10.1: What are image descriptors, feature descriptors, and feature vectors?

Today we are getting started on one of my personal favorite topics in computer vision: **how to quantify and abstractly represent an image using only a list of numbers**.

The process of quantifying an image is called *feature extraction*. The process of feature extraction governs the rules, algorithms, and methodologies we use to abstractly quantify the contents of an image using *only* a list of numbers, called a *feature vector*.

Image descriptors and *feature descriptors* govern ***how*** an image is abstracted and quantified, while *feature vectors* are the output of descriptors and used to quantify the image. Taken as a whole, this process is called *feature extraction*.

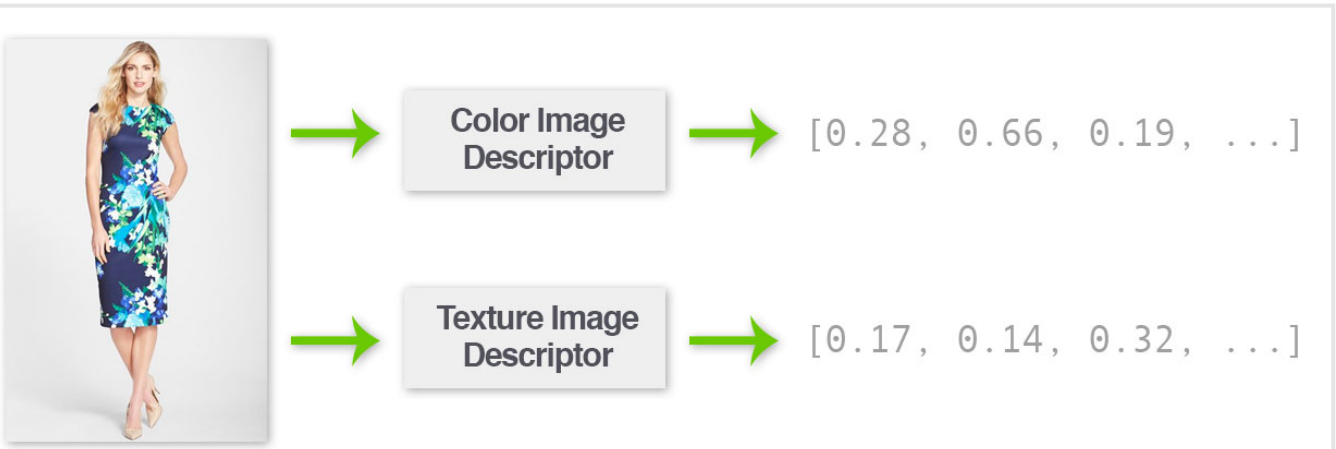
The process of feature extraction has been around for as long as computer vision has existed. The methods have changed and we have gotten much better at doing it — but the general process of inputting an image, running a feature extraction algorithm on the image, and outputting a list of numbers to quantify the image remains the same.

Nearly any algorithm that attempts to understand and interpret the contents of an image utilizes feature extraction at some stage. We could be extracting features from an image for a variety of reasons; to compare to images for similarity; to rank images in search results when building an image search engine; or to use when training an image classifier to recognize the contents of an image.

To perform these comparisons, we need *feature vectors*, or simply *features*.

Feature vectors are used to represent a variety of properties of an image, such as the shape, color, or texture of an object in an image. They can also *combine* various properties. A feature vector could jointly represent shape and color. Or it could represent texture and shape. Or it could represent all three!

For example, when I built Chic Engine I used algorithms to quantify the *texture* and *color* of a piece of clothing in an image:

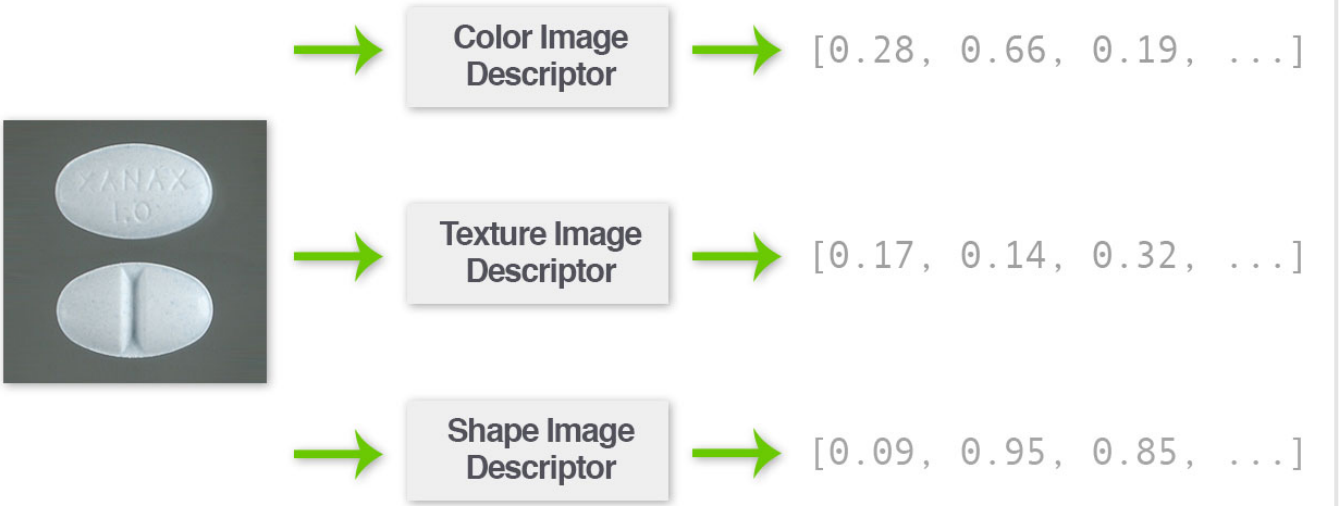


(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/descriptors_chic_engine.jpg)

FIGURE 1: AN EXAMPLE OF APPLYING AN IMAGE DESCRIPTOR TO EXTRACT TWO FEATURE VECTORS — ONE TO DESCRIBE COLOR, AN ANOTHER TO CHARACTERIZE TEXTURE.

Feedback

And when I created ID My Pill, I used custom methodologies to extract feature vectors to represent the *shape*, *color*, and *texture* of the pill in an image:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/descriptors_idmypill.jpg)

FIGURE 2: DEVELOPING ID MY PILL, I EXTRACTED THREE FEATURE VECTORS TO DESCRIBE A PILL: ONE FOR COLOR, ANOTHER FOR TEXTURE, AND A FINAL ONE FOR SHAPE.

As you can see, I rely on feature extraction heavily in work outside of PyImageSearch Gurus, as do many, many other computer vision researchers, developers, and programmers.

But before we take a deep dive into image quantification and feature extraction, we first need to define a few terms– namely *image descriptors*, *feature descriptors*, and *feature vectors*.

These terms all look similar and share many of the same words. In fact, in the computer vision literature these terms are sometimes (if not often) used interchangeably! The only way to tell the difference between these terms comes from the author's context and how they are using the term in (1) a sentence and (2) in the grand scheme of their work.

So with that all said, it becomes extremely important for us to make sure that we understand when to use each one. It's also crucial for us to understand these terms so we can understand the work of other computer vision developers. You'll see these terms throughout the rest of your computer vision career, so make sure you take the time to digest them now!

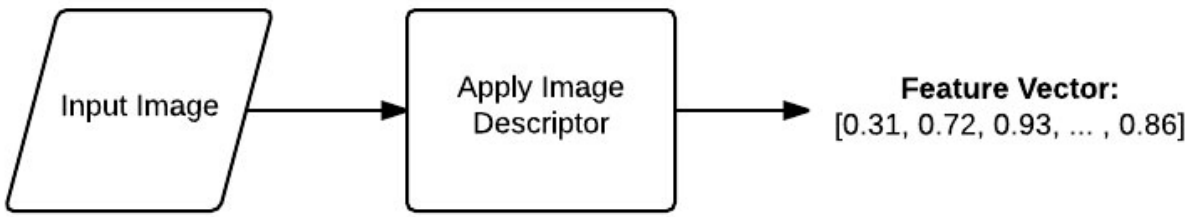
Objective:

Our primary objective in this lesson is to learn about common terms in the computer vision literature related to quantifying and abstractly representing the contents of an image — specifically:

1. Feature vector
2. Image descriptor
3. Feature descriptor

Feature vector

The general process of extracting a *feature vector* from an image is depicted below:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/descriptors_basic_process.jpg)

FIGURE 3: THE PIPELINE OF AN IMAGE DESCRIPTOR. AN INPUT IMAGE IS PRESENTED TO THE DESCRIPTOR, THE IMAGE DESCRIPTOR IS APPLIED, AND A FEATURE VECTOR (I.E A LIST OF NUMBERS) IS RETURNED, USED TO QUANTIFY THE CONTENTS OF THE IMAGE.

On the *left* we have our image which is fed to the image descriptor. We then apply our image descriptor algorithm to the input image *(center)*. On the *right* we are left with a *feature vector* as the output of the image descriptor.

So you're probably scratching your head and wondering two questions:

1. If the feature vector is the final step in the process, why are we discussing it first in this lesson?
2. What exactly *is* a feature vector?

To answer your first question, we are discussing *feature vectors* early in this lesson because they tend to be the easiest term to understand. Furthermore, both *image descriptors* and *feature descriptors* output feature vectors, so it's important to understand the term now.

As for your second question, let's see if this definition of a *feature vector* helps clear things up a bit:

> ***Image Feature Vector:*** *An abstraction of an image used to characterize and numerically quantify the contents of an image. Normally real, integer, or binary valued. Simply put, a feature vector is a list of numbers used to represent an image.*

Feedback

So there you go — a feature vector is just a list of numbers. Given an *N x M* pixel image, we input it to our image descriptor, and out pops a *d*-dimensional feature out the end of the image descriptor. The value of *d* is the *length*, or the number of entries inside the list.

For example, a feature vector with *128* entries is called *128-dimensional*, or simply *128-d* for short. And again, a feature vector with *1,024* entries in the list is of *1,024 dimensionality* or or simply *1,024-d*. It will be very common to see feature vectors expressed and notated in this manner when reading through the academic literature on computer vision and feature extraction.

It's common to hear the term *describing* when we derive a feature vector from an image. For example, take a look at the photo of myself in Florida below — how might you use words to describe it?

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/florida_trip.jpg)

FIGURE 4: WHAT WORDS WOULD YOU USE TO "DESCRIBE" THIS IMAGE?

Well, we can clearly see a *person* in the foreground. We can see a *face*. We can also see that there is some body of *water* in the background along with a *boat*. There is also the *sky* in the top portion of the image.

Pulling out the keywords from our above description, our "feature vector" of words would look something like this:

feature vector = [person, face, water, boat, sky]

But without applying some (very fancy) machine learning, our computers cannot derive this high level, English language description of the image. Instead, our systems must be rely on *feature vectors*.

So instead of our feature vector being a list of words, it would look more like a list of numbers:

feature vector = [0.45, 0.16, 0.42, 0.23, 0.22]

Simply put: a feature vector is nothing more than a list of numbers used to represent and quantify an image.

Of course, these feature vectors cannot be arbitrary and filled with random, meaningless values! We need to define methodologies to examine our images and extract meaningful feature vectors from them.

These algorithms and methodologies used to extract feature vectors are called *image descriptors* and *feature descriptors*, which we'll be covering in the remainder of this lesson.

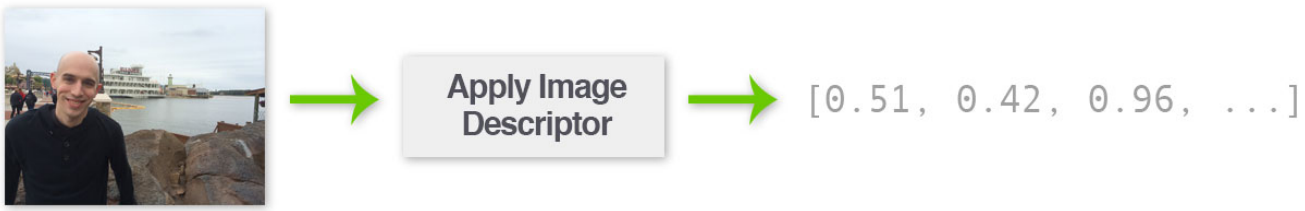
Image Descriptor

Now that we understand what feature vectors are, let's see if we can wrap our heads around *image descriptors*:

> ***Image Descriptor:*** *An image descriptor is an algorithm and methodology that governs how an input image is* ***globally*** *quantified and returns a feature vector abstractly representing the image contents.*

The key term to understand here is ***global*** — this implies that we are examining the ***entire*** image and using the ***whole*** image in the computation of our feature vector.

That's right. Not part of an image. Not a fraction of an image. Not 99% of the image. No, we are taking the ***whole entire*** image and extracting an image descriptor to represent its contents:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/descriptors_florida_trip.jpg)

FIGURE 5: APPLYING AN IMAGE DESCRIPTOR TO DESCRIBE AND QUANTIFY THE ***ENTIRE*** IMAGE.

Just like in **Figure 3**, we are presented with an input image. We then take this input image and apply our *image descriptor* to it. This image descriptor could be quantifying the shape, color, texture, or any combination of the three. Finally, the output of our image descriptor is a list of numbers — our *feature vector* or simply *features*. When presented with 1 input image, our image descriptor will return 1 feature vector.

Examples of image descriptors include **color channel statistics (https://gurus.pyimagesearch.com/lessons/color-channel-statistics/)**, **color histograms (https://gurus.pyimagesearch.com/lessons/color-histograms/)**, and **Local Binary Patterns (https://gurus.pyimagesearch.com/lessons/local-binary-patterns/)**, to name a few. We'll be discussing these image descriptors, and many more, later in this course.

One of the primary benefits of image descriptors is that they tend to be much simpler than feature descriptors. Furthermore, once extracted, the feature vectors derived from image descriptors can be immediately passed down the pipeline to other computer vision methods, such as creating a classifier to recognize the contents of an image or building an image search engine.

However, this simplicity often comes at a price. Often times, while basic and simple to use, our image descriptors are not robust to changes in how the image is rotated, translated, or how viewpoints of an image change. If that is the case, we'll often times need to use the more powerful feature descriptors.

But what happens if we wanted to describe *multiple* regions of an image? What if, in the above figure, we wanted to extract features from the face region? And the water region? And the sky region? How might we go about that? And are there methods to *automatically detect* the regions of an image that we should be focusing on, describing, and extracting features from?

It turns out, yes. And the answer to all these questions is *feature descriptors*.

Feature Descriptors

Let's start off with a definition:

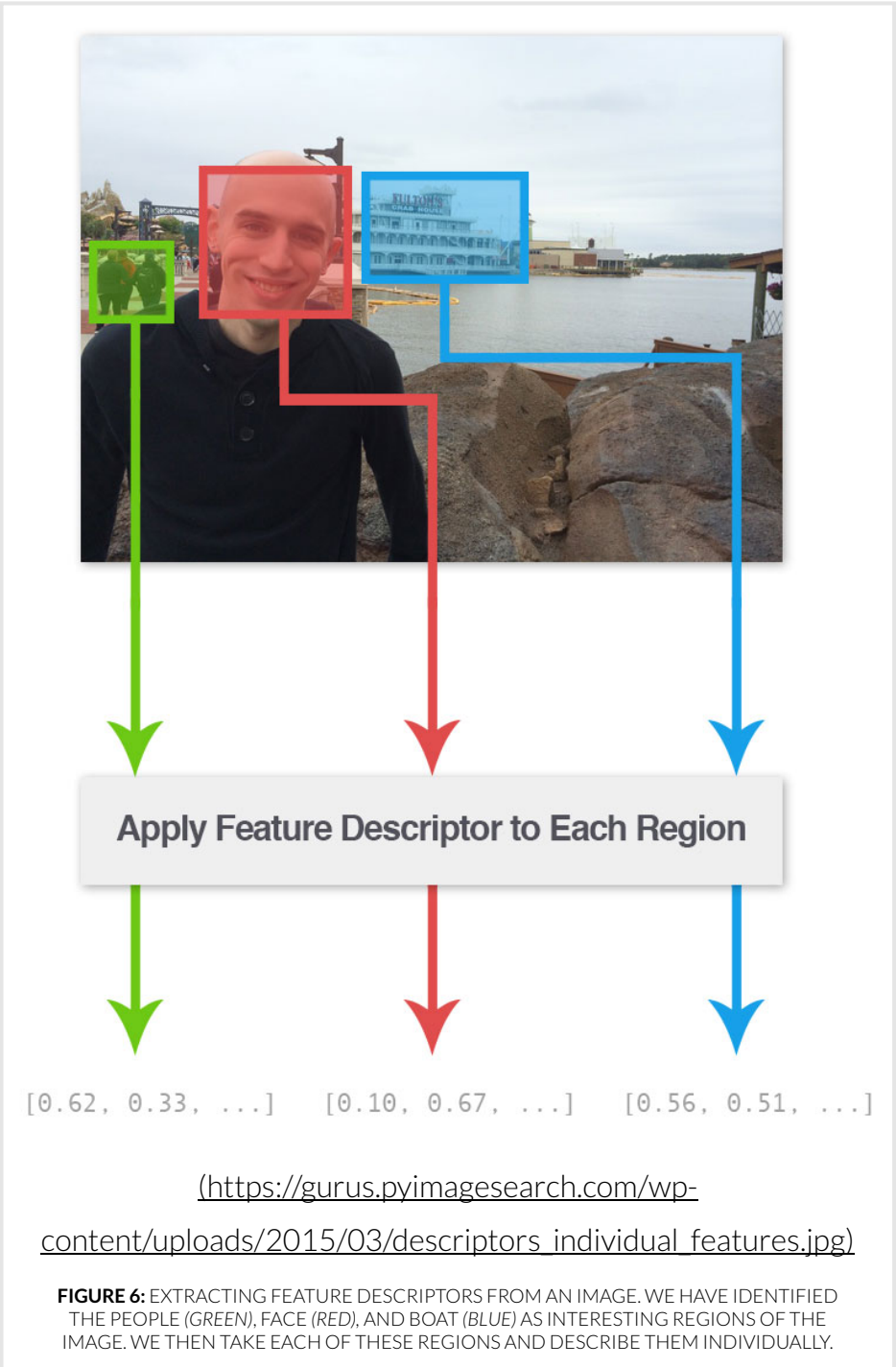
> ***Feature Descriptor:*** *A feature descriptor is an algorithm and methodology that governs how an input* ***region of an image*** *is* ***locally*** *quantified. A feature descriptor accepts a* single *input image and returns* multiple *feature vectors.*

Unlike an image descriptor where, for each input image, there was only *one* output feature vector, a feature descriptor can return *multiple* feature vectors per image:

- **Image descriptor:** 1 image in, 1 feature vector out.

- **Feature descriptor:** 1 image in, many feature vectors out.

To make this point clear, here is a visual depiction of a feature descriptor in action:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/descriptors_individual_features.jpg)

FIGURE 6: EXTRACTING FEATURE DESCRIPTORS FROM AN IMAGE. WE HAVE IDENTIFIED THE PEOPLE *(GREEN)*, FACE *(RED)*, AND BOAT *(BLUE)* AS INTERESTING REGIONS OF THE IMAGE. WE THEN TAKE EACH OF THESE REGIONS AND DESCRIBE THEM INDIVIDUALLY.

On the *top* we have our original input image. We would normally apply keypoint detectors to find the "interesting" regions of an image, but this example we have labeled the people *(green)*, my face *(red)*, and the boat *(blue)* as the "interesting" regions. Each of these image regions is then fed into our *feature descriptor*. The feature descriptor then extracts multiple *feature vectors* — one for each region — and thus quantifying the three regions of the image.

In practice, we only want to describe (i.e. extract features from) the salient or "interesting" regions of an image. It's common for us to apply keypoint detectors to find "interesting" regions of an image. These regions are then used as input to our feature descriptors — given *N* interesting regions of an image to describe, we will receive *N* feature vectors back from the feature descriptor.

Examples of feature descriptors include SIFT, SURF, ORB, BRISK, BRIEF, and FREAK — and again, we'll be covering all of these descriptors inside the rest of the PyImageSearch Gurus course.

Feature descriptors tend to be much more **powerful** than our basic image descriptors since they take into account the locality of regions in an image and describe them in separately. As you'll see later in this section, feature descriptors also tend to be much more robust to changes in the input image, such as rotation, translation, orientation (i.e. rotation), and changes in viewpoint.

However, this robustness and ability to describe multiple regions of an image comes at a price — in most cases the feature vectors extracted using feature descriptors *are not* directly applicable to building an image search engine or constructing an image classifier in their current state (the exception being **keypoint matching/spatial verification (https://gurus.pyimagesearch.com/lessons/spatial-verification/)**, which we detail when **identifying the covers of books (https://gurus.pyimagesearch.com/lessons/identifying-the-covers-of-books/)**). This is because each image is now represented by *multiple* feature vectors rather than just *one*.

To remedy this problem, we construct a **bag-of-visual-words**, which takes all the feature vectors of an image and constructs a histogram, counting the number of times similar feature vectors occur in an image. Sound confusing? It's actually not — and the bag-of-visual-words model is literally my favorite topic in all of computer vision, so we'll have a lot of fun when we get to it.

Summary

We started this lesson by defining three important terms that are prevalent throughout all of computer vision: *image descriptors*, *feature descriptors*, and *feature vectors* (or simply *features*).

These terms all look very similar and share many of the same words. However, it's important to understand the difference between them, as some computer vision researchers, developers, and programmers use them interchangeably and use the context of how the terms are used to define their meaning.

To start, a *feature vector* is simply a list of numbers used to abstractly quantify the contents of an image. Feature vectors are then passed down the pipeline to other computer vision programs, such as creating a machine learning classifier to recognize the contents of image using feature vectors; or comparing feature vectors for similarity when building an image search engine.

To extract *feature vectors* from an image we can use either *image descriptors* or *feature descriptors*.

An *image descriptor* quantifies the **entire** image and returns one feature vector per image. Image descriptors tend to be simple and intuitive to understand, but can lack the power to distinguish between different objects in images.

Conversely, a *feature descriptor* quantifies many **regions** of an image, returning multiple feature vectors per image. Feature descriptors tend to be much more powerful than simple image descriptors and more robust to changes in the input image's rotation, translation, and viewpoint.

However, the added discrimination power comes at an added cost. Not only do we have to store multiple feature vectors per image, which increases our storage overhead, but we need to apply methods such as **bag-of-visual-words** to take the multiple feature vectors extracted from an image and condense them into a single feature vector.

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

Feedback

← Previous Lesson (https://gurus.pyimagesearch.com/lessons/face-recognition-for-security/) Next Lesson → (https://gurus.pyimagesearch.com/lessons/color-channel-statistics/)

This upgrade offer will expire in **30 days, 18 hours**, so don't miss out — be sure to upgrade now.

Upgrade Your Membership! (https://gurus.pyimagesearch.com/register/pyimagesearch-gurus-instant-access-membership-fcff7b5e/)

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