3.3.1: Defining your image descriptor

Topic Progress: (https://gurus.pyimagesearch.com/topic/defining-your-image-descriptor/) (https://gurus.pyimagesearch.com/topic/indexing-your-dataset/) (https://gurus.pyimagesearch.com/topic/defining-your-similarity-metric/) (https://gurus.pyimagesearch.com/topic/searching/)

← Back to Lesson (https://gurus.pyimagesearch.com/lessons/the-4-steps-of-building-any-image-search-engine/)

Feedback

In our previous lesson, we learned **how to build a basic image search engine (https://gurus.pyimagesearch.com/lessons/your-first-image-search-engine/)** from start to finish. It was a lot of fun, and we learned a lot. We made use of OpenCV image descriptors, specifically multi-dimensional color histograms in the HSV color space. More importantly, we got to look at some real-world code to see how exactly an image search engine is built.

But let's back up a step.

During that lesson (along with our introductory lesson on **What Is Content-Based Image Retrieval? (https://gurus.pyimagesearch.com/lessons/what-is-content-based-image-retrieval/)**), I made mention of the four steps in building an image search engine:

1. **Defining your image descriptor:** The first step in building an image search engine is deciding what aspect of the image you want to describe. Are you interested in the color of the image? The shape of an object in the image? Or do you want to characterize texture?

2. **Feature extraction and indexing:** Now that you have chosen an image descriptor, you need to apply the descriptor to each image in your dataset and store the resulting feature vectors in persistent storage. Optionally, you can use a specialized data structure to facilitate faster access and comparisons of the feature vectors.
3. **Defining your similarity metric:** How are you going to determine the "similarity" of two images based on their feature vectors? The answer is to use a distance function/similarity metric — but you'll still need to choose which one to use.
4. **Search:** How does the actual "search" algorithm work? How are queries submitted to your image search engine?

We've seen these steps before — and it certainly won't be the last time I bring them up. Each of these steps is *critical* in building your own CBIR system. While our previous lesson was a good "let's get our hands dirty and write some code" type of article, the next few lessons are going to be a bit higher level and will frame the mindset you'll need to build CBIR systems. But if you plan on building an image search engine of your own, **these are four steps you need to understand**.

Today, we are going to focus on only the first step: **Defining your image descriptor**. We will explore the remaining steps in later lessons of this module.

Objectives:

In this lesson, we will frame our mindset to assess and choose an appropriate image descriptor when building a CBIR system to solve a particular problem.

Defining your image descriptor

In our **simple UKBench image search engine (https://gurus.pyimagesearch.com/lessons/your-first-image-search-engine/)**, we used a 3D color histogram in the HSV color space to characterize the color distribution of each image. We introduced ***locality*** into our color descriptor by computing a separate histogram for the top-left- top-right, bottom-left, bottom-right, and center regions of the images, respectively.

For our UKBench dataset, the 3D color histogram was a good choice. The objects/scenes utilized in the dataset had relatively *different* color distributions from each other, while related images had more *similar* color distributions — thus, this made it easier for color histograms to return relevant results.

Of course, color histograms are not the only image descriptor we can use. We can also utilize methods to describe both the texture and shape of objects in an image.

Color

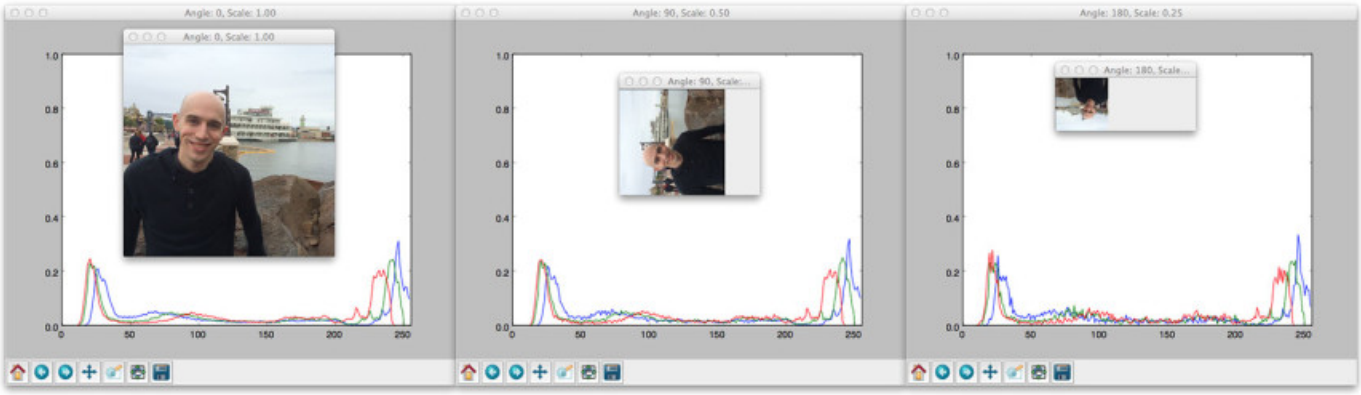
As we've already seen, color is the most basic aspect of an image to describe, and arguably the most computationally simple.

We have already used **simple color statistics (https://gurus.pyimagesearch.com/lessons/color-channel-statistics/)** in this course to characterize the contents of an image. We have also used color histograms to **cluster a small set of vacation photos (https://gurus.pyimagesearch.com/lessons/color-histograms/)**. We then extended color histograms to encode local regions of an image in our **CBIR example (https://gurus.pyimagesearch.com/lessons/your-first-image-search-engine/)**.

One benefit of using simple color methods is that we can easily obtain image size (scale) and orientation (how the image is rotated) invariance.

How is this possible?

Well, let's take a look at the following image at varying scales and orientations and the resulting histograms extracted from each image:



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

FIGURE 1: NO MATTER HOW WE ROTATE AND SCALE THE IMAGE, WE STILL HAVE THE SAME HISTOGRAM.

As you can see, we rotate and resize the image with a varying number of angles and scaling factors. The number of bins is plotted along the *x*-axis and the percentage of pixels placed in each bin on the *y*-axis.

In each case, the histogram is identical, thus demonstrating that the color histogram does not change as the image is scaled and rotated.

Rotation and scale invariance of an image descriptor are both desirable properties of an image search engine. If a user submits a query image to our image search engine, the system should find similar images, *regardless* of how the query image is resized or rotated. When a descriptor is robust to changes such as rotation and scale, we call them *invariants* due to the fact that the descriptor is *invariant* (i.e. does not change) even as the image is rotated and scaled. For more information in invariants, be sure to read through the **What is image classification? (https://gurus.pyimagesearch.com/topic/what-is-image-classification/)** lesson.

Texture

Texture tries to model feel, appearance, and the overall tactile quality of an object in an image; however, texture tends to be difficult to represent. For example, how do we construct an image descriptor that can describe the scales of a T-Rex as *"rough"* or *"coarse"*?

Most methods trying to model texture examine the *grayscale* image, rather than the color image. If we use a grayscale image, we simply have an *N x M* matrix of pixel intensities. We can examine pairs of these pixels and then construct a distribution of how often such pairs occur within *X* pixels of each other. This type of distribution is called a Gray-Level Co-occurrence Matrix (GLCM) and is exactly what the **Haralick texture descriptor (https://gurus.pyimagesearch.com/lessons/haralick-texture/)** uses to characterize texture.

We can also rely on methods such as **Local Binary Patterns (https://gurus.pyimagesearch.com/lessons/local-binary-patterns/)** which not only models the *texture*, but also the *pattern* the texture takes.

While not often thought of as a texture descriptor (normally it's used for rigid shape description), **Histogram of Oriented Gradients (https://gurus.pyimagesearch.com/lessons/histogram-of-oriented-gradients/)** can be used as a simple texture extractor as well. Remember, the Histogram of Oriented Gradients descriptors operates on the gradient magnitude and orientation representations of an image — in some cases you may find that these gradients can be leveraged as texture descriptors.

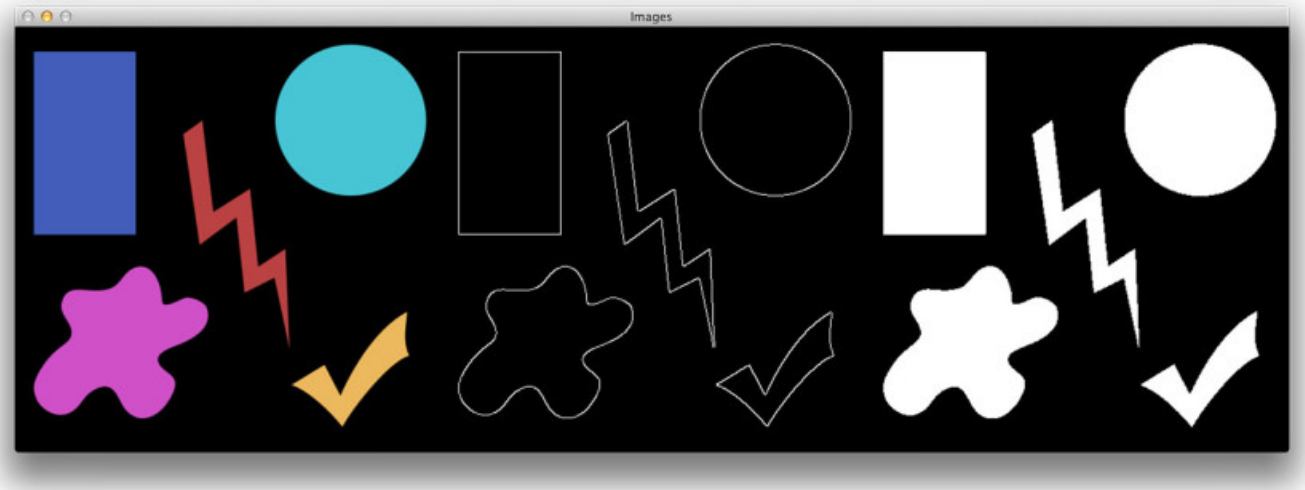
Finally, other texture descriptors exist as well, including taking the Fourier (https://en.wikipedia.org/wiki/Fourier_transform) and Wavelet (https://en.wikipedia.org/wiki/Wavelet_transform) transformation of a grayscale image and examining the coefficients after transformations. Fourier and Wavelet methods are perfectly acceptable texture descriptors; however, I often find their performance a bit lacking and only suitable in very specific (or controlled) circumstances. It's important to note; however, that Fourier and Wavelet methods can be quite computationally expensive without the aid of a GPU since many convolution operations are performed.

Shape

When discussing shape, I am not talking about the shape (or dimension, in terms of width and height) of the NumPy array that an image is represented as. Instead, I'm talking about the *shape/outline* of a particular object in an image.

When using a shape descriptor, our first step is normally to apply a segmentation or edge detection technique, allowing us to focus *strictly* on the contour of the shape(s) we want to describe. Then, once we have the contours, we can again compute statistical moments to represent the shapes.

Let's look at an example image:



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

FIGURE 2: THE IMAGE ON THE *LEFT* IS NOT SUITED FOR SHAPE DESCRIPTION. WE FIRST NEED TO EITHER PERFORM EDGE DETECTION *(MIDDLE)* OR EXAMINE THE COUNTERED, MASKED REGION *(RIGHT)*.

On the *left*, we have the full color image. Typically, we would not use this type of image to describe shape. Instead, we would convert the image to grayscale and perform edge detection (*center*) or utilize the mask (*right)*, i.e. the relevant part of the image we want to describe.

Popular shape descriptor methods include **Hu Moments (https://gurus.pyimagesearch.com/lessons/hu-moments/)** and **Zernike Moments (https://gurus.pyimagesearch.com/lessons/zernike-moments/)**, although for tasks such as object detection in images, the **Histogram of Oriented Gradients (https://gurus.pyimagesearch.com/lessons/histogram-of-oriented-gradients/)** descriptor combined with a bit of machine learning is by far the most used approach.

Ask yourself, *"What do I want to describe?"*

After running PyImageSearch over the past few years, one of the most common questions I get asked is, *"What image descriptor should I use to describe X?"* where *X* can be some object in an image, a set of vacation photos, or a biomedical imaging project.

In each and every case, the answer is always: *it depends*.

Often, you will need to leverage your intimate knowledge of the dataset contents to influence your image descriptor choice.

For example, if you took a lot of photos of scenic views on your vacation, then color histograms might be appropriate. However, if you are a geology nut, you have taken a lot of pictures of rock formations and different types of rocks and gravel, implying a more texture based approach would be more suited. You might also benefit from ***combining multiple image descriptors*** like we did in the **decision trees (https://gurus.pyimagesearch.com/topic/decision-trees/)lesson** (https://gurus.pyimagesearch.com/topic/decision-trees/) to characterize multiple qualities of an image.

When you're just getting started learning CBIR, these may not be questions you can answer immediately — and more times than not, your first choice won't give you the best results. And that's okay! Learning CBIR, and computer vision in general, is an iterative process. You try an approach, examine the results, and adjust accordingly. It will take many experiments and trial-and-error before you start noticing which image descriptors work best for which problems.

Summary

In this lesson, we discussed the first step in building an image search engine: *choosing an image descriptor*. Before a single line of code is written, we first need to examine our dataset and decide what aspects of the images we are going to describe. Is the color distribution of the image important when performing a

search? What about the texture of an object in an image? Or the shape? Or maybe we need to characterize all three?

Choosing a descriptor is just the first step. Next up, we'll explore how to apply our image descriptor(s) to each image in our dataset by applying *feature extraction*.

	Quizzes	Status
1	Defining Your Image Descriptor Quiz (https://gurus.pyimagesearch.com/quizzes/defining-your-image-descriptor-quiz/)	

Next Topic → (https://gurus.pyimagesearch.com/topic/indexing-your-dataset/)

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)