# Irregularity Detection with Box-Counting Method

Theoretical Aspects, Python Implementation and Results

Vlad-Stefan Tudor
Coord: Mircea Olteanu

2020

**Abstract**

The article will present the theoretical fundamentals of the Box-Counting Method and fractal dimensionality, will describe an implementation of the algorithm in Python and will discuss the results on multiple different test scenarios.

## I. Theoretical Aspects

### A. Fractal Dimension

The notion of *dimension* of a space is informally defined as the number of coordinates necessary for determining a certain point in that space. It is well known that the a point is of dimension 0, a line of dimension 1, a surface of dimension 2 and a cube is of dimension 3. However, these are relatively simple shapes, without any unintuitive properties.

An alternate and much more general definition can be formulated, based on the idea of *autosimilarity*. The idea is to see how many identical and autosimilar shapes $n$ times smaller than the original can be created. Let $p$ be this number. We can see that:

- we can divide a cubes in $p = n$ segments $n$ times smaller than the original line
- we can divide a cube in $p = n^2$ cubes $n$ times smaller than the original.

It's natural then to define the auto-similarity dimension as:

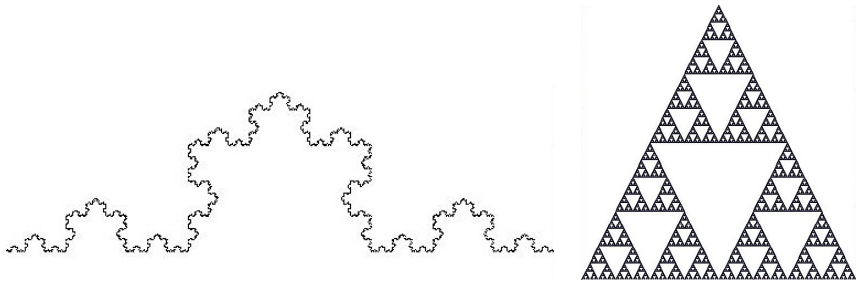
$$D_s = \frac{log(p)}{log(n)}$$

where: $p =$ number of resulted similar copies
$n =$ scaling factor, how small the copies are compared to the original

We see that this formula gives the same values for the regular shapes but allows for computing the dimensions of fractals as well.

Interestingly enough, both the Sierpinsky Triangle and the Koch Curve have a dimension somewhere between 1 and 2. We can interpret this in the following way:

- The Koch Curve, even though technically a line, is not one dimensional - due to it's infinite length it's halfway to being a surface ($D_s = 1.2619$)
- Even though a *hole* in a surface does not decrease its dimensionality, the infinite number of holes in the Sierpinsky Triangle account for it no longer being a true two dimensional shape ($D_s = 1.585$)



(a) Koch's Curve (b) Sierpinsky's Triangle

Fig. 1: Examples of Fractals

### B. Box Counting Method

Inspired from the one-dimensional Compass Method that was used to argue that the Great Britain has an infinitely-long coastline[**?**], a very useful empirical method of computing the fractal dimension is the Box Counting Method. It also builds upon the concept of auto-similarity dimension but, however, it not always produces the same result value.

The main steps of the algorithm are as follows:

1) divide the space of the shape into a grid of squares of size $s$ - the boxes.
2) count the numbers of squares that contain ever so slightly portions of the shape, noted as $N(s)$
3) repeat for smaller and smaller sized boxes - $s$ can decrease by powers of 2: $s = 2^0, 2^{-1}, 2^{-2}, ...$
4) plot $log(N(s))$ against $log(\frac{1}{s})$
5) compute the slope of the regression line fit between the points - this will be the Box-Counting Dimension $D_b$

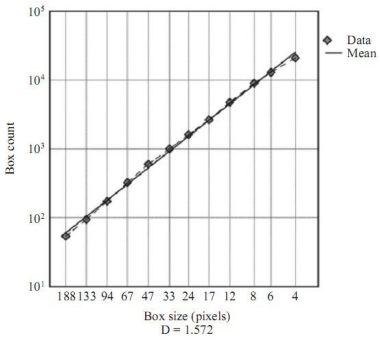



Fig. 2: Plotting the number of boxes against the size of the boxes [**?**]

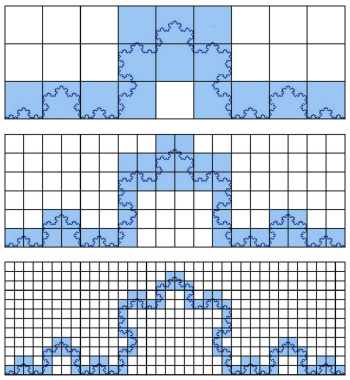


Fig. 3: Applying Box-Counting Method to Koch's Curve [**?**]

## II. Computing the Dimension Map

### A. Applying Box-Counting to a Photo

The algorithm can only handle matrices with boolean values, which, in practice, means black and white photos. However, any photo can be projected into this lower-dimensional representations with some simple operations:

1) gray-scaling - merges the 3 values of each cell of the matrix (RGB - red, green, blue) into a single scalar in the range of $[0, 1]$
2) threshold the normalised value with a meta-parameter, meaning that any value below the threshold remains a zero and everything else becomes a 1.

Both of these operations require meta-parameters in order to fine-tune the process so as to minimize the information loss regarding the specific objective. The result can then be fed into the Box-Counting Algorithm.

### B. Main Idea of Irregularity Detection

We have seen how the overall dimension of a regular photo can be computed with the Box-Counting method. This is a measure of its general complexity or how elaborate the patterns and the shapes are. This is, of course, not a constant value for every part of the photo but rather an average. This means that there might be parts of the photo with larger complexity than others, meaning that the *nature* of the shapes might differ.

Fig. 4: Boolean reduction of the Windows Wallpaper

For example, we would expect that the image of a city will have a different dimensionality than the one of field. This means that if we were to evaluate batches of a satellite image we could identify a city because the batches that contain it will yield a higher Box-Counting dimension. This can be generalised and a detection algorithm formulated as follows:

*In a given photo, objects that stand out from the background can be selected by computing the Box-Counting dimension of sub-parts of the photo and then extracting the regions with dimensions that don't match the dimensions of their surroundings. This kind of selection is not based on distinctions of color or previously modeled assumptions but only on the nature of the object and its defining texture and contour.*

### *C. Implementation in the Algorithm*

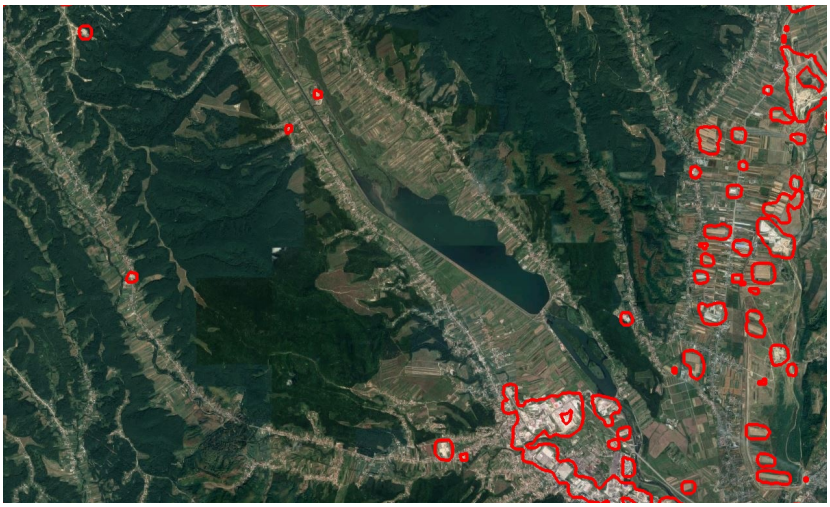
The application defines a method for compiling the box counting dimension. In the meta-parameters size of the batches is defined, alongside the step size between these batches. The threshold for the black and white transformation and a threshold for the tolerance level of the identified shapes need also to be specified.

The algorithm will compute a *dimension map* by evaluating the Box-counting dimension of each batch. We will be counting boxes that are neither full, nor empty, as they are the regions that define contours. The values will be then normalised. The mean value will be also determined.

An irregularity will be defined as a batch that has a specified dimension value deviation from the average dimension of the photo. All these batches will be memorised and then added on top of the original photo so as the user can easily see the output of the algorithm.

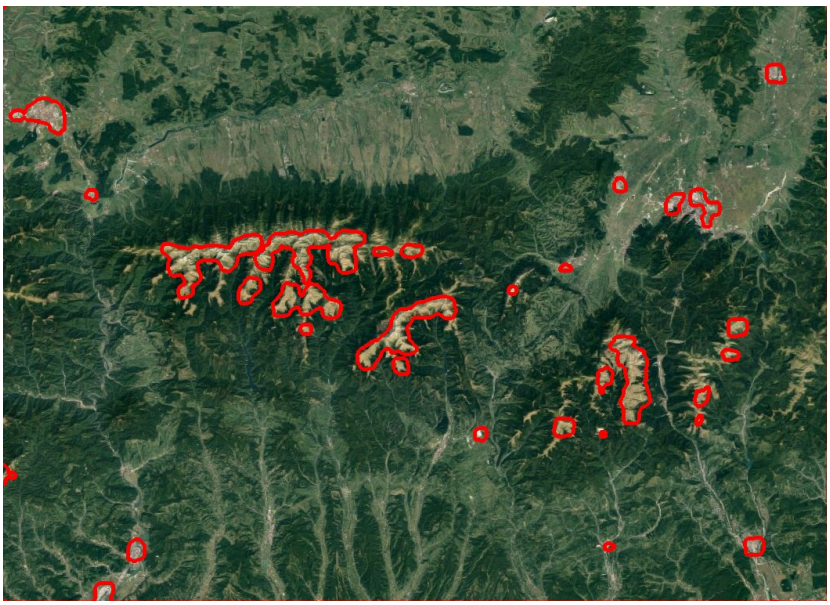
## III. Results - Irregularity Detection in Satellite Maps

1) **Small towns in the mountains**



*Note*: It's important to see that the algorithm identified most of the inhabited areas, both the towns and the few smaller complexes in the mountains, without selecting natural shapes such as the lake or the mountaintops.

2) **Carpathian Mountains**



*Note*:This satellite image is of a much wider zoom than the previous. This means that smaller cities are harder to be identified (since their complexity gets compressed in a relatively low number of pixels). We see that the algorithm selected several cities. An undesired output is the selection of the mountain tops but it was an expected phenomena since they have fractal-like shapes and, therefore, yield a high Box-Counting dimension score.

3) **Bucharest**

*Note*: A big city like Bucharest was expected to stand out from the homogeneous pattern of the surrounding fields. Indeed, the algorithm identified it, along with surrounding towns. Beside several false-positives, it's interesting to see that even though Bucharest is in itself a dimensionaly-complex shape, its complexity varies from neighbourhood to neighbourhood, thus the algorithm outputs several disjoint contours for the city.

4) **New York**

*Note*: The cumulus of streets, tall skyscrapers, halls, parks, piers and avenues makes New York a mesmerising sight, especially in satellite view. It's interesting to see that the algorithm manages to separate higher and lower density areas. Moreover there seems not to be a clear distinction between Central Park and the surrounding neighbourhood, which can indicate that from a complexity point of view, nature and man-made structures are more or less on the same page.

## IV. Results - Irregularity Detection in Photos

For this test, a completely separate image (the dog) was added to stock images in order to evaluate whether irregularity detection can work for foreign objects in regular photos, too.

Fig. 5: Foreign object detection

*Note*: It's important to specify that the image of the dog was copied over the original photo. This means that there is no natural blending of shadows, which should make it much more visible to the algorithm.

## V. Python Code

The Python Code can be found on GitHub at:

`https://github.com/tdrvlad/Irregularity-Detection-with-Box-Counting`