.



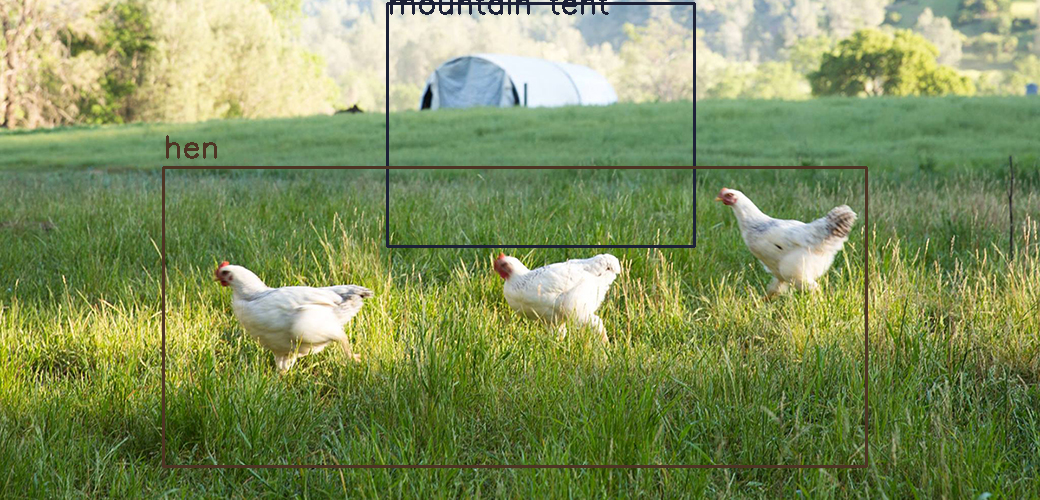
**Option-2:** Receptive Field computed for the Net Prediction, takes into account each of the [n x m] pixels in the corresponding response map of a category and provides the bounding box of the region in the original image which maximized the net prediction of that category.

As before, the results of this approach vary with the image being used.

For some images it returns bounding box neatly enclosing the entire object as illustrated below.



On some images however, the obtained bounding box may enclose together several instances of a category as shown below.



## What’s the verdict?

As you can see, there is no clear verdict as to which of the two options yields a better result.

I recommend readers to experiment further using their own image datasets, and get a broader idea as to what does and does not work for them.

As I initially mentioned, the objective of this post was not to present a robust method for object detection, but rather to explore how a specific model can be used to serve an alternate purpose.

It is not uncommon in machine learning that creators of a model discover that their creation performs well even at tasks it was never trained to do.

With that thought, I would like to end this blog.

I hope that you have enjoyed it and I would be delighted to hear any further ideas and findings that you may come across.

Download the full code [from here](https://github.com/DebalB/Python_public/tree/master/fcnn_object_detector_pytorch).