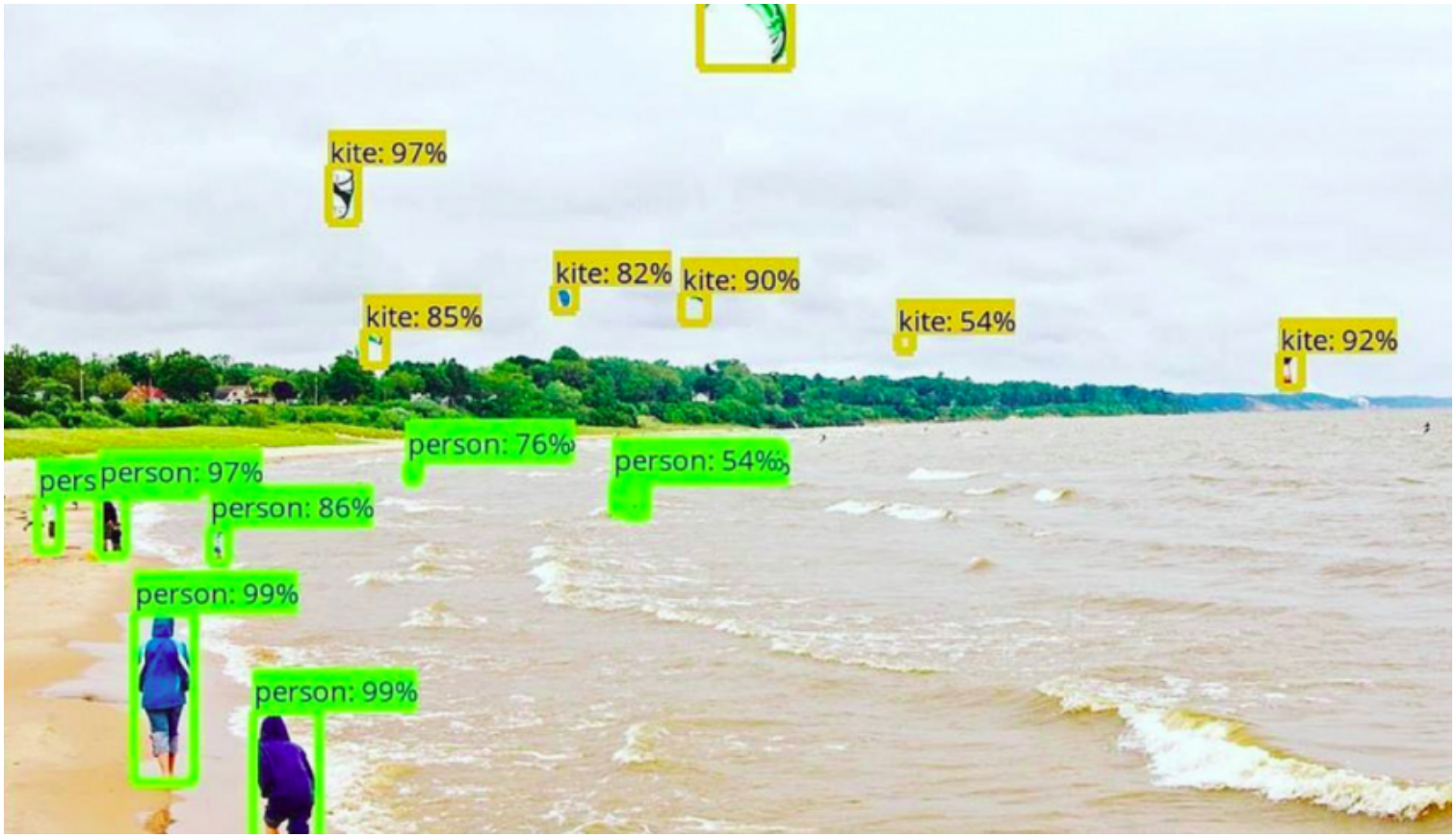
Understanding and Building an Object Detection Model from Scratch in Python

COMPUTER VISION | DEEP LEARNING | IMAGE | OBJECT DETECTION | PROJECT | PYTHON | SUPERVISED | TECHNIQUE | UNSTRUCTURED DATA

Introduction

When we're shown an image, our brain instantly recognizes the objects contained in it. On the other hand, it takes a lot of time and training data for a machine to identify these objects. But with the recent advances in hardware and deep learning, this computer vision field has become a whole lot easier and more intuitive.

Check out the below image as an example. The system is able to identify different objects in the image with incredible accuracy.



Object detection technology has seen a rapid adoption rate in various and diverse industries. It helps self-driving cars safely navigate through traffic, spots violent behavior in a crowded place, assists sports teams analyze and build scouting reports, ensures proper quality control of parts in manufacturing, among many, many other things. And these are just scratching the surface of what object detection technology can do!

In this article, we will understand what object detection is and look at a few different approaches one can take to solve problems in this space. Then we will deep dive into building our own object detection system in Python. By the end of the article, you will have enough knowledge to take on different object detection challenges on your own!

Note: This tutorial assumes that you know the basics of deep learning and have solved simple image processing problems before. In case you haven't, or need a refresher, I recommend reading the following articles first:

- **Fundamentals of Deep Learning – Starting with Artificial Neural Network**
- **Deep Learning for Computer Vision – Introduction to Convolution Neural Networks**
- **Tutorial: Optimizing Neural Networks using Keras (with Image recognition case study)**

Table of Contents

What is Object Detection?

Before we dive into build a state-of-the-art model, let us first try to understand what object detection is. Let's (hypothetically) build a pedestrian detection system for a self-driving car. Suppose your car captures an image like the one below. How would you describe this image?

The image essentially depicts that our car is near a square, and a handful of people are crossing the road in front of our car. As the traffic sign is not clearly visible, the car's pedestrian detection system should identify exactly where the people are walking so that we can steer clear of them.

So what can the car's system do to ensure this happens? What it can do is create a bounding box around these people, so that the system can pinpoint where in the image the people are, and then accordingly make a decision as to which path to take, in order to avoid any mishaps.

Our objective behind doing object detection is two folds:

1. To identify what all objects are present in the image and where they're located
2. Filter out the object of attention

Different Approaches to Solve an Object Detection Problem

Now that we know what our problem statement is, what can be a possible approach (or multiple approaches) to solve it? In this section, we'll look at a few techniques that can be used to detect objects in images. We will start from the simplest approach and find our way up from there. If you have any suggestions or alternate approaches to the ones we will see below, do let me know in the comments section!

Approach 1: Naive way (Divide and Conquer)

The simplest approach we can take is to divide the image into four parts:

- Upper left hand side corner

- Upper right hand side corner

- Lower left hand side corner

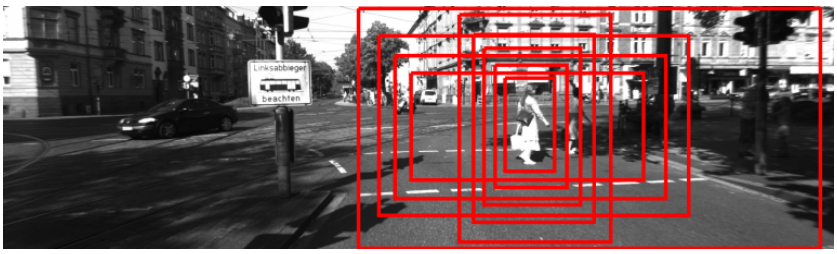
- Lower right hand side corner

Now the next step is to feed each of these parts into an image classifier. This will give us an output of whether that part of the image has a pedestrian or not. If yes, mark that patch in the original image. The output will be somewhat like this:

This is a good approach to try out first, but we are looking for a much more accurate and precise system. It needs to identify the entire object (or a person in this case) because only locating parts of an object could lead to catastrophic results.

Approach 2: Increase the number of divisions

The previous system worked well but what else can we do? We can improve upon it by exponentially increasing the number of patches we input into the system. This is how our output should look like:

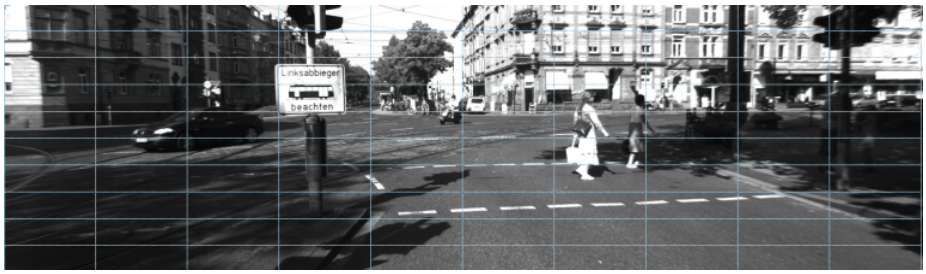


This ended up being a boon and a curse. Of course our solution seems a bit better than the naive approach, but it is riddled with so many bounding boxes which approximate the same thing. This is an issue, and we need a more structured way to solve our problem.

Approach 3: Performing structured divisions

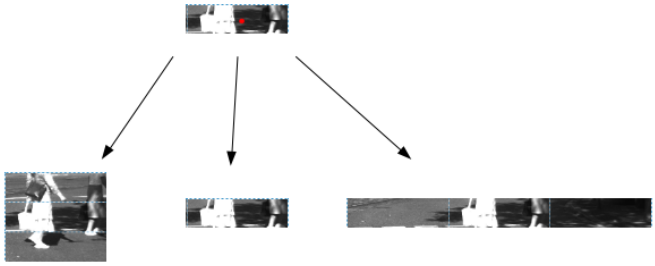
In order to build our object detection system in a more structured way, we can follow the below steps:

Step 1: Divide the image into a 10×10 grid like this:



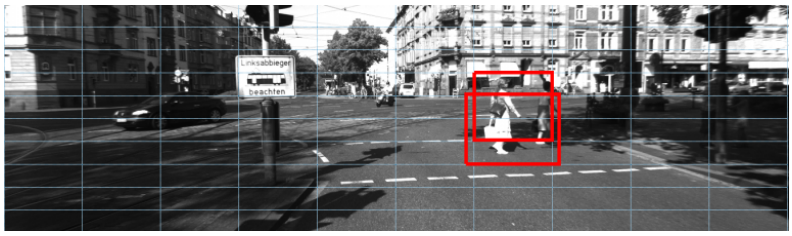
Step 2: Define the centroids for each patch

Step 3: For each centroid, take three different patches of different heights and aspect ratio:



Step 4: Pass all of the patches created through the image classifier to get predictions

So how does the final output look like? A bit more structured and disciplined for sure – take a look below:

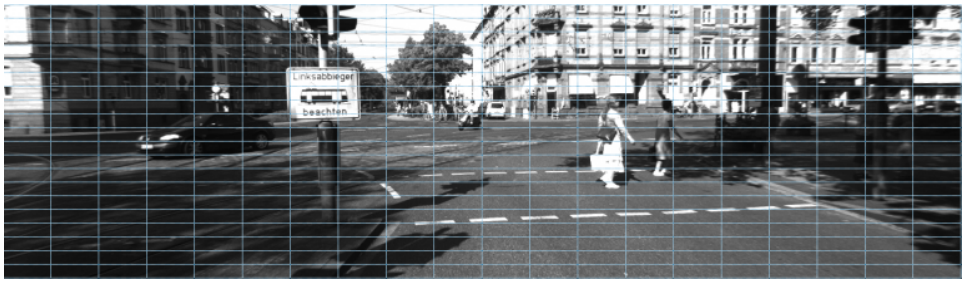


But we can further improve on this! Read on to see yet another approach that will produce even better results.

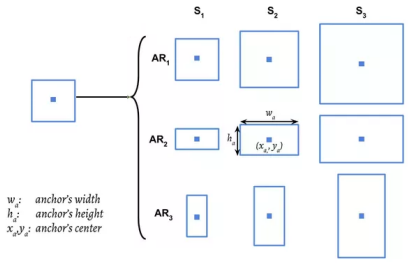
Approach 4: Becoming more efficient

The previous approach we saw is acceptable to quite a good degree, but we can build a system a little more efficient than that. Can you suggest how? Off the top of my mind, I can propose an optimization. If we think about approach #3, we can do two things to make our model better.

1. **Increase the grid size:** So instead of taking the grid size as 10, we can increase it to, say, 20:



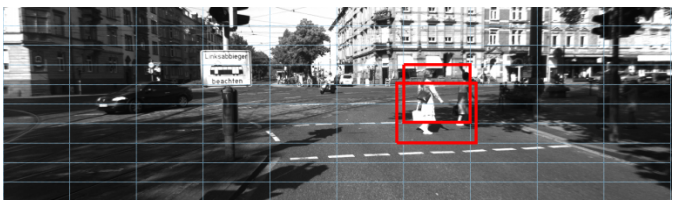
2. **Instead of three patches, take more patches with various heights and aspect ratios:** Here, we can take 9 shapes off of a single anchor, namely three square patches of different heights and 6 vertical and horizontal rectangle patches of different heights. This will provide us with different aspect ratios of the patches.



This again, has its pros and cons. Sure both of the methods will help us go to a more granular level. But it will again create an explosion of all the patches that we have to pass through our image classification model.

What we can do is, **take selective patches instead of taking all of them**. For example, we could build an intermediate classifier which tries to predict if the patch actually has background, or potentially contains an object. This would exponentially decrease the patches that our image classification model has to see.

One more optimization that we can do, is to decrease the predictions which say the “same thing”. Let’s take the output of approach 3 again:



As you can see, both the bounding box predictions are basically of the same person. We have an option to choose any one of them. So to make predictions, we consider all the boxes which “say the same thing” and then pick whichever one has the most probability of detecting a person.

All of these optimizations have so far given us pretty decent predictions. We almost have all the cards in our hands, but can you guess what is missing? Deep Learning of course!

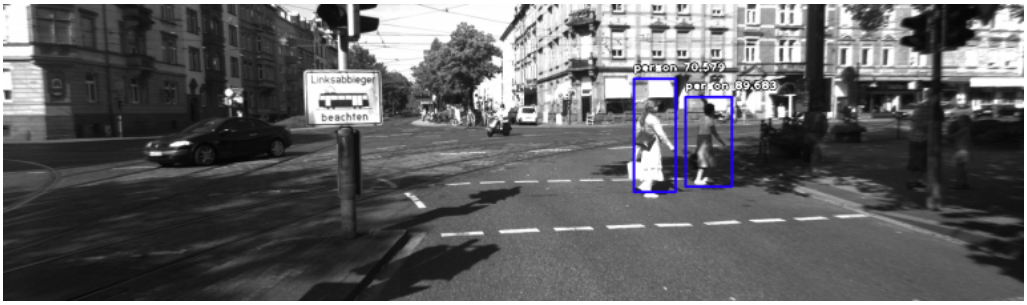
Approach 5: Using Deep Learning for feature selection and to build an end-to-end approach

Deep learning has so much potential in the object detection space. Can you recommend where and how can we leverage it for our problem? I have listed a couple of methodologies below:

- Instead of taking patches from the original image, we can pass the original image through a neural network to **reduce the dimensions**
- We could also use a neural network to **suggest selective patches**
- We can **reinforce** a deep learning algorithm **to give predictions** as **close to the original bounding box** as possible. This will ensure that the algorithm gives more tighter and finer bounding box predictions

Now instead of training different neural networks for solving each individual problem, we can take a single deep neural network model which will attempt to solve all the problems by itself. The advantage of doing this, is that each of the smaller components of a neural network will help in optimizing the other parts of the same neural network. This will help us in jointly training the entire deep model.

Our output would give us the best performance out of all the approaches we have seen so far, somewhat similar to the image below. We will see how to create this using Python in the next section.



Getting Technical: How to build an Object Detection model using the ImageAI library

Now that we know what object detection is and the best approach to solve the problem, let's build our own object detection system! We will be using ImageAI, a python library which supports state-of-the-art machine learning algorithms for computer vision tasks.

Running an object detection model to get predictions is fairly simple. We don't have to go through complex installation scripts to get started. We don't even need a GPU to generate predictions! We will use this ImageAI library to get the output prediction we saw above in approach #5. I highly recommend following along with the code below (on your own machine) as this will enable you to gain the maximum knowledge out of this section.

Please note that you need to set up your system before creating the object detection model. Once you have Anaconda installed in your local system, you can get started with the below steps.

Step 1: Create an Anaconda environment with python version 3.6.

```
conda create -n retinanet python=3.6 anaconda
```

Step 2: Activate the environment and install the necessary packages.

```
source activate retinanet conda install tensorflow numpy scipy opencv pillow matplotlib h5py keras
```

Step 3: Then install the ImageAI library.

```
pip install https://github.com/OlafenwaMoses/ImageAI/releases/download/2.0.1/imageai-2.0.1-py3-none-any.whl
```

Step 4: Now download the pretrained model required to generate predictions. This model is based on RetinaNet (a subject of a future article). Click on the link to download – **RetinaNet Pretrained model**

Step 5: Copy the downloaded file to your current working folder

Step 6: Download the image from this link. Name the image as image.png

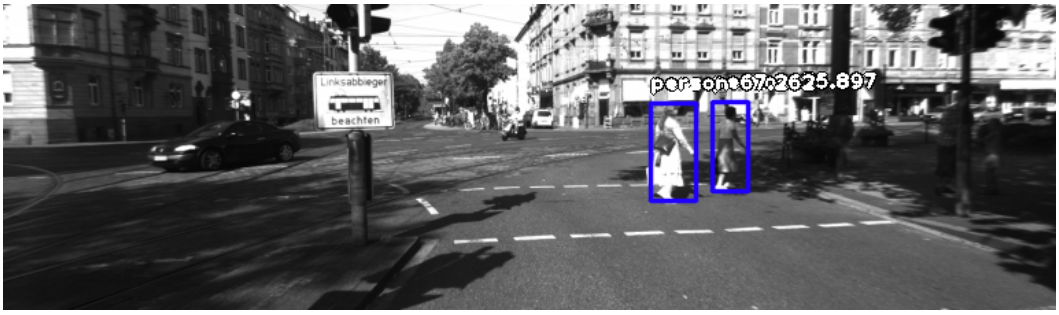
Step 7: Open jupyter notebook (type ***jupyter notebook*** in your terminal) and run the following codes:

```
from imageai.Detection import ObjectDetection import os execution_path = os.getcwd() detector = ObjectDetection() detector.setModelTypeAsRetinaNet() detector.setModelPath( os.path.join(execution_path , "resnet50_coco_best_v2.0.1.h5")) detector.loadModel() custom_objects = detector.CustomObjects(person=True, car=False) detections = detector.detectCustomObjectsFromImage(input_image=os.path.join(execution_path , "image.png"), output_image_path=os.path.join(execution_path , "image_new.png"), custom_objects=custom_objects, minimum_percentage_probability=65) for eachObject in detections: print(eachObject["name"] + " : " + eachObject["percentage_probability"] ) print("----------
----")
```

This will create a modified image file named image_new.png, which contains the bounding box for your image.

Step 8: To print the image use the following code:

```
from IPython.display import Image Image("image_new.png")
```



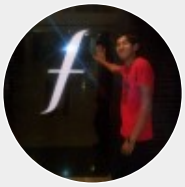
Congratulations! You have created your own object detection model for pedestrian detection. How awesome is that?

End Notes

In this article, we learned what is object detection, and the intuition behind creating an object detection model. We also saw how to build this object detection model for pedestrian detection using the ImageAI library.

By just tweaking the code a bit, you can easily transform the model to solve your own object detection challenges. If you do solve such a problem using the approach above, especially for a social cause, do let me know in the comments below!

Article Url - https://www.analyticsvidhya.com/blog/2018/06/understanding-building-object-detection-model-python/



Faizan Shaikh

Faizan is a Data Science enthusiast and a Deep learning rookie. A recent Comp. Sc. undergrad, he aims to utilize his skills to push the boundaries of AI research.