2.3.1: Image pyramids

Topic Progress: (https://gurus.pyimagesearch.com/topic/image-pyramids/)

(https://gurus.pyimagesearch.com/topic/sliding-windows/)

← Back to Lesson (https://gurus.pyimagesearch.com/lessons/sliding-windows-and-image-pyramids/)

Now that we've seen **the easy way (https://gurus.pyimagesearch.com/topic/object-detection-made-easy/)** to build a custom object detector, let's take a step back and consider all the required steps to create a fully-functional object detection framework from scratch. Object detection systems are not easy to build — there are many components and moving parts that we need to put into place. Throughout the remainder of this module, we'll be dissecting the object detection pipeline, exploring each step in detail, and writing code to ***create our own custom object detection framework***.

This lesson will focus on the first step of building our framework: *scanning an image at multiple locations and scales*. The goal of the scanning step in an object detection framework is to facilitate a method to *localize* the object in an image, regardless of *where* in the image the object appears and how *large/small* it is.

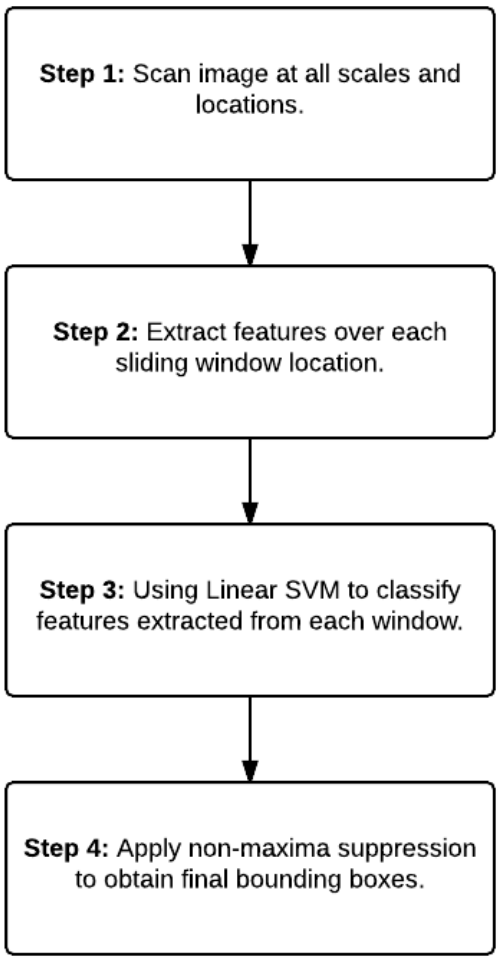
Objectives:

In this lesson, we will:

- Discuss image pyramids for object detection.
- Implement our own image pyramid using Python and OpenCV.

Image pyramids

To create our own custom object detection framework, we'll need to implement (at a bare minimum) the following pipeline:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/image_pyramid_pipeline.png)

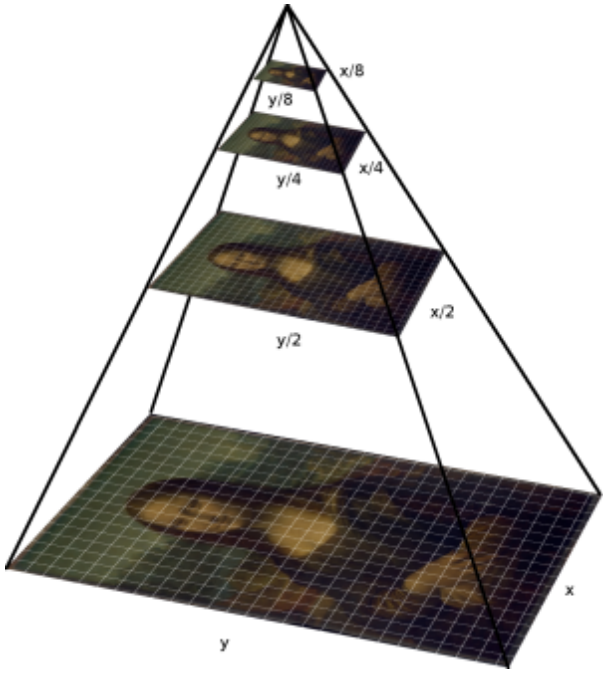
FIGURE 1: THE PIPELINE OF CONSTRUCTING AN OBJECT DETECTOR.

As we can see from the flowchart above, the first step when creating our custom object detection framework is to implement the "scanning" functionality, which will enable us to find objects in images at *various sizes and locations*.

This "scanner" can be broken into two components:

- **Component #1:** An image pyramid.
- **Component #2:** A sliding window.

An ***image pyramid*** is simply a ***multi-scale representation*** of an image:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/brisk_pyramid_example.png)

FIGURE 2: AN EXAMPLE OF CONSTRUCTING AN IMAGE PYRAMID BY HALVING THE SIZE OF THE IMAGE AT EACH LAYER.

Utilizing an image pyramid allows us to **find objects in images at different scales** of an image.

At the bottom of the pyramid, we have the original image at its original size (in terms of width and height). At each subsequent layer, the image is resized (subsampled) and optionally smoothed via Gaussian blurring.

The image is progressively subsampled until some stopping criterion is met, which is normally a minimum size being reached, indicating that no further subsampling needs to take place.

When combined with the second component of our "scanner" the ***sliding window***, we can find objects in images at *various locations*. As the name suggests, a *sliding window* "slides" from left to right and top to bottom of *each scale* in the image pyramid. Again, by leveraging both image pyramids and sliding windows together, we are able to detect objects at various locations and scales in an image.

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/image_pyramid_sliding_window_example.gif)

FIGURE 3: COMBINING BOTH AN IMAGE PYRAMID AND SLIDING WINDOWS ALLOWS US TO DETECT OBJECTS AT VARIOUS SCALES AND LOCATIONS IN AN IMAGE. THE GREEN BOX REPRESENTS OUR SLIDING WINDOW. NOTICE HOW THE SLIDING WINDOWS STOP AT MY FACE — GIVEN A CLASSIFIER TRAINED ON FACE DATA, MY FACE COULD BE DETECTED.

In the remainder of this lesson, we'll be focusing on the first component of our scanner, the *image pyramid*. Specifically, we'll learn how to implement an image pyramid using the OpenCV library. In our next lesson, we'll implement the *sliding window* and combine it with our *image pyramid* to complete the "scanning" functionality of our object detection framework.

Implementing an image pyramid

Earlier in this lesson, I mentioned that this module will focus on creating a custom object detection framework that we can use to train and deploy object detectors of our own. With that in mind, let's start off by examining the basic file structure required to implement the "scanning" step of our framework:

```
Directory structure of our object detector                    Shell
```

```
|--- pyimagesearch
|    |--- __init__.py
|    |--- object_detection
|    |    |---- __init__.py
|    |    |--- helpers.py
|--- test_pyramid.py
|--- test_sliding_window.py
```

Inside the root `pyimagesearch` module, we'll create an `object_detection` sub-module. As the name suggests, this sub-module is going to store all code specifically related to object detection.

We'll then create a `helpers.py` file inside `object_detection` where we'll define two methods: `pyramid` and `sliding_window`.

Last, we have the `test_pyramid.py` and `test_sliding_window.py` scripts that will be used as drivers to test our image pyramid and sliding window implementations, respectively.

Now that we have the directory structure of scanning step laid out, let's go ahead and define the `pyramid` function now:

helpers.py Python

```python
# import the necessary packages
import imutils

def pyramid(image, scale=1.5, minSize=(30, 30)):
    # yield the original image
    yield image

    # keep looping over the pyramid
    while True:
        # compute the new dimensions of the image and resize it
        w = int(image.shape[1] / scale)
        image = imutils.resize(image, width=w)

        # if the resized image does not meet the supplied minimum
        # size, then stop constructing the pyramid
        if image.shape[0] < minSize[1] or image.shape[1] < minSize[0]:
            break

        # yield the next image in the pyramid
        yield image
```

We'll start off by importing the `imutils` package which contains a handful of image processing convenience functions — specifically, we'll be making use of the `resize` method.

Next up, we define our `pyramid` function on **Line 4**. This function takes two arguments. The first argument is the `scale`, which controls how much the image is resized at each layer. A small `scale` yields more layers in the pyramid, while a larger `scale` yields less layers.

Second, we define the `minSize`, which is the minimum required width and height of the layer. If an image in the pyramid falls below this `minSize`, we stop constructing the image pyramid.

Line 6 yields the original image in the pyramid (the bottom layer).

From there, we start looping over the image pyramid on **Line 9**.

Lines 11 and 12 handle computing the size of the image in the next layer of the pyramid (while preserving the aspect ratio). This scale is controlled by the `scale` factor.

At this point, you need to decide whether or not you want to apply Gaussian blurring to smooth the layer. Traditional image pyramids implemented in OpenCV via the `pyrUp` and `pyrDown` methods *do* perform smoothing at each level of the pyramid. **However, we need to consider the context of our image processing pipeline.** As the work of Dalal and Triggs showed (https://gurus.pyimagesearch.com/wp-content/uploads/2015/05/dalal_2005.pdf), applying Gaussian smoothing at each layer of the pyramid can actually *hurt the performance* of the HOG descriptor — hence we skip this step in our image pyramid.

That said, it may not *always* be the case that Gaussian smoothing reduces accuracy. I would suggest running two experiments, one with smoothing and one without to see if accuracy substantially changes. In most cases, you'll see that accuracy *drops* when smoothing is performed at each layer, hence why I am leaving out the Gaussian blur. However, there may be some outlier datasets and circumstances where blurring can actually help — *so keep this in mind!*

On **Lines 16 and 17**, we make a check to ensure that the image meets the `minSize` requirements. If it does not, we break from the loop.

Finally, **Line 20** yields our resized image.

As you can see, there isn't much to the `pyramid` function! It's quite simple, but the utility it gives us is invaluable when applied to find objects in images at various scales.

That said, let's see our `pyramid` function in action. Open up our `test_pyramid.py` file and insert the following code:

`test_pyramid.py` — Python

```
# import the necessary packages
from pyimagesearch.object_detection.helpers import pyramid
import argparse
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True, help="path to the input image")
ap.add_argument("-s", "--scale", type=float, default=1.5, help="scale factor size")
args = vars(ap.parse_args())

# load the input image
image = cv2.imread(args["image"])

# loop over the layers of the image pyramid and display them
for (i, layer) in enumerate(pyramid(image, scale=args["scale"])):
    cv2.imshow("Layer {}".format(i + 1), layer)
    cv2.waitKey(0)
```

The code here is fairly straightforward. We start off by importing our packages, parsing our command line arguments, and loading our original image from disk. We'll specify a default `--scale` value of `1.5` to indicate that the size of the image should be reduced by *1.5x* the size of the previous layer at each layer of the pyramid.

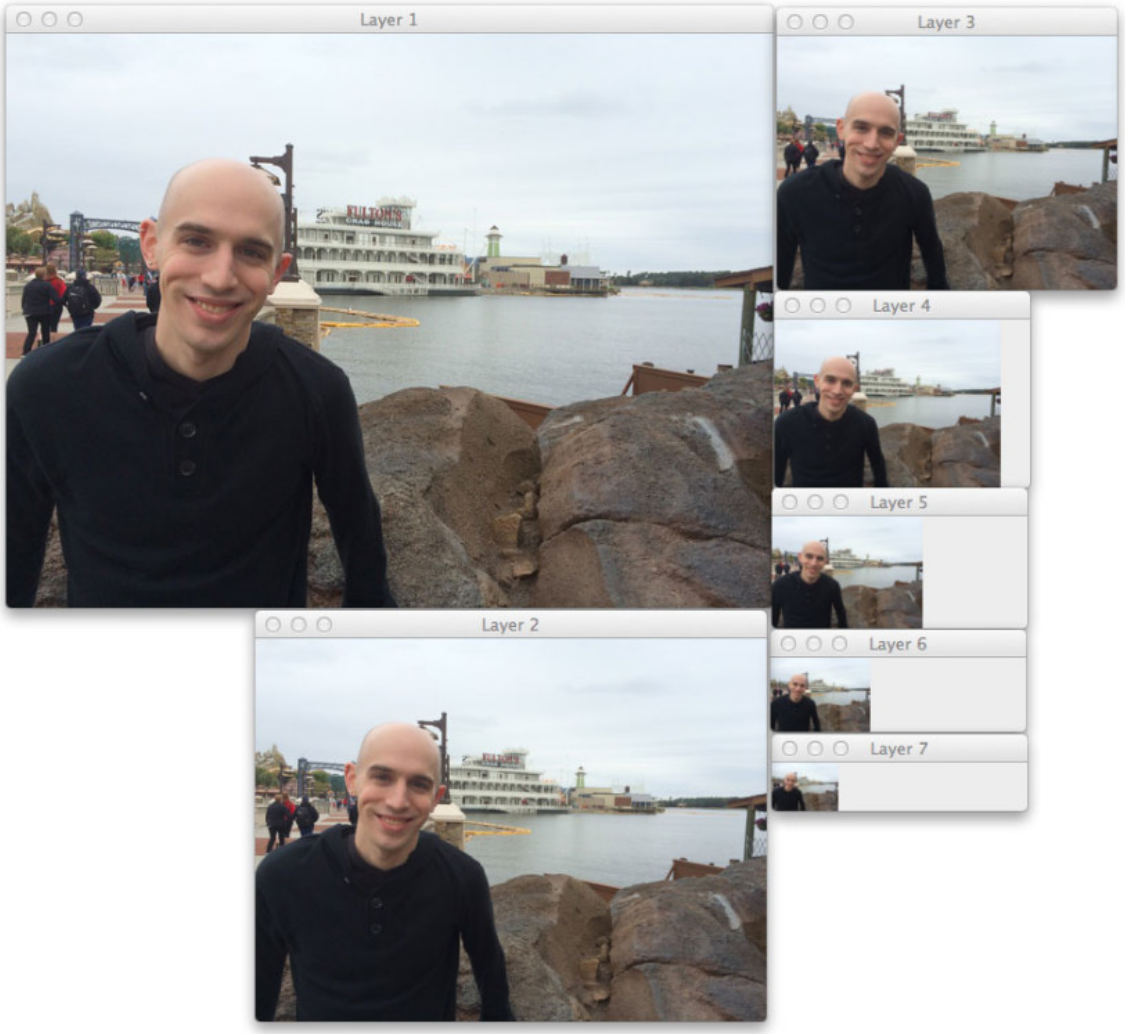
Finally, **Lines 16-18** handle looping over each layer of the pyramid and displaying the results in our screen.

To see our script in action, open up a terminal, change the directory to where your code is located, and execute the following command:

test_pyramid.py — Shell

```
$ python test_pyramid.py --image florida_trip.png --scale 1.5
```

You should see results similar to this:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/image_pyramid_output.jpg)

FIGURE 4: CONSTRUCTING AN IMAGE PYRAMID WITH SEVEN LAYERS.

Here, we can see that seven layers have been generated from the original input image, where each subsequent layer is smaller than the previous one. Layer creation terminates when the image size falls below the minimum *30 x 30* pixel dimensions.

In general, there is a tradeoff between performance and the number of layers that you generate. The smaller your scale factor is, the more layers you need to create and process — but this also gives your image classifier a better chance at localizing the object you want to detect in the image.

A larger scale factor will yield less layers, and perhaps might hurt your object classification performance; however, you will obtain much higher performance gains (in terms of image throughput) since you will have less layers to process.

Summary

In this lesson, we discussed the concept of image pyramids for object detection. Image pyramids are simply a multi-scale representation of an image. Combined with sliding windows, we can find objects in images at *multiple scales and locations* in an image. This is an absolutely critical step in the object detection pipeline.

We then implemented our `pyramid` method using OpenCV. In fact, this is the *same method* that I use in my own personal projects! Unlike the traditional image pyramid implemented inside OpenCV, our `pyramid` method *does not* smooth the image with a Gaussian blur at each layer of the pyramid, thus making it more acceptable for use with the HOG descriptor.

In our next lesson, we'll finish the "scanning" component of our object detection framework by implementing *sliding windows*.

Downloads:

Download the Code (https://gurus.pyimagesearch.com/protected/code/object_detector/image_pyram

Feedback

Quizzes	Status
1	Image Pyramids Quiz (https://gurus.pyimagesearch.com/quizzes/image-pyramids-quiz/)

Next Topic → (https://gurus.pyimagesearch.com/topic/sliding-windows/)

Course Progress

Ready to continue the course?

Click the button below to **continue your journey to computer vision guru**.

I'm ready, let's go! (/pyimagesearch-gurus-course/)

Resources & Links

- PyImageSearch Gurus Community (https://community.pyimagesearch.com/)
- PyImageSearch Virtual Machine (https://gurus.pyimagesearch.com/pyimagesearch-virtual-machine/)
- Setting up your own Python + OpenCV environment (https://gurus.pyimagesearch.com/setting-up-your-python-opencv-development-environment/)
- Course Syllabus & Content Release Schedule (https://gurus.pyimagesearch.com/course-syllabus-content-release-schedule/)
- Member Perks & Discounts (https://gurus.pyimagesearch.com/pyimagesearch-gurus-discounts-perks/)
- Your Achievements (https://gurus.pyimagesearch.com/achievements/)
- Official OpenCV documentation (http://docs.opencv.org/index.html)

Your Account

- Account Info (https://gurus.pyimagesearch.com/account/)
- Support (https://gurus.pyimagesearch.com/contact/)
- Logout (https://gurus.pyimagesearch.com/wp-login.php?action=logout&redirect_to=https%3A%2F%2Fgurus.pyimagesearch.com%2F&_wpnonce=5736b21cae)

Search

Feedback

/