Face Recognition Documentation

Release 0.1.0

Adam Geitgey

Apr 20, 2017

Contents

Contents:

CHAPTER 1

Face Recognition

Recognize and manipulate faces from Python or from the command line with the world's simplest face recognition library.

Built using dlib's state-of-the-art face recognition built with deep learning. The model has an accuracy of 99.38% on the Labeled Faces in the Wild benchmark.

This also provides a simple `face_recognition` command line tool that lets you do face recognition on a folder of images from the command line!

Features

Find faces in pictures

Find all the faces that appear in a picture:

```
import face_recognition
image = face_recognition.load_image_file("your_file.jpg")
face_locations = face_recognition.face_locations(image)
```

Find and manipulate facial features in pictures

Get the locations and outlines of each person's eyes, nose, mouth and chin.

```
import face_recognition
image = face_recognition.load_image_file("your_file.jpg")
face_landmarks_list = face_recognition.face_landmarks(image)
```

Finding facial features is super useful for lots of important stuff. But you can also use for really stupid stuff like applying digital make-up (think 'Meitu'):

Identify faces in pictures

Recognize who appears in each photo.

```
import face_recognition
known_image = face_recognition.load_image_file("biden.jpg")
unknown_image = face_recognition.load_image_file("unknown.jpg")

biden_encoding = face_recognition.face_encodings(known_image)[0]
unknown_encoding = face_recognition.face_encodings(unknown_image)[0]

results = face_recognition.compare_faces([biden_encoding], unknown_encoding)
```

Installation

Python 3 / Python 2 are fully supported. Only macOS and
Linux are tested. I have no idea if this will work on Windows.

Install this module from pypi using `pip3` (or `pip2` for Python 2):

```
pip3 install face_recognition
```

IMPORTANT NOTE: It's very likely that you will run into problems when pip tries to compile the `dlib` dependency. If that happens, check out this guide to installing dlib from source (instead of from pip) to fix the error:

How to install dlib from source

After manually installing `dlib`, try running `pip3 install face_recognition` again to complete your installation.

Usage

Command-Line Interface

When you install `face_recognition`, you get a simple command-line program called `face_recognition` that you can use to recognize faces in a photograph or folder full for photographs.

First, you need to provide a folder with one picture of each person you already know. There should be one image file for each person with the files named according to who is in the picture:

Next, you need a second folder with the files you want to identify:

Then in you simply run the command `face_recognition`, passing in the folder of known people and the folder (or single image) with unknown people and it tells you who is in each image:

```
$ face_recognition ./pictures_of_people_i_know/ ./unknown_pictures/

/unknown_pictures/unknown.jpg,Barack Obama
/face_recognition_test/unknown_pictures/unknown.jpg,unknown_person
```

There's one line in the output for each face. The data is comma-separated with the filename and the name of the person found.

An `unknown_person` is a face in the image that didn't match anyone in your folder of known people.

If you simply want to know the names of the people in each photograph but don't care about file names, you could do this:

```
$ face_recognition ./pictures_of_people_i_know/ ./unknown_pictures/ | cut -d ',' -f2

Barack Obama
unknown_person
```

Python Module

You can import the `face_recognition` module and then easily manipulate faces with just a couple of lines of code. It's super easy!

API Docs: https://face-recognition.readthedocs.io.

Automatically find all the faces in an image

```
import face_recognition

image = face_recognition.load_image_file("my_picture.jpg")
face_locations = face_recognition.face_locations(image)

# face_locations is now an array listing the co-ordinates of each face!
```

See this example
to try it out.

Automatically locate the facial features of a person in an image

```
import face_recognition

image = face_recognition.load_image_file("my_picture.jpg")
face_landmarks_list = face_recognition.face_landmarks(image)

# face_landmarks_list is now an array with the locations of each facial feature in␣
↪each face.
# face_landmarks_list[0]['left_eye'] would be the location and outline of the first␣
↪person's left eye.
```

See this example
to try it out.

Recognize faces in images and identify who they are

```
import face_recognition

picture_of_me = face_recognition.load_image_file("me.jpg")
my_face_encoding = face_recognition.face_encodings(picture_of_me)[0]

# my_face_encoding now contains a universal 'encoding' of my facial features that can␣
↪be compared to any other picture of a face!

unknown_picture = face_recognition.load_image_file("unknown.jpg")
unknown_face_encoding = face_recognition.face_encodings(unknown_picture)[0]

# Now we can see the two face encodings are of the same person with `compare_faces`!

results = face_recognition.compare_faces([my_face_encoding], unknown_face_encoding)

if results[0] == True:
    print("It's a picture of me!")
else:
    print("It's not a picture of me!")
```

See this example
to try it out.

Python Code Examples

All the examples are available here.

- Find faces in a photograph
- Identify specific facial features in a photograph
- Apply (horribly ugly) digital make-up
- Find and recognize unknown faces in a photograph based on photographs of known people
- Recognize faces in live video using your webcam (Requires OpenCV to be installed)

How Face Recognition Works

If you want to learn how face location and recognition work instead of
depending on a black box library, read my article.

Caveats

- The face recognition model is trained on adults and does not work very well on children. It tends to mix up children quite easy using the default comparison threshold of 0.6.

Common Issues

Solution: `dlib` is compiled with SSE4 or AVX support, but your CPU is too old and doesn't support that.
You'll need to recompile `dlib` after making the code change outlined here.

Solution: Your webcam probably isn't set up correctly with OpenCV. Look here for more.

Solution: The face_recognition_models file is too big for your available pip cache memory. Instead,
try `pip2 --no-cache-dir install face_recognition` to avoid the issue.

Thanks

- Many, many thanks to Davis King (@nulhom) for creating dlib and for providing the trained facial feature detection and face encoding models used in this library. For more information on the ResNet that powers the face encodings, check out his blog post.

- Thanks to everyone who works on all the awesome Python data science libraries like numpy, scipy, scikit-image, pillow, etc, etc that makes this kind of stuff so easy and fun in Python.
- Thanks to Cookiecutter and the audreyr/cookiecutter-pypackage project template for making Python project packaging way more tolerable.

CHAPTER 2

Installation

Stable release

To install Face Recognition, run this command in your terminal:

```
$ pip3 install face_recognition
```

This is the preferred method to install Face Recognition, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

From sources

The sources for Face Recognition can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/ageitgey/face_recognition
```

Or download the tarball:

```
$ curl  -OL https://github.com/ageitgey/face_recognition/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use Face Recognition in a project:

```
import face_recognition
```

See the examples in the /examples folder on github for how to use each function.

You can also check the API docs for the 'face_recognition' module to see the possible paramters for each function.

The basic idea is that first you load an image:

```
import face_recognition

image = face_recognition.load_image_file("your_file.jpg")
```

That loads the image into a numpy array. If you already have an image in a numpy array, you can skip this step.

Then you can perform operations on the image, like finding faces, identifying facial features or finding face encodings:

```
# Find all the faces in the image
face_locations = face_recognition.face_locations(image)

# Or maybe find the facial features in the image
face_landmarks_list = face_recognition.face_landmarks(image)

# Or you could get face encodings for each face in the image:
list_of_face_encodings = face_recognition.face_encodings(image)
```

Face encodings can be compared against each other to see if the faces are a match. Note: Finding the encoding for a face is a bit slow, so you might want to save the results for each image in a database or cache if you need to refer back to it later.

But once you have the encodings for faces, you can compare them like this:

```
# results is an array of True/False telling if the unknown face matched anyone in the␣
↪known_faces array
results = face_recognition.compare_faces(known_face_encodings, a_single_unknown_face_
↪encoding)
```

It's that simple! Check out the examples for more details.

CHAPTER 4

face_recognition

face_recognition package

Module contents

face_recognition.api.**compare_faces**(*known_face_encodings*, *face_encoding_to_check*, *tolerance=0.6*)

Compare a list of face encodings against a candidate encoding to see if they match.

Parameters

- **known_face_encodings** – A list of known face encodings
- **face_encoding_to_check** – A single face encoding to compare against the list
- **tolerance** – How much distance between faces to consider it a match. Lower is more strict. 0.6 is typical best performance.

Returns A list of True/False values indicating which known_face_encodings match the face encoding to check

face_recognition.api.**face_distance**(*face_encodings*, *face_to_compare*)

Given a list of face encodings, compare them to a known face encoding and get a euclidean distance for each comparison face. The distance tells you how similar the faces are.

Parameters

- **faces** – List of face encodings to compare
- **face_to_compare** – A face encoding to compare against

Returns A numpy ndarray with the distance for each face in the same order as the 'faces' array

face_recognition.api.**face_encodings**(*face_image*, *known_face_locations=None*, *num_jitters=1*)

Given an image, return the 128-dimension face encoding for each face in the image.

Parameters

- **face_image** – The image that contains one or more faces
- **known_face_locations** – Optional - the bounding boxes of each face if you already know them.
- **num_jitters** – How many times to re-sample the face when calculating encoding. Higher is more accurate, but slower (i.e. 100 is 100x slower)

Returns A list of 128-dimentional face encodings (one for each face in the image)

face_recognition.api.**face_landmarks**(*face_image*, *face_locations=None*)

Given an image, returns a dict of face feature locations (eyes, nose, etc) for each face in the image

Parameters

- **face_image** – image to search
- **face_locations** – Optionally provide a list of face locations to check.

Returns A list of dicts of face feature locations (eyes, nose, etc)

face_recognition.api.**face_locations**(*img*, *number_of_times_to_upsample=1*)

Returns an array of bounding boxes of human faces in a image

Parameters

- **img** – An image (as a numpy array)
- **number_of_times_to_upsample** – How many times to upsample the image looking for faces. Higher numbers find smaller faces.

Returns A list of tuples of found face locations in css (top, right, bottom, left) order

face_recognition.api.**load_image_file**(*filename*, *mode='RGB'*)

Loads an image file (.jpg, .png, etc) into a numpy array

Parameters

- **filename** – image file to load
- **mode** – format to convert the image to. Only 'RGB' (8-bit RGB, 3 channels) and 'L' (black and white) are supported.

Returns image contents as numpy array

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/ageitgey/face_recognition/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/ageitgey/face_recognition/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *face_recognition* for local development.

1. Fork the *face_recognition* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/face_recognition.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv face_recognition
$ cd face_recognition/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 face_recognition tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/ageitgey/face_recognition/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ python -m unittest tests.test_face_recognition
```

CHAPTER 6

Authors

- Adam Geitgey <ageitgey@gmail.com>

Thanks

- Many, many thanks to Davis King (@nulhom) for creating dlib and for providing the trained facial feature detection and face encoding models used in this library.
- Thanks to everyone who works on all the awesome Python data science libraries like numpy, scipy, scikit-image, pillow, etc, etc that makes this kind of stuff so easy and fun in Python.
- Thanks to Cookiecutter and the audreyr/cookiecutter-pypackage project template for making Python project packaging way more tolerable.

CHAPTER 7

History

0.1.12 (2017-04-13)

- Fixed: Face landmarks wasn't returning all chin points.

0.1.11 (2017-03-30)

- Fixed a minor bug in the command-line interface.

0.1.10 (2017-03-21)

- Minor pref improvements with face comparisons.
- Test updates.

0.1.9 (2017-03-16)

- Fix minimum scipy version required.

0.1.8 (2017-03-16)

- Fix missing Pillow dependency.

0.1.7 (2017-03-13)

- First working release.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index

C

F

L