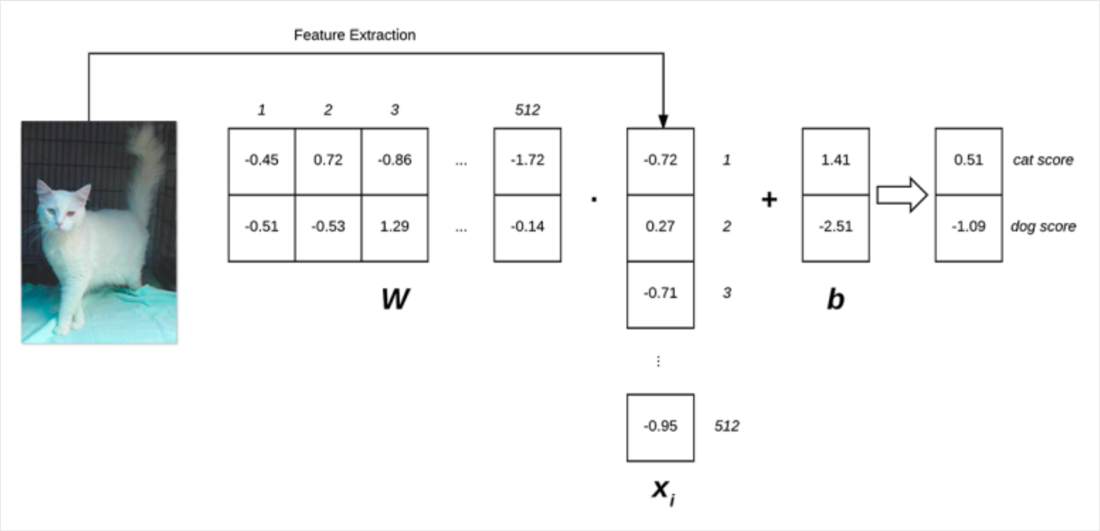
Navigation

An intro to linear classification with Python

by **Adrian Rosebrock** on August 22, 2016 in **Machine Learning**, **Tutorials**

50



Over the past few weeks, we've started to learn more and more about machine learning and the role it plays in *computer vision*, *image classification*, and *deep learning*.

We've seen how Convolutional Neural Networks (CNNs) such as LetNet can be used to classify handwritten digits from the MNIST dataset. We've applied the k-NN algorithm to classify whether or not an image contains a dog or a cat. And we've learned how to apply hyperparameter tuning to optimize our model to obtain higher classification accuracy.

However, there is another *very important* machine learning algorithm we have yet to explore — one that can be built upon and extended *naturally* to Neural Networks and Convolutional Neural Networks.

What is the algorithm?

It's a simple linear classifier — and while it's a straightforward algorithm, it's considered the cornerstone building block of more advanced machine learning and deep learning algorithms.

Keep reading to learn more about linear classifiers and how they can be applied to image classification.

Looking for the source code to this post?
Jump right to the downloads section.

[redacted]

An intro to linear classification with Python

The first half of this tutorial focuses on the basic theory and mathematics surrounding *linear classification* — and in general — *parameterized classification algorithms* that actually "learn" from their training data.

From there, I provide an actual linear classification implementation and example using the scikit-learn library that can be used to classify the contents of an image.

4 components of parametrized learning and linear classifiers

I've used the word "parameterized" a few times now, but what exactly does it mean?

Simply put: parameterization is the process of defining the r

In the task of machine learning, parameterization involves defining ou

1. **Data:** This is our *input data* that we are going to learn from. This vectors, color histograms, raw pixel intensities, etc.) and their as
2. **Score function:** A function that accepts our data as input and r input feature vectors, the score function takes these data points en returns the predicted class labels.
3. **Loss function:** A loss function quantifies how well our *predicte* higher level of agreement between these two sets of labels, the , at least on the training data). Our goal is to minimize our loss fun
4. **Weight matrix:** The weight matrix, typically denoted as *W*, is c at we'll actually be optimizing. Based on the output of our score fu g with the values of our weight matrix to increase classification acc

Note: *Depending on your type of model, there may exist* many *more* e *4 building blocks of parameterized learning that you'll commonly see.*

Once we've defined these 4 key components, we can then apply optimization methods that allow us to find a set of parameters *W* that minimize our loss function with respect to our score function (while increasing classification accuracy on the data).

Next, we'll look at how these components work together to build a linear classifier, transforming the input data into actual predictions.

Linear classification: from images to labels

In this section, we are going to look at a more mathematical motivation of the parameterized model to machine learning.

To start we need our ***data***. Let's assume that our training dataset (either of images or extracted feature vectors) is denoted as x_i where each image/feature vector has an associated class label y_i. We'll assume $i = 1 \dots N$ and $y_i = 1 \dots K$ implying that we have *N* data points of dimensionality *D* (the "length" of the feature vector), separated into *K* unique categories.

To make this more concrete, consider our previous tutorial on using the k-NN classifier to recognize dogs and cats in images based on extracted color histograms.

This this dataset, we have *N = 25,000* total images. Each image is [redacted] channel, respectively. This yields a feature vector with *D = 8 x 8 x 8 = 512* entries. Finally, we know there are a total of *K = 2* class labels, one for the *"dog"* class and another for the *"cat"* class.

Given these variables, we must now define a score function *f* that maps the feature vectors to the class label scores. As the title of this blog post suggests, we'll be using a simple linear mapping:

$$f(x_i, W, b) = Wx_i + b$$

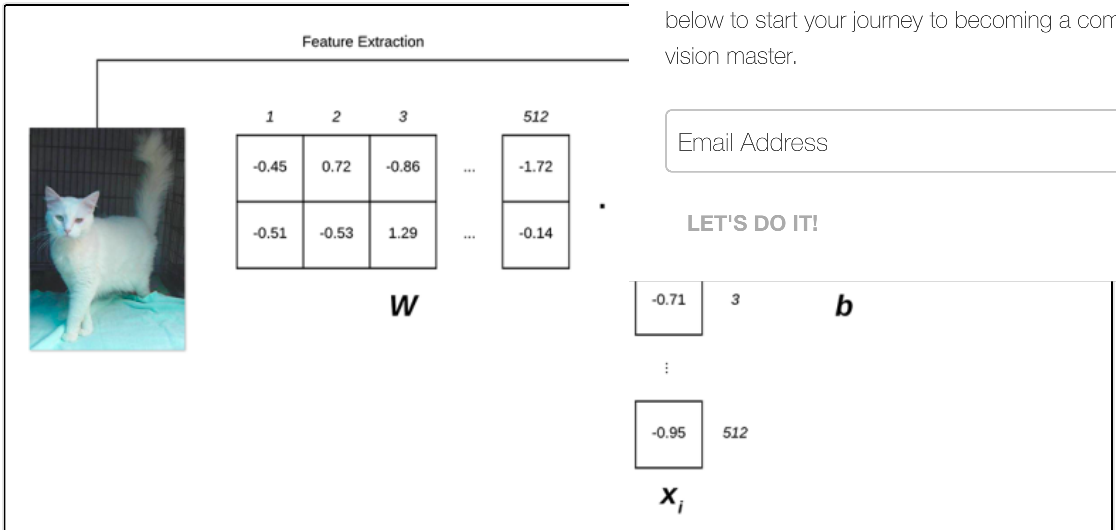
We'll assume that each x_i is represented as a single column vector with shape *[D x 1]*. Again, in this example, we'll be using color histograms — but if we were utilizing raw pixel intensities, we can simply *flatten* the pixels of the image into a single vector.

Our weight matrix *W* has a shape of *[K x D]* (the number of class labe

Finally, *b*, the **bias vector** is of size *[K x 1]*. The bias vector essentia direction or another, without actually influencing our weight matrix *W*

Going back to the Kaggle Dogs vs. Cats example, each x_i is repres as the shape *[512 x 1]*. The weight matrix *W* will have a shape of *[2 x 5*

Below follows an illustration of the linear classification scoring functior



1	2	3	...	512
-0.45	0.72	-0.86	...	-1.72
-0.51	-0.53	1.29	...	-0.14

Figure 1: Illustrating the dot product of weight matrix W and feature vector x, followed by addition of the bias term. (Inspired by Karpathy's example in the CS231n course).

On the *left*, we have our original input image, which we extract features from. In this example, we're computing a 512-d color histogram, but any other feature representation could be used (including the raw pixel intensities themselves), but in this case, we'll simply use a color distribution — this histogram is our x_i representation.

We then have our weight matrix *W*, which contains 2 rows (one for each class label) and 512 columns (one for each of the entries in the feature vector).

After taking the dot product between W and x_i, we add in the bias vector b, which has a shape of *[2 x 1]*.

Finally, this yields two values on the *right*: the scores associated with

Looking at the above equation, you can convince yourself that the input [redacted] *modify*. Sure, we can obtain different x_i 's by applying a different feature extraction technique — but once the features are extracted, **these values do not change.**

In fact, the only parameters that we have any control over are our weight matrix *W* and our bias vector *b*. Therefore, our goal is to utilize both our scoring function and loss function to *optimize* (i.e., modify) the weight and bias vectors such that our classification accuracy *increases*.

Exactly *how* we optimize the weight matrix depends on our loss function, but typically involves some form of gradient descent — we'll be reviewing optimization and loss functions in a future blog post, but for the time being, simply understand that given a scoring function, we also define a loss function that tells us how "good" our predictions are on the input data.

Advantages of parametrized learning and line

There are two primary advantages to utilizing parameterized learning,

1. **Once we are done training our model, we can discard th** **W** **and the bias vector *b*.** This substantially reduces the size of c vectors (versus the *entire* training set).
2. **Classifying new test data is *fast*.** In order to perform a class W and x_i , followed by adding in the bias b . Doing this is *sub* g point to *every* training example (as in the k-NN algorithm).

Now that we understand linear classification, let's see how we can im

Linear classification of images with Python, O

Much like in our previous example on the Kaggle Dogs vs. Cats data histograms from the dataset; however, unlike the previous example, w

Specifically, we'll be using a Linear Support Vector Machine (SVM) which constructs a maximum-margin separating hyperplane between data classes in an *n*-dimensional space. The goal of this separating hyperplane is to place all examples (or as many as possible, given some tolerance) of class *i* on one side of the hyperplane and then all examples *not of class i* on the other side of the hyperplane.

A detailed description of how Support Vector Machines work is outside the scope of this blog post (but is covered inside the **PyImageSearch Gurus course**).

In the meantime, simply understand that our Linear SVM utilizes a score function *f* similar to the one in the *"Linear classification: from images to class labels"* section of this blog post and then applies a loss function that is used to determine the maximum-margin separating hyperplane to classify the data points (again, we'll be looking at loss functions in future blog posts).

To get started, open up a new file, name it `linear_classifier.py` , and insert the following code:

An intro to linear classification with Python Python

```python
# import the necessary packages
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import LinearSVC
from sklearn.metrics import classification_report
from sklearn.cross_validation import train_test_split
from imutils import paths
import numpy as np
```

[redacted]

```
8  import argparse
9  import imutils
10 import cv2
11 import os
```

Lines 2-111 handle importing our required Python packages. We'll be making use of the scikit-learn library, so if you do not already have it installed, make sure you follow these instructions to get it setup on your machine.

We'll also be using my imutils Python package, a set of image processing convenience functions. If you do not already have `imutils` installed, just let `pip` install it for you:

An intro to linear classification with Python Shell

```
1 $ pip install imutils
```

We'll now define our `extract_color_histogram` function which will input images:

An intro to linear classification with Python n

```
13 def extract_color_histogram(image, bins=(8, 8, 8)):
14     # extract a 3D color histogram from the HSV color s
15     # the supplied number of `bins` per channel
16     hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
17     hist = cv2.calcHist([hsv], [0, 1, 2], None, bins,
18         [0, 180, 0, 256, 0, 256])
19
20     # handle normalizing the histogram if we are using
21     if imutils.is_cv2():
22         hist = cv2.normalize(hist)
23
24     # otherwise, perform "in place" normalization in Op
25     # personally hate the way this is done
26     else:
27         cv2.normalize(hist, hist)
28
29     # return the flattened histogram as the feature vec
30     return hist.flatten()
```

This function accepts an input `image`, converts it to the HSV color g the supplied number of `bins` for each channel.

After computing the color histogram using the `cv2.calcHist` function, the histogram is normalized and then returned to the calling function.

For a more detailed review of the `extract_color_histogram` method, please refer to this blog post.

Next, let's parse our command line arguments and initialize a few variables:

An intro to linear classification with Python Python

```
32 # construct the argument parse and parse the arguments
33 ap = argparse.ArgumentParser()
34 ap.add_argument("-d", "--dataset", required=True,
35     help="path to input dataset")
36 args = vars(ap.parse_args())
37
38 # grab the list of images that we'll be describing
39 print("[INFO] describing images...")
40 imagePaths = list(paths.list_images(args["dataset"]))
41
42 # initialize the data matrix and labels list
43 data = []
44 labels = []
```

Lines 33-36 parse our command line arguments. We only need a [redacted] our input Kaggle Dogs vs. Cats dataset.

We then grab the `imagePaths` to where each of the 25,000 images reside on disk, followed by initializing a `data` matrix to store our extracted feature vectors along with our class `labels` .

Speaking of extracting features, let's go ahead and do that:

An intro to linear classification with Python Python

```
46 # loop over the input images
47 for (i, imagePath) in enumerate(imagePaths):
48     # load the image and extract the class label (assuming that our
49     # path as the format: /path/to/dataset/{class}.{image_num}.jpg
50     image = cv2.imread(imagePath)
51     label = imagePath.split(os.path.sep)[-1].split(".")
52 
53     # extract a color histogram from the image, then up
54     # data matrix and labels list
55     hist = extract_color_histogram(image)
56     data.append(hist)
57     labels.append(label)
58 
59     # show an update every 1,000 images
60     if i > 0 and i % 1000 == 0:
61         print("[INFO] processed {}/{}".format(i, len(im
```

On **Line 47** we start looping over our input `imagePaths` . For each the class `label` , and then quantify the image by computing a color his ts, respectively.

Currently, our `labels` list is represented as a list of *strings*, either "c algorithms in scikit-learn prefer that the `labels` are encoded as *inte*

Performing this conversion of class label string-to-integer is easy with

An intro to linear classification with Python Python

```
63 # encode the labels, converting them from strings to integers
64 le = LabelEncoder()
65 labels = le.fit_transform(labels)
```

After the `.fit_transform` method is called, our `labels` are now represented as a list of *integers.*

Our final code block will handle partitioning our data into training/testing splits, followed by training our Linear SVM and evaluating it:

An intro to linear classification with Python Python

```
67 # partition the data into training and testing splits, using 75%
68 # of the data for training and the remaining 25% for testing
69 print("[INFO] constructing training/testing split...")
70 (trainData, testData, trainLabels, testLabels) = train_test_split(
71     np.array(data), labels, test_size=0.25, random_state=42)
72 
73 # train the linear regression clasifier
74 print("[INFO] training Linear SVM classifier...")
75 model = LinearSVC()
76 model.fit(trainData, trainLabels)
77 
78 # evaluate the classifier
79 print("[INFO] evaluating classifier...")
80 predictions = model.predict(testData)
81 print(classification_report(testLabels, predictions,
82     target_names=le.classes_))
```

Lines 70 and 71 handle constructing our training and testing split. [redacted] remaining 25% for testing.

To train our Linear SVM, we'll utilize the `LinearSVC` implementation from the scikit-learn library (**Lines 75 and 76**).

Finally, we evaluate our classifier on **Lines 80-82**, displaying a nicely formatted classification report on how well our model performed.

One thing you'll notice here is that I'm *purposely* not tuning hyperparameters here, simply to keep this example shorter and easier to digest. However, with that said, I leave tuning the hyperparameters of the `LinearSVC` classifier as an exercise to you, the reader. Use our previous blog post on tuning the hyperparameters of the k-NN classifier as an example.

Evaluating our linear classifier

To test out our linear classifier, make sure you have downloaded:

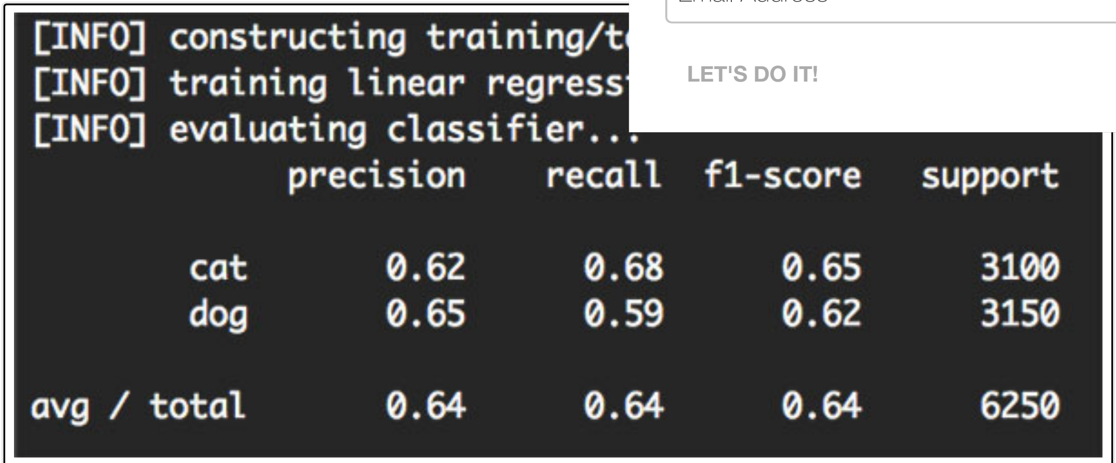
1. The source code to this blog post using the ***"Downloads"*** sec
2. The Kaggle Dogs vs. Cats dataset.

Once you have the code and dataset, you can execute the following

```
An intro to linear classification with Python
$ python linear_classifier.py --dataset kaggle_dogs_vs_c
```

The feature extraction process should take approximately 1-3 minute

From there, our Linear SVM is trained and evaluated:



```
[INFO] constructing training/te
[INFO] training linear regress
[INFO] evaluating classifier...
              precision    recall  f1-score   support

        cat       0.62      0.68      0.65      3100
        dog       0.65      0.59      0.62      3150

avg / total       0.64      0.64      0.64      6250
```

Figure 2: Training and evaluating our linear classifier using Python, OpenCV, and scikit-learn.

As the above figure demonstrates, we were able to obtain ***64% classification accuracy***, or approximately the same accuracy as using tuned hyperparameters from the k-NN algorithm in this tutorial.

Note: *Tuning the hyperparameters to the Linear SVM will lead to a higher classification accuracy — I simply left out this step to make the tutorial a little shorter and less overwhelming.*

Furthermore, not only did we obtain the same classification accuracy as k-NN, but our model is *much* faster at testing time, requiring only a (highly optimized) dot product between the weight matrix and data points, followed by a simple addition.

We are also able to *discard* the training data after training is complete is vector *b*, leading to a much more compact representation of the clas

Summary

In today's blog post, I discussed the basics of parameterized learning and linear classification. While simple, the linear classifier can be seen as the fundamental building blocks of more advanced machine learning algorithms, extending naturally to Neural Networks and Convolutional Neural Networks.

You see, Convolutional Neural Networks will perform a mapping of raw pixels to class labels similar to what we did in this tutorial — only our score function *f* will become substantially more complex and contain *many* more parameters.

A primary benefit of this parameterized approach to learning is that it allows us to *discard* our training data after our model has been trained. We can then perform classification using *only* the parameters (i.e., weight matrix and bias vector) learned from the data.

This allows classification to be performed ***much more efficiently*** s g
data in our model, like in k-NN and (2) we do not need to compare a
scales *O(N)*, and can become quite cumbersome given many trainin

In short, this method is ***significantly faster***, requiring only a single

Finally, we applied a linear classifier using Python, OpenCV, and sciki
extracting color histograms from the dataset, we trained a Linear Sup
obtained a classification accuracy of ***64%***, which is fairly reasonable e
for characterizing dogs vs. cats and (2) we did not tune the hyperpar

At this point, we are starting to understand the basic building blocks
Neural Networks, and Deep Learning models, but there is still a ways

To start, we need to understand loss functions in more detail, and in ur
weight matrix to obtain more accurate predictions. Future blog posts

Before you go, be sure to sign up for the PyImageSearch Newsletter using the form below to be notified when new blog posts are published!

Downloads:

If you would like to download the code and images used in this post, please enter your email address in the form below. Not only will you get a .zip of the code, I'll also send you a **FREE 11-page Resource Guide** on Computer Vision and Image Search Engines, including **exclusive techniques** that I don't post on this blog! Sound good? If so, enter your email address and I'll send you the code immediately!

Email address:

Your email address

DOWNLOAD THE CODE!

Resource Guide (it's totally free).

[redacted]

Image Search Engine
Resource Guide

Adrian Rosebrock

pyimagesearch

Enter your email address below to get my **free 11-page Image Search Engine Resource Guide PDF**. Uncover **exclusive techniques** that I don't publish on this blog and start building image search engines of your own!

Your email address

DOWNLOAD THE GUIDE!

classification, **color descriptor**, **color histograms**, **dogs and ca**

‹ How to tune hyperparameters with Python and scikit-learn

le ›

8 Responses to *An intro to linear classification w*

Sifan August 23, 2016 at 11:15 am #

Great tutorial and explanation,thank you! I want to use feature
Color_histogram,can you show me how?thank you

Adrian Rosebrock August 24, 2016 at 8:30 am #

I demonstrate how to use Histogram of Oriented Gradients as feature inputs to a Linear SVM inside Practical Python and OpenCV. To utilize SIFT/SURF or other keypoint detectors + local invariant descriptors you'll need to construct a bag of visual words (BOVW) model. I discuss the BOVW model in detail (with lots of code examples) and apply it to image classification inside PyImageSearch Gurus. Be sure to take a look!

Rohan Saxena September 14, 2016 at 2:38 am # REPLY

First of all, being bored at work does pay well if you have a Smartphone and you browse through blogs.Amazing information with facts thoughtfully incorporated within. Definitely going to come back for more! 🙂

Adrian Rosebrock September 15, 2016 at 9:35 am # REPLY

Thank you for the kind words Rohan! 🙂

Antoni October 18, 2016 at 4:54 am #

[redacted]

Now that I have train the model how do I test it

I mean with a different image from the ones i used and it isnt labeled like the ones in the data set

Adrian Rosebrock October 20, 2016 at 8:59 am # REPLY

You would load your image from disk, pre-process it exactly like we did in this blog post, and call `model.predict` on it.

For what it's worth, I demonstrate the basics of using machine learning classifiers with OpenCV + Python inside my book, Practical Python and OpenCV.

Trackbacks/Pingbacks

Multi-class SVM Loss - PyImageSearch - September 5, 2016
[…] couple weeks ago,we discussed the concepts of both linear classific
allows us to take a set of input data and class labels, and actually learn a

Gradient descent with Python - PyImageSearch - October 10, 2016
[…] started with an introduction to linear classification that discussed the c
learning enables us to […]

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Resource Guide (it's totally free).

Click the button below to get my **free 11-page Image Search Engine Resource Guide PDF**. Uncover **exclusive techniques** that I don't publish on this blog and start building image search engines of your own.

[redacted]

Download for Free!

Deep Learning for Computer Vision with Python Book

You're interested in deep learning and computer vision, *but you don't know how* **all you need to know about deep learning.**

CLICK HERE TO PRE-ORDER MY NEW BOOK

Free 21-day crash course on computer vision & image search engines

Interested in computer vision and image search engines, but don't know where to start? Let me help. I've created a free, 21-day crash course that is hand-tailored to give you the best possible introduction to computer vision. Sound good? Enter your email below to start your journey to becoming a computer vision master.

Email Address

LET'S DO IT!

You can detect faces in images & video.

Are you interested in **detecting faces in images & video?** But **tired of Googling for tutorials** that *never work?* Then let me help! I guarantee that my new book will turn you into a **face detection ninja** by the end of this weekend. Click here to give it a shot yourself.

CLICK HERE TO MASTER FACE DETECTION

PyImageSearch Gurus: NOW ENROLLING!

Free 21-day crash course on computer vision & image search engines

[redacted]

The PyImageSearch Gurus course is *now enrolling!* Inside the course you'll learn how to perform:

- Automatic License Plate Recognition (ANPR)
- Deep Learning
- Face Recognition
- *and much more!*

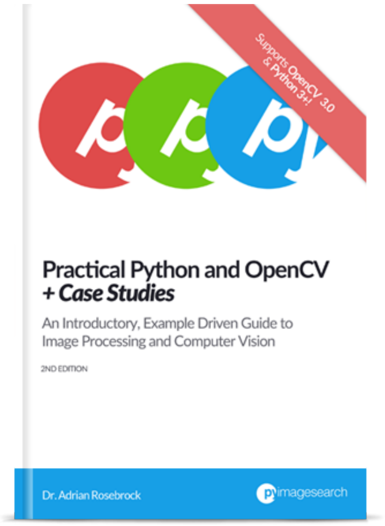
Click the button below to learn more about the course, take a tour, a

TAKE A TOUR & GET 10 (FREE) LESSONS

Hello! I'm Adrian Rosebrock.

I'm an entrepreneur and Ph.D who has launched two succ here to share my tips, tricks, and hacks I've learned along

Learn computer vision in a single weekend.


Supports OpenCV 3.0 & Python 3+!
Practical Python and OpenCV
+ Case Studies
An Introductory, Example Driven Guide to Image Processing and Computer Vision
2ND EDITION
Dr. Adrian Rosebrock
pyimagesearch

Want to learn computer vision & OpenCV? I can teach you in a **single weekend**. I know. It sounds crazy, but it's no joke. My new book is your **guaranteed, quick-start guide** to becoming an OpenCV Ninja. So why not give it a try? Click here to become a computer vision ninja.

CLICK HERE TO BECOME AN OPENCV NINJA

[redacted]

Subscribe via RSS

Never miss a post! Subscribe to the PyImageSearch RSS Feed and keep up to date with my image search engine tutorials, tips, and tricks

POPULAR

Install OpenCV and Python on your Raspberry Pi 2 and B+
FEBRUARY 23, 2015

Home surveillance and motion detection with the Raspberry Pi, Python, OpenCV, and Dropbox
JUNE 1, 2015

How to install OpenCV 3 on Raspbian Jessie
OCTOBER 26, 2015

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3
APRIL 18, 2016

Basic motion detection and tracking with Python and OpenCV
MAY 25, 2015

Install OpenCV 3.0 and Python 2.7+ on Ubuntu
JUNE 22, 2015

Accessing the Raspberry Pi Camera with OpenCV and Python
MARCH 30, 2015

Search

Search...

Find me on **Twitter**, **Facebook**, **Google+**, and **LinkedIn**.