3.3.4: Searching

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/)

We are now at the final step of building an image search engine — accepting a query image and performing an actual search.

Let's take a second to review how we got here:

- **Step 1: Defining your image descriptor:** Before we even consider building an image search engine, we need to consider how we are going to represent and quantify our image using only a list of numbers (i.e. a feature vector). We explored three aspects of an image that can easily be described: *color*, *texture*, and *shape*. We can use one of these aspects, or many of them.
- **Step 2: Feature extraction and indexing:** Now that we have selected a descriptor, we can apply the descriptor to extract features from each and every image in our dataset. These extracted feature vectors are then written to disk for later use, where we can apply "indexing" to use specialized data structures to facilitate faster searches. Feature extraction and indexing are both tasks easily made parallel by utilizing multiple cores/processors on our machine.
- **Step 3: Defining your similarity metric:** In Step 1, we defined a method to extract features from an image. Now, we need to define a method to compare our feature vectors. A distance function should

accept two feature vectors and then return a value indicating how "similar" they are. Common choices for similarity functions include (but are certainly not limited to) the Euclidean, Manhattan, Cosine, and χ^2 Distances.

Objectives:

In this lesson, we will:

- Review the concept of a "query image".
- Detail the four steps required to perform a search.

The query

Before we can perform a search, we need a query.

The last time you went to Google, you typed in some keywords into the search box, right? The text you entered into the input form was your "query".

Google then took your query, analyzed it, compared it to their gigantic index of webpages, ranked them, and returned the most relevant webpages back to you.

Similarly, when we are building an image search engine, *we need a query image*.

Query images come in two flavors: an *internal* query image and an *external* query image. As the name suggests, **an internal query image already belongs in our index**. We have already analyzed it, extracted features from it, and stored its feature vector.

An **external query image** can be seen as the equivalent of typing text keywords into Google. We have never seen this query image before, and we can't make any assumptions about it. We simply apply our image descriptor, extract features, rank the images in our index based on similarity to the query, and return the most relevant results.

You may remember when we **built your first CBIR system (https://gurus.pyimagesearch.com/lessons/your-first-image-search-engine/)**, where we used images *already part of the UKBench dataset* as our queries.

Why did we do that — other than to visually inspect the output of your CBIR system, of course?

To answer that question, let's think back to our **similarity metrics (https://gurus.pyimagesearch.com/topic/3-3-3-defining-your-similarity-metric/)** for a second and assume that we are using the Euclidean distance. The Euclidean distance has a nice property called the Coincidence Axiom, implying that the function returns a value of *0* (indicating perfect similarity) if and only if the two feature vectors are identical.

If I were to search for an image already in my index, then the Euclidean distance between the two feature vectors would be *zero*, *implying perfect similarity*. This image would then be placed at the top of my search results since it is the most relevant. This makes sense and is the intended behavior:

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

FIGURE 1: WHEN SEARCHING FOR AN IMAGE THAT IS ALREADY INSIDE OUR INDEX, WE EXPECT IT TO BE THE #1 RESULT.

How strange it would be if I searched for an image already in my index and did not find it in the #1 result position! That would likely imply that there was a bug in my code somewhere, or I've made some very poor choices in image descriptors and similarity metrics:

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

FIGURE 2: IF THE IMAGE IS NOT THE #1 RESULT, THEN IT'S LIKELY THAT WE HAVE (1) A BUG IN OUR CODE, OR (2) WE ARE NOT PROPERLY USING OUR IMAGE DESCRIPTORS AND SIMILARITY METRICS.

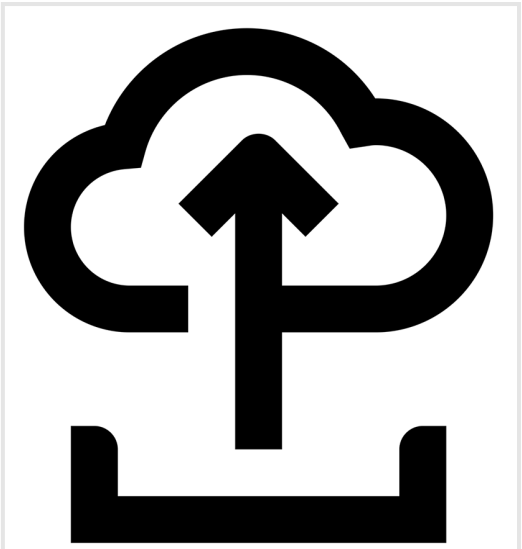
Overall, using an internal query image serves as a sanity check. It allows you to ensure that your image search engine is functioning as expected. Once you can confirm your image search engine is working properly, you can then accept external query images that are not already part of your index.

Feedback

The Search

So what's the process of actually performing a search? Check out the outline below:

1. Accept a query image from the user



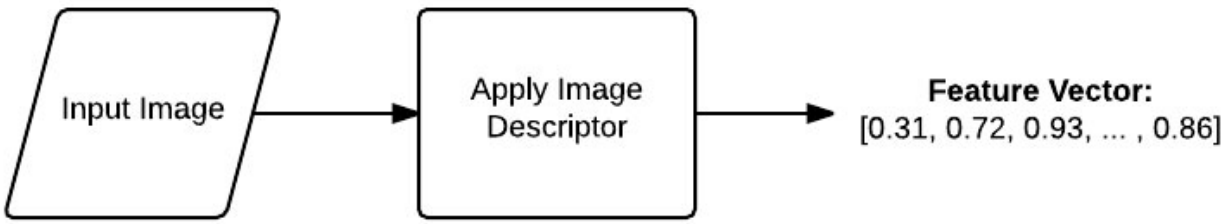
(https://gurus.pyimagesearch.com/wp-

content/uploads/2015/06/searching_upload_icon.png)

FIGURE 3: THE FIRST STEP TO PERFORMING A SEARCH IS TO ACCEPT A QUERY IMAGE. THIS IMAGE COULD BE UPLOADED FROM A COMPUTER, A MOBILE DEVICE, OR COULD ALREADY BE PART OF THE DATASET WE HAVE INDEXED.

A user could be uploading an image from their desktop or from their mobile device. As image search engines become more prevalent, I suspect that most queries will come from devices such as iPhones and Androids. It's simple and intuitive to snap a photo of a place, object, or something that interests you using your cellphone, and then have it automatically analyzed and relevant results returned.

2. Describe the query image



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

FIGURE 4: EXTRACT FEATURES FROM YOUR QUERY IMAGE IN THE SAME MANNER YOU EXTRACTED FEATURES FROM THE REST OF YOUR DATASET.

Feedback

Now that you have a query image, you need to describe it using the exact same image descriptor(s) as you did in the indexing phase. For example, if I used an RGB color histogram with 32 bins per channel when I indexed the images in my dataset, I am going to use the same 32 bin per channel histogram when describing my query image. This ensures I have a consistent representation of my images. After applying my image descriptor, I now have a feature vector for the query image.

3. Perform the search

To perform the most basic method of searching, you need to loop over *all* the feature vectors in your index. Then, you use your similarity metric to compare the feature vectors in your index to the feature vectors from your query. Your similarity metric will tell you how "similar" the two feature vectors are. Finally, sort your results by similarity.

If you would like to see this entire step in action, just head over to our **first CBIR lesson (https://gurus.pyimagesearch.com/lessons/your-first-image-search-engine/)**, where **Step 3** and **Step 4** show the relevant Python code.

Looping over your entire index may be feasible for small datasets. But if you have a large image dataset, like Google or TinEye, this simply isn't possible — you can't compute the distance between your query features and the billions of feature vectors already present in your dataset.

And that's exactly where specialized data structures such as kd-trees, approximate nearest neighbor algorithms, locality sensitive hashing, and random projection forests come into play. By using these algorithms, we can dramatically decrease our search time for a linear *O(N)* to a sub-linear *O(log N)*!

As we work through the rest of this module, I'll be demonstrating how you can apply these methods to perform speedy searches, even in very large datasets.

4. Display your results to the user

Now that we have a ranked list of relevant images, we need to display them to the user. This can be done using a simple web interface if the user is on a desktop, or we can display the images using some sort of app if they are on a mobile device. This step is pretty trivial in the overall context of building an image search engine, but you should still give thought to the user interface and how the user will interact with your image search engine.

Summary

So there you have it, the four steps of building an image search engine, from front to back:

1. Define your image descriptor.
2. Extract features/index your dataset.
3. Select a distance metric/similarity function.
4. Perform a search, rank the images in your index in terms of relevancy, and display the results to the user.

At this point, you should have a strong grasp on the steps required to build an image search engine. In the following lessons in our CBIR module, we'll start to dive into more advanced techniques (with lots of code).

We'll start off by extracting local invariant descriptors from images, such as SIFT, RootSIFT, and SURF. We'll then take these features and *cluster* them to form a *codebook* and *bag-of-visual-words* (BOVW). The BOVW model is the cornerstone of modern CBIR systems — *and it's one of my favorite sub-topics in CBIR!*

See you there!

Quizzes	Status
1	Searching Quiz (https://gurus.pyimagesearch.com/quizzes/searching-quiz/)

← Previous Topic (https://gurus.pyimagesearch.com/topic/defining-your-similarity-metric/)

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