2.3.2: Sliding windows

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/)

In our previous lesson, we discovered **how to construct an image pyramid (https://gurus.pyimagesearch.com/topic/image-pyramids/)**.

Today, we are going to extend the image pyramid example and introduce the concept of a ***sliding window***. Sliding windows play an integral role in object classification, as they allow us to localize exactly *where* in an image an object resides.

Utilizing both a sliding window and an image pyramid, we are able to detect objects in images at various scales and locations.

Objectives:

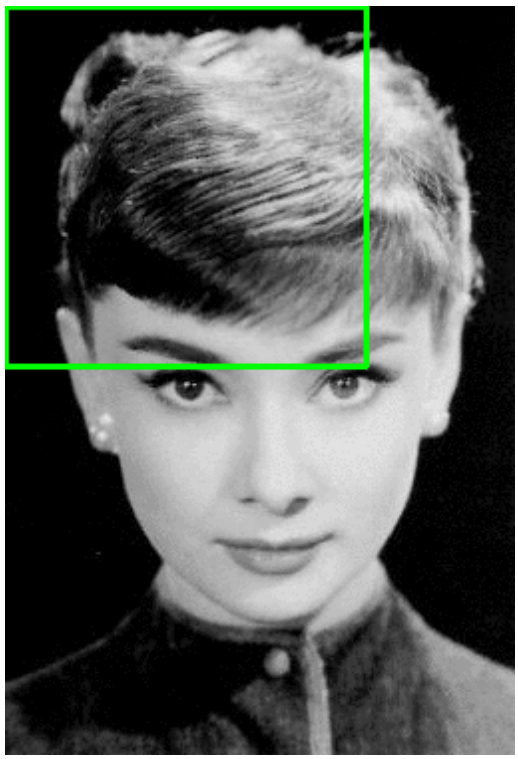
In this lesson, we will:

- Discover what a *sliding window* is.
- Learn how image pyramids and sliding windows work in conjunction with each other when detecting objects at various scales and locations in an image.
- Implement our own `sliding_window` method.

Sliding windows

So what exactly is a *sliding window*?

In the context of computer vision and machine learning (and as the name suggests), a sliding window is a rectangular region of fixed width and height that "slides" across an image, such as in the following figure:



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

FIGURE 1: EXAMPLE OF THE SLIDING WINDOW APPROACH, WHERE WE SLIDE A WINDOW FROM LEFT-TO-RIGHT AND TOP-TO-BOTTOM.

For each of these windows, we would *take the window region*, *extract features from it*, and *apply an image classifier* to determine if the window contains an object that interests us — in this case, a face.

Combined with **image pyramids (https://gurus.pyimagesearch.com/topic/image-pyramids/)**, we can create image classifiers that can **recognize objects at varying *scales* and *locations* in an image**.

These techniques, while simple, play an *absolutely critical role* in object detection and image classification.

Implementing a sliding window

Let's go ahead and build on our image pyramid example from the last lesson (https://gurus.pyimagesearch.com/topic/image-pyramids/).

Remember the `helpers.py` file? Open it back up and insert the `sliding_window` function:

helpers.py Python

```
22 def sliding_window(image, stepSize, windowSize):
23     # slide a window across the image
24     for y in range(0, image.shape[0], stepSize):
25         for x in range(0, image.shape[1], stepSize):
26             # yield the current window
27             yield (x, y, image[y:y + windowSize[1], x:x + windowSize[0]])
```

The `sliding_window` function requires three arguments. The first is the `image` that we are going to loop over. The second argument is the `stepSize` .

The stepSize indicates how many pixels we are going to "skip" in both the *(x, y)* direction. Normally, we would ***not*** want to loop over each and every pixel of the image (i.e. `stepSize=1`) as this would be computationally prohibitive if we were applying an image classifier at each window.

Instead, the `stepSize` is determined on a *per-dataset* basis and is tuned to give optimal performance based on your dataset of images. In practice, it's common to use a `stepSize` of 4 to 8 pixels. ***Remember, the smaller your step size is, the more windows you'll need to examine.*** The larger your step size is, the less windows you'll need to examine — note, however, that while this will be computationally more efficient, you may miss detecting objects in images if your step size becomes too large.

The last argument, `windowSize` , defines the width and height (in terms of pixels) of the window we are going to extract from our `image` .

Lines 24-27 are fairly straightforward and handle the actual "sliding" of the window.

Lines 24-26 define two `for` loops that loop over the *(x, y)* coordinates of the image, incrementing their respective `x` and `y` counters by the provided step size.

Then, **Line 27** returns a tuple containing the `x` and `y` coordinates of the sliding window, along with the window itself.

To see the sliding window in action, we'll have to write a driver script for it. Open up the `test_sliding_window.py` file and insert the following code:

test_sliding_window.py Python

Feedback

```python
# import the necessary packages
from pyimagesearch.object_detection.helpers import sliding_window
from pyimagesearch.object_detection.helpers import pyramid
import argparse
import time
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("-w", "--width", required=True, type=int, help="width of sliding window")
ap.add_argument("-t", "--height", required=True, type=int, help="height of sliding window")
ap.add_argument("-s", "--scale", type=float, default=1.5, help="scale factor size")
args = vars(ap.parse_args())

# load the input image and unpack the command line arguments
image = cv2.imread(args["image"])
(winW, winH) = (args["width"], args["height"])
```

Lines 2-6 start by importing our necessary packages — nothing too special here, we're just grabbing our `sliding_window` and `pyramid` methods.

We then move on to parsing our command line arguments on **Lines 9-14**. We'll require three switches and a fourth optional one. The `--image` switch is the path to our image on disk that we want to run an image pyramid and sliding windows on. The `--width` and `--height` switches define the width and height of the sliding window bounding box, respectively. Finally, we'll (optionally) parse the `--scale`, which is the resizing factor of the image pyramid.

Lines 17 and 18 then load our image from disk and put the width and height of the sliding windows into convenience variables.

Now, let's actually apply the image pyramid and sliding window together:

test_sliding_window.py — Python

```
20 # loop over the image pyramid
21 for layer in pyramid(image, scale=args["scale"]):
22     # loop over the sliding window for each layer of the pyramid
23     for (x, y, window) in sliding_window(layer, stepSize=32, windowSize=(winW, winH)):
24         # if the current window does not meet our desired window size, ignore it
25         if window.shape[0] != winH or window.shape[1] != winW:
26             continue
27 
28         # THIS IS WHERE WE WOULD PROCESS THE WINDOW, EXTRACT HOG FEATURES, AND
29         # APPLY A MACHINE LEARNING CLASSIFIER TO PERFORM OBJECT DETECTION
30 
31         # since we do not have a classifier yet, let's just draw the window
32         clone = layer.copy()
33         cv2.rectangle(clone, (x, y), (x + winW, y + winH), (0, 255, 0), 2)
34         cv2.imshow("Window", clone)
35 
36         # normally we would leave out this line, but let's pause execution
37         # of our script so we can visualize the window
38         cv2.waitKey(1)
39         time.sleep(0.025)
```

We start by looping over each layer of the image pyramid on **Line 21**.

For each layer of the image pyramid, we'll also loop over each window in the `sliding_window` on **Line 23**. We also make a check on **Lines 25 and 26** to ensure that our sliding window has met the minimum size requirements.

If we were applying a machine learning classifier to detect objects, we would do this on **Lines 28 and 29** by extracting features from the window and passing them on to our classifier. Our classifier would then return a boolean indicating, *"Yes,"* this window does contain an object you are interested in; or *"No,"* this window does not contain an object.

Since we do not have an image classifier, we'll just visualize the sliding window results by drawing a rectangle on the current layer of the pyramid indicating where the sliding window is (**Lines 32-39**).

Visualizing our sliding window

To see our image pyramid and sliding window in action, fire up a terminal and execute the following command:

test_sliding_window.py — Shell

```
1 $ python test_sliding_window.py --image florida_trip.png --width 64 --height 64
```

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

FIGURE 2: AN EXAMPLE OF SLIDING A *64 X 64* WINDOW ACROSS AN IMAGE. IN THIS CASE, WE WOULD BE TRYING TO FIND FACES AT EACH SCALE AND LOCATION IN THE IMAGE PYRAMID.

Here, you can see that we have specified a *64 x 64* pixel sliding window — for each of the layers in the pyramid a window is "slid" across it. Again, if we had an image classifier ready to go, we could take each of these windows and classify the contents of the window. **An example could be *"does this window contain a face or not?"***

Choosing an appropriate sliding window size is *critical* to obtaining a high-accuracy object detector — along with associated **Histogram of Oriented Gradients parameters (https://gurus.pyimagesearch.com/lessons/histogram-of-oriented-gradients/)**, but we'll save that for another lesson.

For example, let's try another sliding window, this time specifying a window of size *96 x 36* pixels:

```
test_sliding_window.py                                                    Shell
1 $ python test_sliding_window.py --image car.jpg --width 96 --height 36
```

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

FIGURE 3: HERE, WE HAVE ADJUSTED THE SIZE OF OUR SLIDING WINDOW, MAKING IT RECTANGULAR AND MORE SUITABLE FOR FINDING CARS IN IMAGES.

Being rectangular, this sliding window is more aptly suited for detecting the presence of cars in images than faces. It wouldn't make much sense to apply the *square* sliding window size from **Figure 2** above if our goal is to detect cars. Motor vehicles are naturally *rectangular* in shape, being longer than they are tall, hence we would want to choose a sliding window size that reflects this geometry as well.

Before you start developing your own custom object detector, be sure to consider the shape, size, and aspect ratio of the objects you are trying to detect — these values will help you choose appropriate values for the sliding window size.

Summary

In this lesson, we learned all about sliding windows and their application to object detection and image classification. By combining a sliding window with an image pyramid, we will be able to localize and detect objects in images at multiple ***scales*** and ***locations***.

While both sliding windows and image pyramids are very simple techniques, they are absolutely critical in object detection.

Coming up in the next lesson, we'll review the **6-step framework to object detection (https://gurus.pyimagesearch.com/lessons/getting-started-with-dalal-and-triggs-object-detectors/)** proposed by Dalal and Triggs. This framework is the standard method used to detect objects in images using HOG + Linear SVM and is the framework we'll be implementing in the rest of this module.

Downloads:

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

	Quizzes	Status
1	Sliding Windows Quiz (https://gurus.pyimagesearch.com/quizzes/sliding-windows-quiz/)	

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

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)