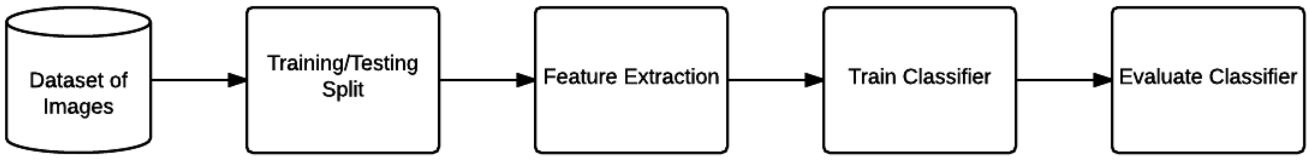
PyImageSearch Gurus Course

4.2: The image classification pipeline



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/image_classification_flowchart.png)

Based on our previous two lessons on **image classification (https://gurus.pyimagesearch.com/topic/what-is-image-classification/)** and the **types of learning algorithms (https://gurus.pyimagesearch.com/topic/types-of-learning/)**, you might be starting to feel a bit overwhelmed.

I mean, how in the world do you use these algorithms to identify various categories of images?! Is this some sort of black art?

Actually, it's not — and it's fairly straightforward *once you understand the process.*

In this lesson, we'll review the pipeline you'll use when building *any* image classifier. By the time you reach the bottom of this page, you'll have a clear understanding of how image classification algorithms work and the steps that go into building your own image classifier.

Objectives:

The primary objective of this lesson is to understand the **five steps of the image classification pipeline:**

- **Step 1:** Structuring your initial dataset.
- **Step 2:** Splitting the dataset into two (optionally three) parts.
- **Step 3:** Extracting features.
- **Step 4:** Training your classification model.
- **Step 5:** Evaluating your classifier.

In practice, we can often swap the order of **Step 2** and **Step 3**. It doesn't matter if we split our dataset first, or if we extract features from our dataset and then split the extract features. As long as both steps are performed, the end result is the same.

The image classification pipeline

Before we get into anything complicated, let's start this lesson off with something that we're all (most likely) familiar with: *the Fibonacci sequence*.

The Fibonacci sequence is a series of numbers where the next number of the sequence is found by summing the two integers before it.

For example, given the sequence *0, 1, 1*, the next number is found by adding *1 + 1 = 2*. Similarly, given *0, 1, 1, 2*, the next integer in the sequence is *1 + 2 = 3*. Following that pattern, the first handful of numbers in the sequence are as follows:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

Of course, we can also define this as an (extremely unoptimized) Python function using recursion:

A Python function for the Fibonacci sequence   Python

```python
>>> def fib(n):
...     if n == 0:
...             return 0
...     elif n == 1:
...             return 1
...     else:
...             return fib(n-1) + fib(n-2)
...
>>>
```

Using this code, we can compute the *n-th* number in the sequence by supplying a value of `n` to the `fib` function. For example, let's compute the 7th number in the Fibonacci sequence:

The 7th number of the Fibonacci sequence   Python

```python
>>> fib(7)
13
```

And the 13th number:

The 13th number of the Fibonacci sequence Python
```
>>> fib(13)
233
```

And finally the 35th number:

The 35th number of the Fibonacci sequence Python
```
>>> fib(35)
9227465
```

As you can see, the Fibonacci sequence is straightforward and is an example of a family of functions that:

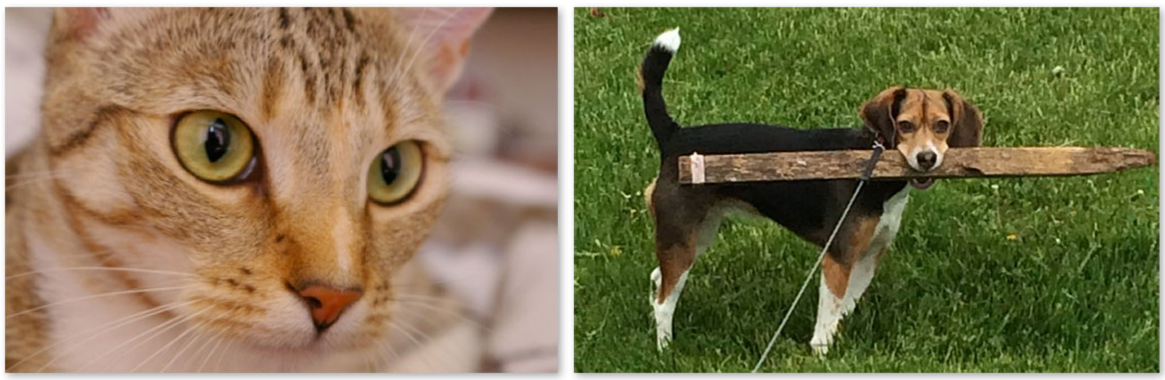
- Accepts an input, returns an output.
- The process is well defined.
- The output is easily verifiable for correctness.
- Lends itself well to code coverage and test suites.

In general, you've probably written thousands upon thousands of procedural functions like these in your life. Whether you're computing a Fibonacci sequence, pulling data from a database, or calculating the mean and standard deviation of a list of numbers, these functions are all well defined and easily verifiable for correctness.

Unfortunately, this is not the case for machine learning and image classification!

Feedback

Recall our **introduction to image classification (https://gurus.pyimagesearch.com/topic/what-is-image-classification/)** where we looked at these pictures of a cat and a dog:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/05/image_classification_cat_and_dog.jpg)

FIGURE 1: HOW WOULD YOU WRITE CODE TO RECOGNIZE THE DIFFERENCE BETWEEN THESE TWO IMAGES?

Now, imagine trying to write a procedural function that can not only tell the difference between *these two photos*, but ***any*** photo of a cat and a dog. How would you go about doing that? Would you check individual pixel values at various *(x, y)*-coordinates? Write hundreds of "if/else" statements? And how in the world would you maintain and verify the correctness of such a massive rule-based system?

The short answer is: **you don't.**

Unlike coding up an algorithm to compute the Fibonacci sequence or sort a list of numbers, it's not exactly intuitive or obvious how to create an algorithm to tell the difference between pictures of cats and dogs.

Therefore, instead of trying to code up a rule-based system to describe what each category "looks like", we can instead take a ***data driven approach*** by supplying *examples* of what each category looks like and then *teaching* our algorithm to recognize the difference between the categories using only the examples.

We call these examples our *training dataset* of labeled images, where each data point in our training dataset consists of:

- An image
- The label/category (i.e. dog, cat, cow, pig, etc.) of the image

Again, it's important that each of these images have a label associated with it because our **supervised learning algorithm (https://gurus.pyimagesearch.com/topic/types-of-learning/)** will need to see these labels to "teach itself" how to recognize each category.

Feedback

Keeping this in mind, let's go ahead and work through the **five steps to building an image classifier**.

Step 1: Gathering your dataset

As I mentioned above, the first thing we need is our initial dataset. We need the *images themselves* as well as the *labels* associated with each image. These labels could come from a finite set of categories, such as: **categories = {cat, cow, dog, horse, wolf}**.

Furthermore, the number of images for each category should be fairly uniform (i.e. the same). If we have twice the number of cat images than dog images, and five times the number of horse images than cat images, then our classifier will become naturally biased to "overfitting" into these heavily-represented categories.

In order to fix this problem, we normally sample our dataset so that each category is represented equally.

Step 2: Splitting our dataset

Now that we have our initial dataset, we need to split it into two parts: a *training set* and a *testing set*.

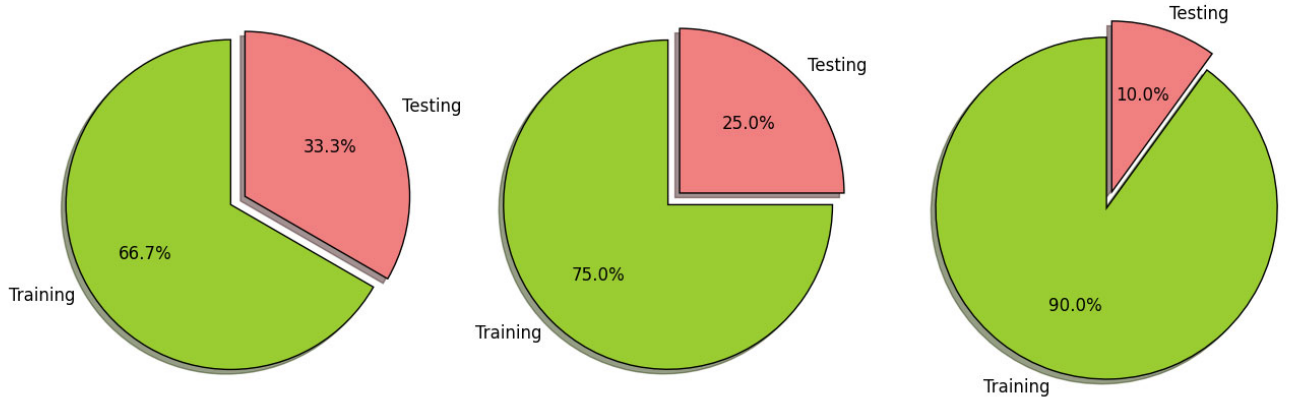
A *training set* is used by our classifier to "learn" what each category looks like by making predictions on the input data and then corrected when the predictions are wrong.

After the classifier has been trained, we can then evaluate its performance on a *testing set*.

It's extremely important that the training set and testing set are independent of each other and do not overlap!

If you use your testing set as part of your training data then your classifier has an unfair advantage, since it has already seen the testing examples before and "learned" from them. Instead, you must keep this testing set entirely separate from your training process and use it ***only to evaluate your classifier.***

Common split sizes for training and testing include:



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

FIGURE 2: EXAMPLES OF COMMON TRAINING AND TESTING DATA SPLITS.

These data splits make sense, **but what happens if you have parameters to tune?**

Algorithms such as Logistic Regression, Support Vector Machines, and Random Forest Classifiers have many knobs and levers that need to be tuned and dialed to obtain optimal performance. We call these types of parameters ***hyperparameters***.

In practice, we need to test a bunch of these hyperparameters and find the ones that work best. You might be tempted to use your testing data to tweak these values — ***but again, this is a major no-no!***

The test set is *only* used in evaluating the performance of your classifier.

Feedback

Instead, you create a *third* data split called the ***validation set***. This split of the data normally comes from the training data and is used as a "fake test data", so we can tune our hyperparameters. Only after we have tuned our hyperparameters using the validation set do we move on to collecting final accuracy results on the test data. **We normally allocate roughly 10-20% of the training data for validation data.**

If splitting your data into chunks sounds complicated, it's actually not. As we'll find out in our next lesson on using a **k-Nearest Neighbor classifier (https://gurus.pyimagesearch.com/lessons/k-nearest-neighbor-classification/)**, it's actually quite simple and can be accomplished only with a **single line of code** thanks to the scikit-learn library.

Step 3: Feature extraction

Now that we have our data splits defined, we need to extract features to abstractly quantify and represent each image. Common choices of features include:

- Color Histograms (https://gurus.pyimagesearch.com/lessons/color-histograms/)
- Histogram of Oriented Gradients (https://gurus.pyimagesearch.com/lessons/histogram-of-oriented-gradients/)
- Local Binary Patterns (https://gurus.pyimagesearch.com/lessons/local-binary-patterns/)
- **…and pretty much every other descriptor we cover in this course. (https://gurus.pyimagesearch.com/lessons/what-are-image-descriptors-feature-descriptors-and-feature-vectors/)**

In some cases, we may even use the raw pixel intensities of the images themselves as feature vectors, although this is quite uncommon and is only done in very specific circumstances, like in our next lesson on the **k-Nearest Neighbor classifier (https://gurus.pyimagesearch.com/lessons/k-nearest-neighbor-classification/)**.

As I mentioned at the top of this lesson, we can easily swap the order of Step 2 and Step 3. In some cases, it may be easier for us to first split our dataset and then perform feature extraction. In others, we may want to extract features from our images first and then split the dataset — it all depends on the problem and what makes the most logical sense. Either way, swapping these steps will have no impact on your final classifier as long as both steps are performed.

Step 4: Training your classifier

Given the feature vectors associated for the training data, we can now train our classifier. The goal here is for our classifier to "learn" how to recognize each of the categories in our label data. When the classifier makes a mistake, it learns from this mistake and improves itself.

So how does the actual "learning" work?

Well, that depends on each individual algorithm. Support Vector Machines (https://gurus.pyimagesearch.com/topic/support-vector-machines/) work in high-dimensional spaces seeking an optimal hyper-plane to separate the categories. Decision Trees (https://gurus.pyimagesearch.com/topic/decision-trees/) and Random Forest Classifiers (https://gurus.pyimagesearch.com/topic/random-forests/) look for optimal splits in the data based on entropy. Meanwhile, algorithms such as k-Nearest Neighbor (https://gurus.pyimagesearch.com/lessons/k-nearest-neighbor-classification/) perform no actual "learning" because they simply rely on the distance between feature vectors in an *n*-dimensional space to make predictions.

We'll be covering many machine learning models inside this course, so don't feel the need to perform a deep dive into machine learning just yet. Instead, just simply understand that the actual training and learning process varies dramatically on an algorithm-to-algorithm basis, but they all share the common goal of attempting to learn and distinguish between categories.

Step 5: Evaluation

Last, we need to evaluate our trained classifier. For each of the feature vectors in our testing set, we present them to our classifier and ask it to *predict* what it thinks the label of image is. We then tabulate the predictions of the classifier for each point in the testing set.

Feedback

Finally, these *classifier predictions* are compared to the *ground-truth* label from our testing set. The ground-truth labels represent what the category *actually* is. From there, we can compute the number of predictions our classifier got right and compute aggregate reports such as **precision, recall, and f-measure**, which are used to quantify the performance of our classifier as a whole.

Summary

In this lesson we reviewed the five steps of building an image classifier, which include: *gathering your dataset*; *splitting your dataset into a training and a testing set (and an optional validation set); extracting feature vectors from our data; training your classifier;* and finally *evaluating your model*.

Each of these steps is equally important and can have a dramatic impact on the total accuracy of your classifier, so it's important to consider each step carefully. That said, I think it's extremely helpful to see how each of these steps is done in the real world.

Coming up, we'll do exactly that and get our hands dirty using the **k-Nearest Neighbor (https://gurus.pyimagesearch.com/lessons/k-nearest-neighbor-classification/)** classifier to recognize and identify 10 different types of image categories, including *airplanes*, *cards*, *birds*, *cats*, *ships, and more!*

Quizzes		Status
1	The Image Classification Pipeline Quiz (https://gurus.pyimagesearch.com/quizzes/the-image-classification-pipeline-quiz/)	

← Previous Lesson (https://gurus.pyimagesearch.com/lessons/a-high-level-overview-of-image-classification/) Next Lesson → (https://gurus.pyimagesearch.com/lessons/k-nearest-neighbor-classification/)

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)

Feedback

Search