

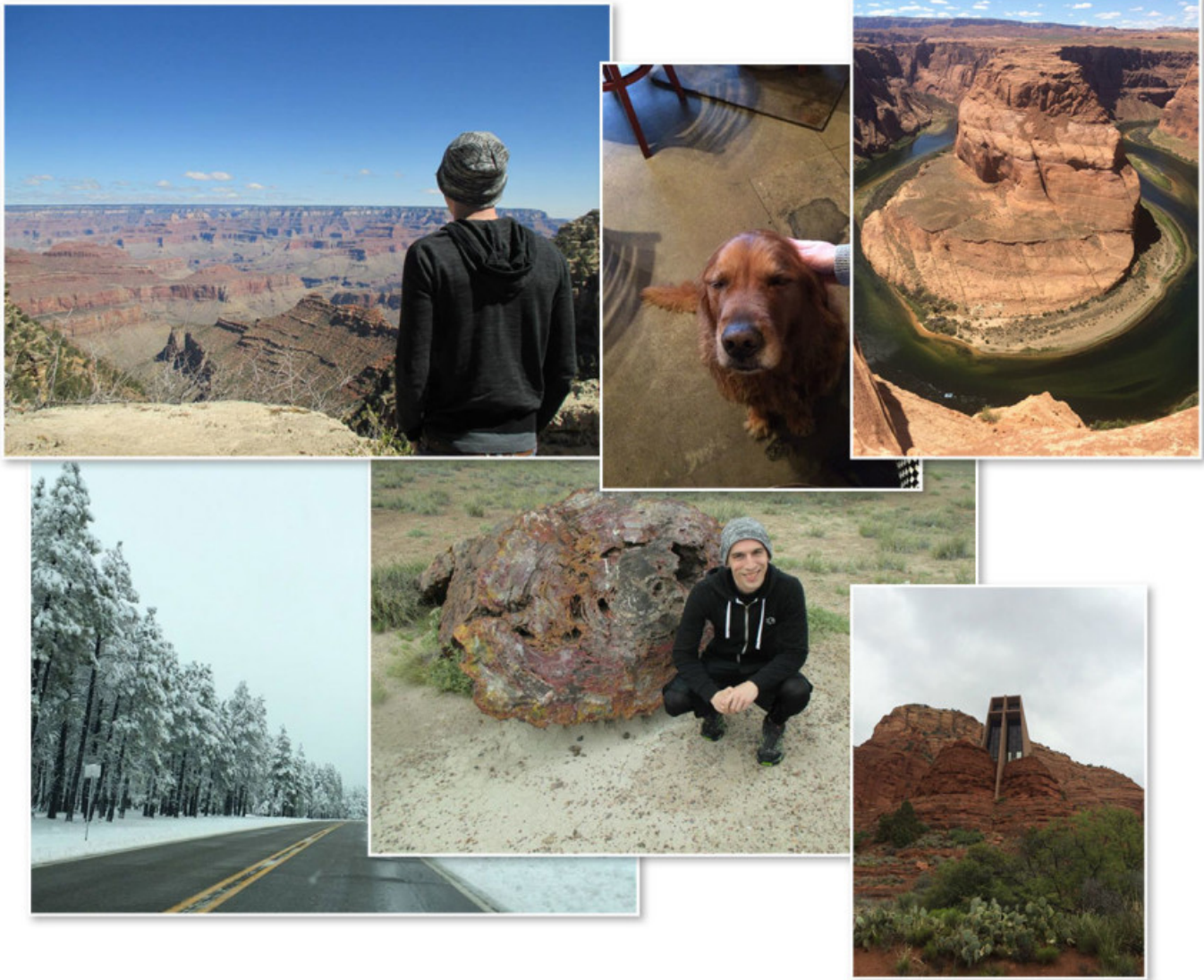
## PyImageSearch Gurus Course

[🏠 \(HTTPS://GURUS.PYIMAGESEARCH.COM\)](https://gurus.pyimagesearch.com/) >

# 3.1: What is Content-Based Image Retrieval?

So picture this.

You just got back from your family vacation visiting the National Parks on the west coast of the United States, including historical landmarks such as the Grand Canyon, Bryce Canyon, Lake Powell, Yosemite, Redwoods, and Yellowstone:



[https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/cbir\\_vacation.jpg](https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/cbir_vacation.jpg)

**FIGURE 1:** GOING ON VACATION LEAVES YOU WITH A LOT OF IMAGES TO ORGANIZE. IS THERE AN EFFICIENT WAY TO ORGANIZE THEM AND MAKE THEM SEARCHABLE?

Since you're an avid photographer, you brought along a whole bundle of SD cards. By the end of the trip you had accumulated exactly 8,732 *photos* from your vacation.

That's quite a lot.

Sure, you could sit there and import all those images into iPhoto and attempt to categorize them all. Maybe iPhoto is even smart enough to utilize the timestamps on your images to "roughly" categorize them. If you're lucky, you might even have the GPS coordinates embedded in the meta-data of the image.

But as we know, timestamps and GPS coordinates are not always accurate and reliable.

Instead, what we *really need* is a method to make these images *visually searchable*.

That is, given an input image of the Grand Canyon, we want to search our vacation photo dataset and return all relevant images — all without using textual tags or hints.

Is this possible?

You bet!

Image search engines encompass methods to make a dataset of images *visually searchable* using only the contents of the image. Academics call this *Content-Based Image Retrieval (CBIR)*. Throughout the rest of this course I will be using the terms “*image search engine*” and “*CBIR*” interchangeably.

At the core, image search engines rely on **extracting features from images** (<https://gurus.pyimagesearch.com/lessons/what-are-image-descriptors-feature-descriptors-and-feature-vectors/>), and then comparing the images for similarity based on the extracted feature vectors and distance metric. CBIR also includes methods to:

- Efficiently store features extracted of images.
- Scale the time it takes to perform a search *logarithmically* as the size of the image dataset increases *linearly*.
- Combine techniques from computer vision, information retrieval, and databases to build real-world images search engines that can be deployed online.

Feedback

## Objectives:

In this lesson we will cover:

- The three types of image search engines.
- Important terms such as *feature extraction*, *feature vector*, *indexing*, *distance metrics*, *querying*, and *result set*.
- The 4 steps of building any CBIR system.
- How to evaluate a CBIR system.
- The difference between CBIR and machine learning.

## What are CBIR/image search engines?

So you're probably wondering: what exactly is an image search engine?

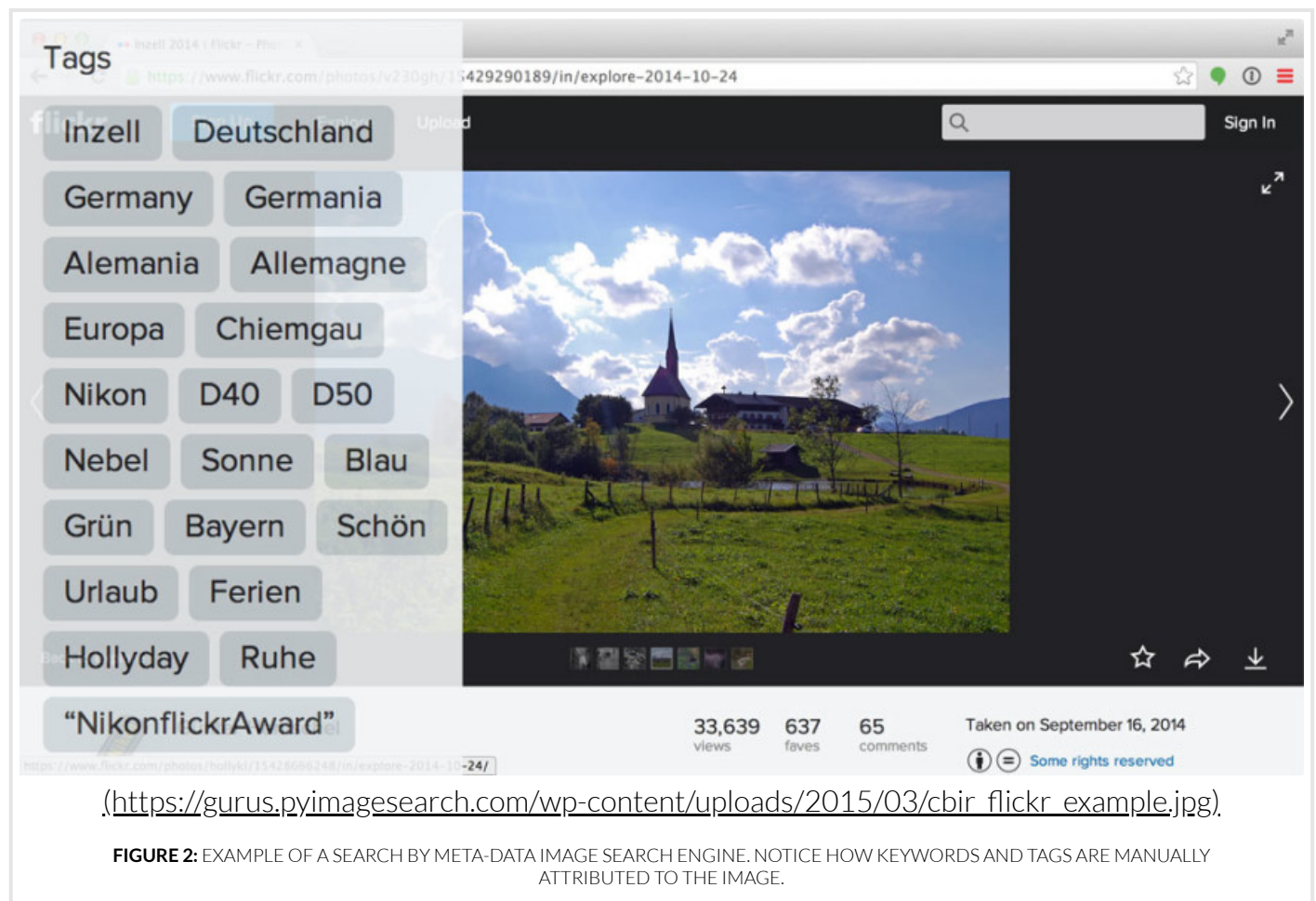
I mean, we're all familiar with text based search engines such as Google, Bing, and DuckDuckGo — you simply enter a few keywords related to the content you want to find (i.e., your “query”), and then your results are returned to you. But for image search engines, things work a little differently — you're not using *text* as your query, you are instead using an *image*.

Sounds pretty hard to do, right? I mean, how do you quantify the contents of an image to make it searchable?

We'll cover the answer to that question in a bit. But to start, let's learn a little more about image search engines.

In general, there tend to be three types of image search engines: **search by meta-data**, **search by example**, and a **hybrid approach** of the two.

## Search by Meta-Data



Searching by meta-data is only marginally different than your standard keyword-based search engines mentioned above. Search by meta-data systems rarely examine the contents of the image itself. Instead, they rely on textual clues such as (1) manual annotations and tagging performed by humans along with

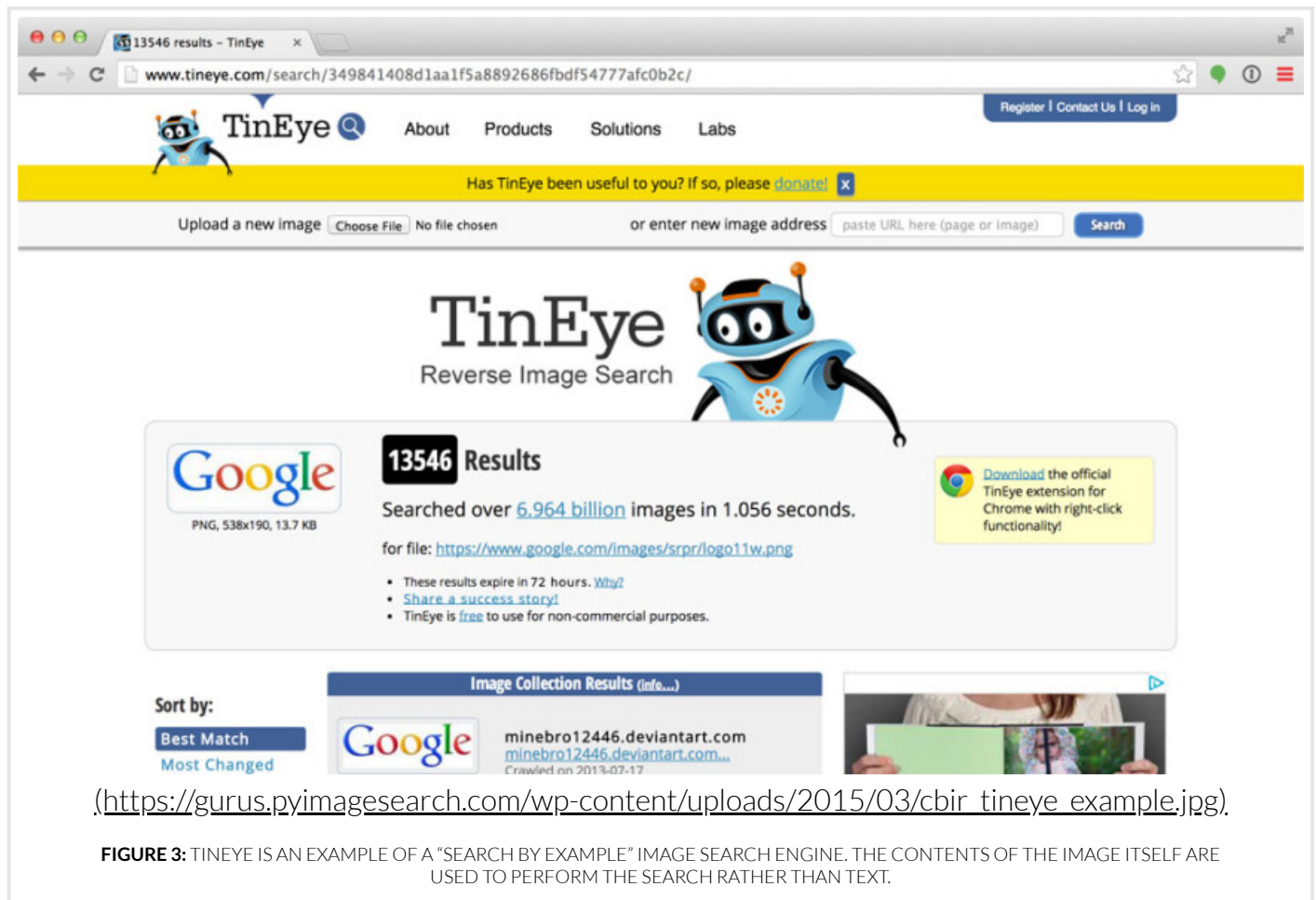
(2) automated contextual hints, such as the text that appears near the image on a webpage.

When a user performs a search on a search by meta-data system they provide a query, just like in a traditional text search engine, and then images that have similar tags or annotations are returned.

Again, when utilizing a search by meta-data system the actual image itself is rarely examined.

A great example of a Search by Meta-Data image search engine is Flickr (<http://www.flickr.com>). After uploading an image to Flickr you are presented with a text field to enter tags describing the contents of images you have uploaded. Flickr then takes these keywords, indexes them, and utilizes them to find and recommend other relevant images.

## Search by Example



The screenshot shows the TinEye website interface. At the top, there's a navigation bar with links for 'About', 'Products', 'Solutions', and 'Labs'. Below this is a yellow banner asking if the user has found TinEye useful and offering a 'donate!' button. The main search area has a 'Upload a new image' button and a text input field for a new image address. The TinEye logo and 'Reverse Image Search' text are prominently displayed. The search results section shows '13546 Results' for the file 'https://www.google.com/images/srpr/logo11w.png'. It mentions that the search was performed over 6.964 billion images in 1.056 seconds. Below the results, there's a 'Sort by' section with 'Best Match' and 'Most Changed' options. A table of 'Image Collection Results' is shown, with the first result being a Google logo from 'minebro12446.deviantart.com'. A small video player is visible on the right side of the results section.

([https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/cbir\\_tineye\\_example.jpg](https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/cbir_tineye_example.jpg)).

**FIGURE 3:** TINEYE IS AN EXAMPLE OF A "SEARCH BY EXAMPLE" IMAGE SEARCH ENGINE. THE CONTENTS OF THE IMAGE ITSELF ARE USED TO PERFORM THE SEARCH RATHER THAN TEXT.

Search by example systems, on the other hand, rely solely on the contents of the image — no keywords are assumed to be provided. The image is analyzed, quantified, and stored so that similar images are returned by the system during a search.

Image search engines that quantify the contents of an image are called **Content-Based Image Retrieval** (CBIR) systems. The term CBIR is commonly used in the academic literature, but in reality, it's simply a fancier way of saying "image search engine" with the added poignancy that the search engine is relying *strictly* on the contents of the image and not any textual annotations associated with the image.

A great example of a Search by Example system is TinEye (<https://www.tineye.com/>). TinEye is actually a reverse image search engine where you first provide a query image, and then TinEye returns near-identical matches of the same image, along with the webpage that the original image appeared on.

Take a look at the example image at the top of this section. Here I have uploaded an image of the Google logo. TinEye has examined the contents of the image and returned to me the 13,000+ webpages that the Google logo appears on after searching through an index of over 6 billion images.

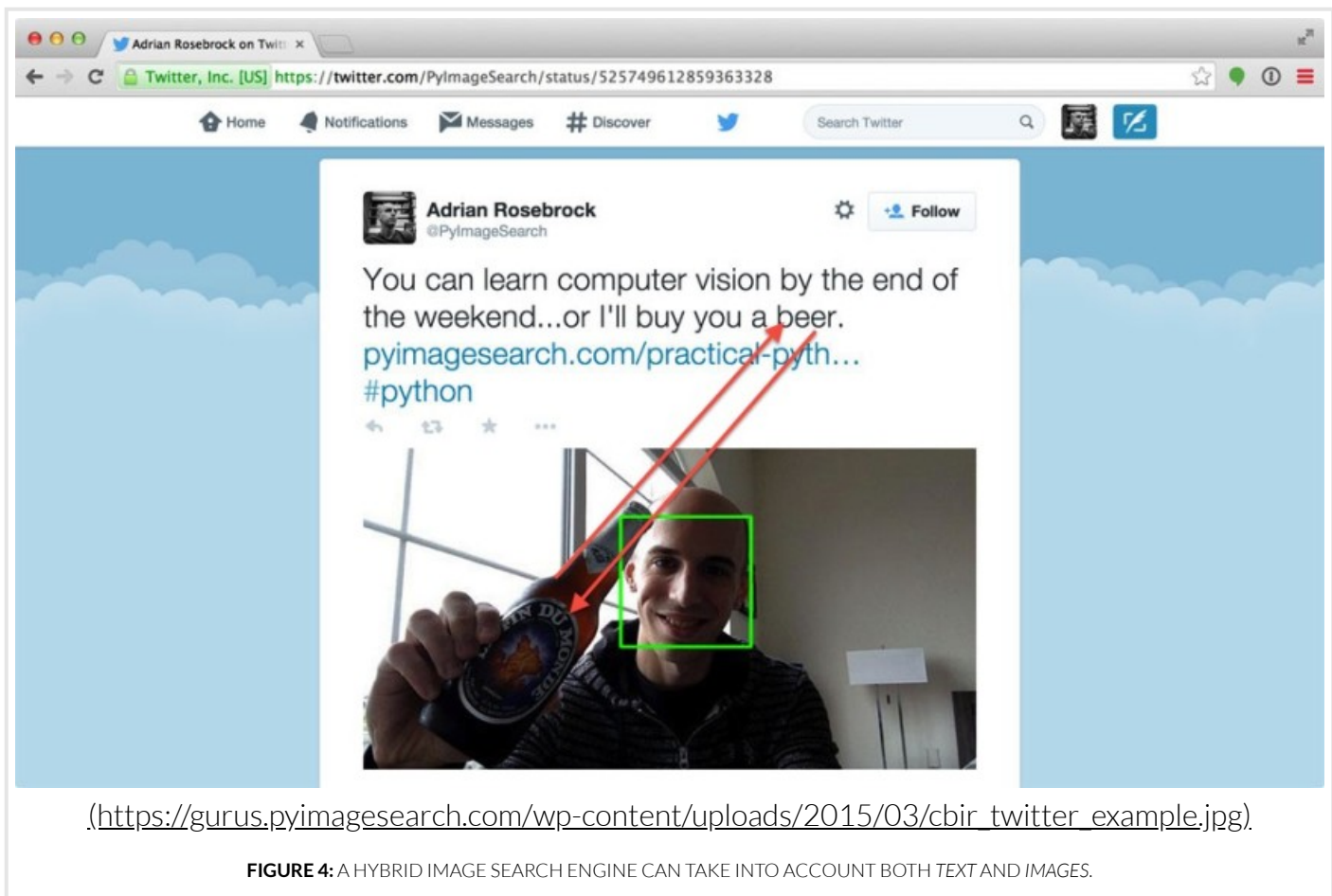
So consider this: *Are you going to manually label each of these six billion images in TinEye?* **Of course not.** That would take an army of employees and would be extremely costly.

Instead, you utilize some sort of algorithm to extract "features" (i.e., a list of numbers to quantify and abstractly represent the image) from the image itself. Then, when a user submits a query image, you extract features from the query image, compare them to your database of features, and try to find similar images.

Again, it's important to reinforce the point that Search by Example systems rely *strictly* on the contents of the image. These types of systems tend to be extremely hard to build and scale, but allow for a fully automated algorithm to govern the search — no human intervention is required.

## Hybrid Approach





Of course, there is a middle ground between the two — consider Twitter, for instance.

On Twitter you can upload photos to accompany your tweets. A hybrid approach would be to correlate the features extracted from the image with the text of the tweet. Using this approach you could build an image search engine that could take both contextual hints along with a Search by Example strategy.

Let's move on to defining some important terms that we'll use regularly when describing and building image search engines.

## Some Important Terms

Before we get too in-depth, let's take a little bit of time to define a few important terms.

The first term we need to define is *feature extraction*, which we cover in the **Image Descriptors** (<https://gurus.pyimagesearch.com/lessons/what-are-image-descriptors-feature-descriptors-and-feature-vectors/>), module of our course. 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*.

**Features**, on the other hand, are the output of our **image descriptor** or **feature descriptor**. When you put an image into an image descriptor, you will get **features** out the other end.

In the most basic terms, **features** (or **feature vectors**) are just a list of numbers used to abstractly represent and quantify images. Take a look at the example figure below:



([https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/cbir\\_describing\\_images.jpg](https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/cbir_describing_images.jpg)).

**FIGURE 5:** 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 WHICH IS USED TO QUANTIFY AND ABSTRACTLY REPRESENT THE CONTENTS OF THE IMAGE.

Here we are presented with an input image. We apply our image descriptor. And then our output is a list of features used to quantify the image.

We often use the terms *features* and *feature vectors* interchangeably.

Extracting features from images is inherently a parallelizable task. Given a dataset of images, our goal is to apply the *exact same image descriptor* to *each and every image* in the dataset. Thus, we can leverage multiple cores of our system to make the feature extraction stage run considerably faster. We could even go as far as to use big data tools such as Hadoop to speed up feature extraction even further.

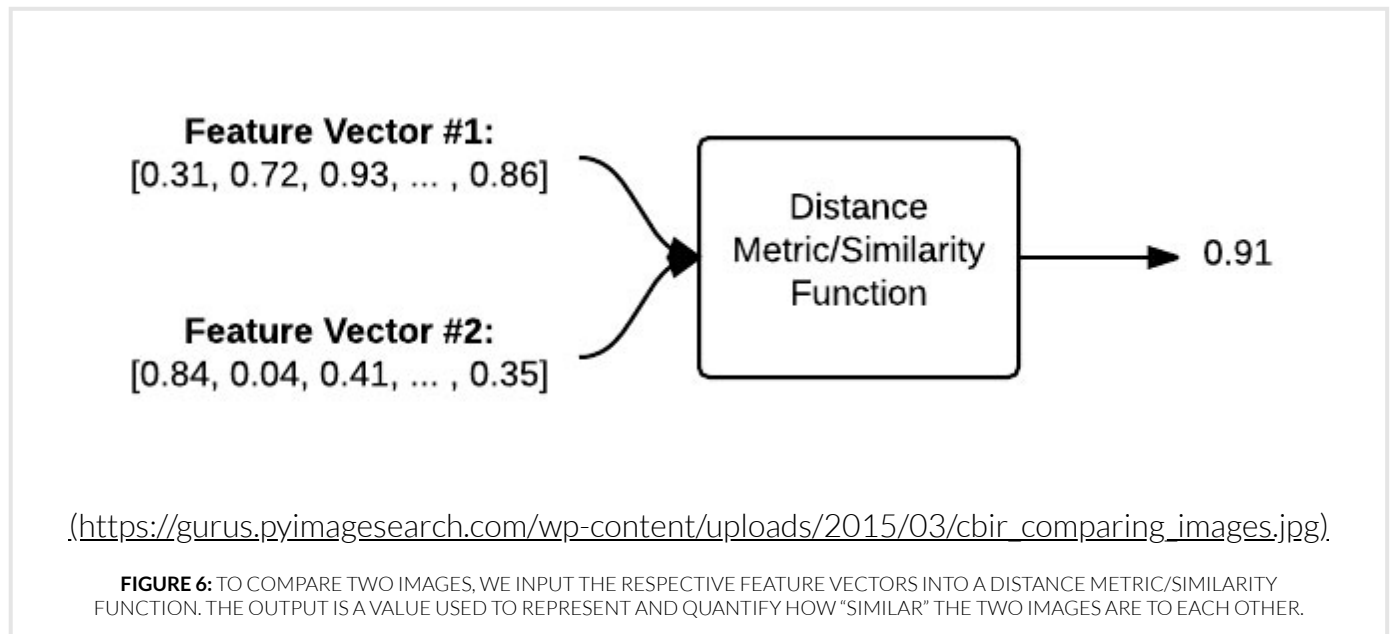
Regardless of whether we extract features from images serially, using multiple processes, or on a Hadoop cluster, when building an image search engine, we often have to **index our dataset of features**. Here we take the extracted features from our dataset and store the features in an efficient manner so that they can be compared for relevancy.

In some cases, indexing our dataset is as simple as storing our feature vectors in a .csv file. But for large-scale image search engines, we use specialized data structures and algorithms to make searches in an  $N$ -dimensional space run in sub-linear time.



Now that we have our feature vectors extracted and stored, we can compare pairs for relevance and similarity by using a **distance metric** or **similarity function**. Distance metrics and similarity functions take two feature vectors as inputs and then output a number that represents how “similar” the two feature vectors are.

The figure below visualizes the process of comparing two images:



The output of the distance function is a single floating point value used to represent the similarity between the two images.

Once we have our distance metric or similarity function in place, we can perform an actual search. The image we actually submit to our CBIR system (like typing text keywords into Google) is called our **query image** or simply **query**.

Our CBIR system takes our query image, extracts features from it, and uses our indexed dataset and distance metric to compare images for similarity strictly by utilizing the extracted feature vectors.

The **result set**, or the set of images that are relevant to the query, are then sorted and presented to the user.

If that sounds like a lot, don’t worry — it’s actually not too bad. As we’ll see in the next lesson where we build our first image search engine, we can apply *all* of these terms and steps in under 150 lines of code.

That said, let’s take our collection of terms and then use them to define the 4 steps of building any CBIR system.

# The 4 Steps of Any CBIR System

No matter what Content-Based Image Retrieval System you are building, they all can be boiled down into 4 distinct steps:

1. **Defining your image descriptor:** At this phase you need to decide 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 your dataset:** Now that you have your image descriptor defined, your job is to apply this image descriptor to each image in your dataset, extract features from these images, and write the features to storage (ex. CSV file, RDBMS, Redis, etc.) so they can later be compared for similarity. Furthermore, you need to consider if any specialized data structures are going to be used to facilitate faster searching.
3. **Defining your similarity metric:** We now have a collection of (perhaps index) feature vectors. But how are you going to compare them for similarity? Popular choices include the Euclidean distance, Cosine distance, and  $\chi^2$  distance, but the actual choice is highly dependent on (1) your dataset and (2) the types of features you extracted.
4. **Searching:** The final step is to perform an actual search. A user will submit a query image to your system (from an upload form or via a mobile app, for instance) and your job will be to (1) extract features from this query image and then (2) apply your similarity function to compare the query features to the features already indexed. From there, you simply return the most relevant results according to your similarity function.

Again, these are the most basic four steps of any CBIR system. As they become more complex and utilize different feature representations, the number of steps grow and you'll add a significant number of sub-steps to each step mentioned above. But for the time being, let's keep things simple and utilize just these four steps.

Let's take a look at a few graphics to make these high-level steps a little more concrete. The figure below details Steps 1 and 2:

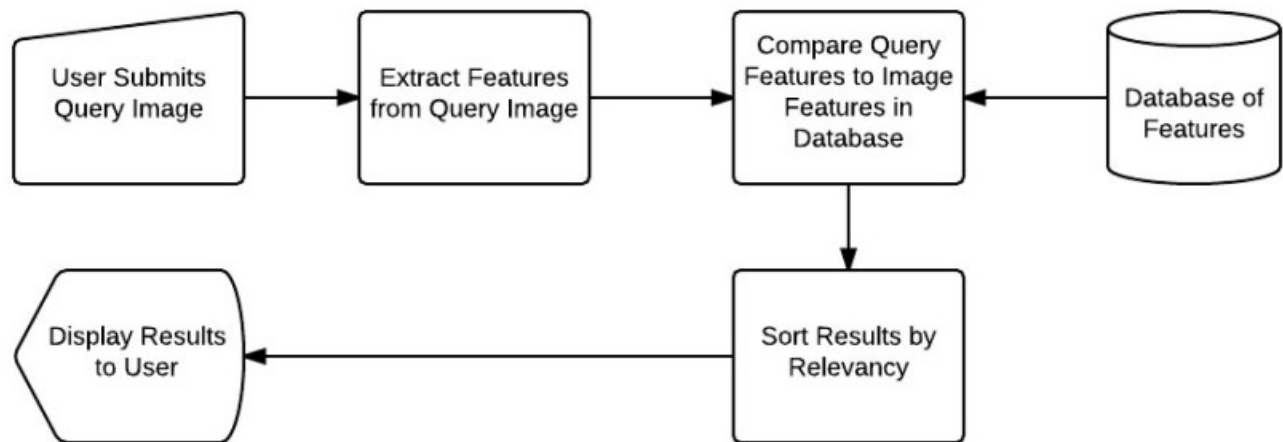


([https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/cbir\\_preprocessing\\_and\\_indexing.jpg](https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/cbir_preprocessing_and_indexing.jpg)).

**FIGURE 7:** A FLOWCHART REPRESENTING THE PROCESS OF EXTRACTING FEATURES FROM EACH IMAGE IN THE DATASET.

We start by taking our dataset of images, extracting features from each image, and then storing these features in a database.

We can then move on to performing a search (Steps 3 and 4):



([https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/cbir\\_searching.jpg](https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/cbir_searching.jpg)).

**FIGURE 8:** PERFORMING A SEARCH ON A CBIR SYSTEM. A USER SUBMITS A QUERY; THE QUERY IMAGE IS DESCRIBED; THE QUERY FEATURES ARE COMPARED TO EXISTING FEATURES IN THE DATABASE; RESULTS ARE SORTED BY RELEVANCY AND THEN PRESENTED TO THE USER.

First, a user must submit a query image to our image search engine. We then take the query image and extract features from it. These “query features” are then compared to the features of the images we already indexed in our dataset. Finally, the results are then sorted by relevancy and presented to the user.

## Evaluating a CBIR system

To evaluate a particular CBIR system, researchers and academics use three important metrics: **precision**, **recall**, and **f-score**:

**Precision** is defined as the ***fraction of retrieved images that are relevant to the query***:

$$precision = \frac{\# \text{ of relevant images retrieved}}{\text{total } \# \text{ of images retrieved from database}}$$

We then have **recall**, which is the ***fraction of relevant images returned by the query***:

$$recall = \frac{\# \text{ of relevant images retrieved}}{\text{total } \# \text{ of relevant images in database}}$$

Finally, we can combine both **precision** and **recall** to form the **f-score**, sometimes called the **F1-score** or **f-measure**:

$$f\text{-score} = 2 \times \frac{precision \times recall}{precision + recall}$$

Obviously, the assumption when using these metrics is that we **know** how many relevant images there are to a particular query in our dataset. This works well in academic and research datasets where CBIR systems must be benchmarked evaluated and measured against a standard dataset with known ground-truth result sets.

But in many real-world CBIR problems, it may not be possible for us to know *exactly* how many relevant images that are to a particular query. Instead, we must (1) visually investigate the result sets from particular queries, (2) utilize our general intuition when deciding what is working well and what is not, and (3) create small “test sets” for particular images that we can use to measure the accuracy of our CBIR system.

## How are CBIR and machine learning/image classification different?

A very common query I get asked is how CBIR and image classification differ — and that’s a very fair question to ask. Both fields of study involve extracting features from images. Both fields include some notion of “accuracy” and “relevancy.” Both methods require datasets of images.

Machine learning encompasses methodologies to make computers do intelligent human tasks such as prediction, classification, recognition, etc. Furthermore, machine learning governs algorithms to make computers capable of performing these intelligent tasks *without being explicitly programmed*.

On the other hand, CBIR does indeed leverage some machine learning techniques — namely dimensionality reduction and clustering, but CBIR systems do not perform any actual learning.

Perhaps the **primary** difference is that CBIR does not **directly** try to understand and interpret the contents of an image. Instead, CBIR systems rely on:

- The quantification of the image through the extraction of feature vectors.
- Comparison of the feature vectors — images with similar feature vectors are assumed to have similar visual contents.

Based on these two components, image search engines are able to compare a query to a dataset of images and return the most relevant results without having to actually “know” the contents of the image.

In machine learning and image classification, being able to learn and understand the contents of an image requires some notion of a *training set* — a set of labeled data used to teach the computer what each visual object in the dataset looks like.

CBIR systems, on the other hand, require no labeled data. They simply take the image dataset, extract features from each of the images, and make the dataset visually searchable. In some ways, you can think of a CBIR system as a “dumb” image classifier that has no notion of labels to make itself more intelligent — it simply relies on (1) the features extracted from the images and (2) the similarity function used to give meaningful results back to the user.

## Summary

In this lesson we learned about image search engines, commonly called Content-Based Image Retrieval (or simply CBIR) in the academic and research world.

We started by exploring the three different types of image search engines: **search by meta-data**, **search by example**, and a **hybrid approach** that combines the previous two methods together.

We also learned some important terms such as *feature extraction*, or the process of quantifying and extracting *feature vectors* from an image to abstractly represent the image’s contents. In order to make our dataset of images visually searchable, we will also want to *index* our dataset so that special algorithms can make our search run in sub-linear time. In order to compare our images for similarity, we’ll need a *distance metric* or *similarity function*. A distance metric/similarity function takes two feature vectors and returns a value indicating how *similar* the two feature vectors are. Finally, a user can submit a *query image* to our dataset, where we extract features from the query and compare it to our pre-index dataset of feature vectors. The *result set*, or set of images relevant to the query, is accumulated and presented to the user.

These terms are then collected into the 4 steps of building any image search engine, namely:

1. Defining your image descriptor.
2. Extracting features from our dataset using your image descriptor, followed by indexing the feature vectors.
3. Defining your similarity function used to compare two feature vectors.
4. Performing an actual search by submitting a query and receiving a result set back.

Once we have our result set for a particular query, we can use the *precision*, *recall*, and *f-measure* to measure the accuracy of our CBIR system — provided that we know the number of relevant images to a particular query ahead of time. In reality, we may not know the number of relevant images for a particular query. In that case, we need to visually investigate our results and use our general intuition to ensure that the CBIR system is returning accuracy results. We should consider developing small testing sets of data that we can evaluate precision, recall, and f-measure for.

Finally, it's important to note that CBIR is indeed different than machine learning and image classification. The main difference is that CBIR systems do not actually “interpret” and “understand” the contents of an image. Instead, they simply rely on the discrimination power of the features extracted and the distance metric to return relevant results. Machine learning systems, on the other hand, leverage datasets of images *and* labels associated with each of the images to “learn” what each of the classes looks like. For more information on image classification and machine learning, be sure to take a look at the **[image classification module \(https://gurus.pyimagesearch.com/lessons/a-high-level-overview-of-image-classification/\)](https://gurus.pyimagesearch.com/lessons/a-high-level-overview-of-image-classification/)**.

So now that we have a general high-level overview of CBIR systems, we can move on to our next lesson where we build our first image search engine.

Quizzes		Status
1	What is Content-Based Image Retrieval Quiz ( <a href="https://gurus.pyimagesearch.com/quizzes/what-is-content-based-image-retrieval-quiz/">https://gurus.pyimagesearch.com/quizzes/what-is-content-based-image-retrieval-quiz/</a> )	



[← Previous Lesson \(https://gurus.pyimagesearch.com/lessons/tips-on-training-your-own-object-detectors/\)](https://gurus.pyimagesearch.com/lessons/tips-on-training-your-own-object-detectors/). [Next Lesson → \(https://gurus.pyimagesearch.com/lessons/your-first-image-search-engine/\)](https://gurus.pyimagesearch.com/lessons/your-first-image-search-engine/).

## 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! \(/pyimagesearch-gurus-course/\)](/pyimagesearch-gurus-course/)

## Resources & Links

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

## Your Account

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

🔍 Search

