Menpo Documentation

Release 0.6.2

Joan Alabort-i-Medina, Epameinondas Antonakos, James Booth,

Jan 09, 2021

CONTENTS

Welcome to the MenpoDetect documentation!

MenpoDetect is a Python package designed to make object detection, in particular face detection, simple. MenpoDetect relies on the core package of Menpo, and thus the output of MenpoDetect is always assumed to be Menpo core types. If you aren't sure what Menpo is, please take a look over at Menpo.org.

A short example is often more illustrative than a verbose explanation. Let's assume that you want to load a set of images and that we want to detect all the faces in the images. We could do this using the Viola-Jones detector provided by OpenCV as follows:

```
import menpo.io as mio
from menpodetect import load_opencv_frontal_face_detector

opencv_detector = load_opencv_frontal_face_detector()

images = []
for image in mio.import_images('./images_folder'):
    opencv_detector(image)
    images.append(image)
```

Where we use Menpo to load the images from disk and then detect as many faces as possible using OpenCV. The detections are automatically attached to each image in the form of a set of landmarks.

CHAPTER

ONE

SUPPORTED DETECTORS

MenpoDetect was not designed for performing novel object detection research. Therefore, it relies on a number of existing packages and merely normalizes the inputs and outputs so that they are consistent with core Menpo types. These projects are as follows:

- dlib - Provides the detection capabilities of the Dlib project. This is a HOG-SVM based detector that will return a very low number of false positives.
- OpenCV - Provides the detection capabilities of the OpenCV project. OpenCV implements a Viola-Jones detector and provides models for both frontal and profile faces as well as eyes.

We would be very happy to see this collection expand, so pull requests are very welcome!

1.1 The MenpoDetect API

This section attempts to provide a simple browsing experience for the MenpoDetect documentation. In MenpoDetect, we use legible docstrings, and therefore, all documentation should be easily accessible in any sensible IDE (or IPython) via tab completion. However, this section should make most of the core classes available for viewing online.

1.1.1 `menpodetect.detect`

This module contains a base implementation of the generic detection method. It also provides other helper methods that are useful for all detectors. In general you will never instantiate one of these directly.

Core

detect

`menpodetect.detect.`**`detect`**(*detector_callable*, *image*, *greyscale=True*, *image_diagonal=None*, *group_prefix='object'*, *channels_at_back=True*)

Apply the general detection framework.

This involves converting the image to greyscale if necessary, rescaling the image to a given diagonal, performing the detection, and attaching the scaled landmarks back onto the original image.

uint8 images cannot be converted to greyscale by this framework, so must already be greyscale or `greyscale=False`.

Parameters

- **detector_callable** (*callable* or *function*) – A callable object that will perform detection given a single parameter, a *uint8* numpy array with either no channels, or channels as the *last* axis.
- **image** (*menpo.image.Image*) – A Menpo image to detect. The bounding boxes of the detected objects will be attached to this image.
- **greyscale** (*bool*, optional) – Convert the image to greyscale or not.
- **image_diagonal** (*int*, optional) – The total size of the diagonal of the image that should be used for detection. This is useful for scaling images up and down for detection.
- **group_prefix** (*str*, optional) – The prefix string to be appended to each each landmark group that is stored on the image. Each detection will be stored as group_prefix_# where # is a count starting from 0.
- **channels_at_back** (*bool*, optional) – If `True`, the image channels are placed onto the last axis (the back) as is common in many imaging packages. This is contrary to the Menpo default where channels are the first axis (at the front).

Returns **bounding_boxes** (*list* of *menpo.shape.PointDirectedGraph*) – A list of bounding boxes representing the detections found.

Convenience

menpo_image_to_uint8

`menpodetect.detect.`**menpo_image_to_uint8**(*image*, *channels_at_back=True*)

Return the given image as a uint8 array. This is a copy of the image.

Parameters

- **image** (*menpo.image.Image*) – The image to convert. If already uint8, only the channels will be rolled to the last axis.
- **channels_at_back** (*bool*, optional) – If `True`, the image channels are placed onto the last axis (the back) as is common in many imaging packages. This is contrary to the Menpo default where channels are the first axis (at the front).

Returns **uint8_image** (*ndarray*) – *uint8* Numpy array, channels as the back (last) axis if `channels_at_back == True`.

1.1.2 menpodetect.dlib

This module contains a wrapper of the detector provided by the Dlib[1][2] project. In particular, it provides access to a frontal face detector that implements the work from[3]. The Dlib detector is also trainable.

[1] http://dlib.net/
[2] King, Davis E. "Dlib-ml: A machine learning toolkit." The Journal of Machine Learning Research 10 (2009): 1755-1758.
[3] King, Davis E. "Max-Margin Object Detection." arXiv preprint arXiv:1502.00046 (2015).

Detection

DlibDetector

class menpodetect.dlib.**DlibDetector**(*model*)
Bases: object

A generic dlib detector.

Wraps a dlib object detector inside the menpodetect framework and provides a clean interface to expose the dlib arguments.

__call__(*image*, *greyscale=False*, *image_diagonal=None*, *group_prefix='dlib'*, *n_upscales=0*)
Perform a detection using the cached dlib detector.

The detections will also be attached to the image as landmarks.

Parameters

- **image** (*menpo.image.Image*) – A Menpo image to detect. The bounding boxes of the detected objects will be attached to this image.
- **greyscale** (*bool*, optional) – Convert the image to greyscale or not.
- **image_diagonal** (*int*, optional) – The total size of the diagonal of the image that should be used for detection. This is useful for scaling images up and down for detection.
- **group_prefix** (*str*, optional) – The prefix string to be appended to each each landmark group that is stored on the image. Each detection will be stored as group_prefix_# where # is a count starting from 0.
- **n_upscales** (*int*, optional) – Number of times to upscale the image when performing the detection, may increase the chances of detecting smaller objects.

Returns **bounding_boxes** (*list* of *menpo.shape.PointDirectedGraph*) – The detected objects.

load_dlib_frontal_face_detector

menpodetect.dlib.**load_dlib_frontal_face_detector**()
Load the dlib frontal face detector.

Returns **detector** (*DlibDetector*) – The frontal face detector.

Training

train_dlib_detector

menpodetect.dlib.**train_dlib_detector**(*images*, *epsilon=0.01*, *add_left_right_image_flips=False*, *verbose_stdout=False*, *C=5*, *detection_window_size=6400*, *num_threads=None*)
Train a dlib detector with the given list of images.

This is intended to easily train a list of menpo images that have their bounding boxes attached as landmarks. Each landmark group on the image will have a tight bounding box extracted from it and then dlib will train given these images.

Parameters

- **images** (*list* of *menpo.image.Image*) – The set of images to learn the detector from. Must have landmarks attached to **every** image, a bounding box will be extracted for each landmark group.
- **epsilon** (*float*, optional) – The stopping epsilon. Smaller values make the trainer's solver more accurate but might take longer to train.
- **add_left_right_image_flips** (*bool*, optional) – If `True`, assume the objects are left/right symmetric and add in left right flips of the training images. This doubles the size of the training dataset.
- **verbose_stdout** (*bool*, optional) – If `True`, will allow dlib to output its verbose messages. These will only be printed to the stdout, so will **not** appear in an IPython notebook.
- **C** (*int*, optional) – C is the usual SVM C regularization parameter. Larger values of C will encourage the trainer to fit the data better but might lead to overfitting.
- **detection_window_size** (*int*, optional) – The number of pixels inside the sliding window used. The default parameter of `6400 = 80 * 80` window size.
- **num_threads** (*int* > 0 or `None`) – How many threads to use for training. If `None`, will query multiprocessing for the number of cores.

Returns **detector** (*dlib.simple_object_detector*) – The trained detector. To save this detector, call save on the returned object and pass a string path.

Examples

Training a simple object detector from a list of menpo images and save it for later use:

```
>>> images = list(mio.import_images('./images/path'))
>>> in_memory_detector = train_dlib_detector(images, verbose_stdout=True)
>>> in_memory_detector.save('in_memory_detector.svm')
```

References

1.1.3 menpodetect.opencv

This module contains a wrapper of the detector provided by the OpenCV[1] project. At the moment, we assume the use of OpenCV v2.x and therefore this detector will not be available for Python 3.x. We provide a number of pre-trained models that have been provided by the OpenCV community, all of which are implementations of the Viola-Jones method[2].

[1] http://opencv.org/

[2] Viola, Paul, and Michael Jones. "Rapid object detection using a boosted cascade of simple features." Computer Vision and Pattern Recognition, 2001. CVPR 2001.

Detection

OpenCVDetector

class menpodetect.opencv.**OpenCVDetector**(*model*)
Bases: object

A generic opencv detector.

Wraps an opencv object detector inside the menpodetect framework and provides a clean interface to expose the opencv arguments.

__call__(*image*, *image_diagonal=None*, *group_prefix='opencv'*, *scale_factor=1.1*, *min_neighbours=5*, *min_size=(30, 30)*, *flags=None*)
Perform a detection using the cached opencv detector.

The detections will also be attached to the image as landmarks.

Parameters

- **image** (*menpo.image.Image*) – A Menpo image to detect. The bounding boxes of the detected objects will be attached to this image.
- **image_diagonal** (*int*, optional) – The total size of the diagonal of the image that should be used for detection. This is useful for scaling images up and down for detection.
- **group_prefix** (*str*, optional) – The prefix string to be appended to each each landmark group that is stored on the image. Each detection will be stored as group_prefix_# where # is a count starting from 0.
- **scale_factor** (*float*, optional) – The amount to increase the sliding windows by over the second pass.
- **min_neighbours** (*int*, optional) – The minimum number of neighbours (close detections) before Non-Maximum suppression to be considered a detection. Use 0 to return all detections.
- **min_size** (*tuple* of 2 ints) – The minimum object size in pixels that the detector will consider.
- **flags** (*int*, optional) – The flags to be passed through to the detector.

Returns **bounding_boxes** (*list* of *menpo.shape.PointDirectedGraph*) – The detected objects.

load_opencv_frontal_face_detector

menpodetect.opencv.**load_opencv_frontal_face_detector**()
Load the opencv frontal face detector: haarcascade_frontalface_alt.xml

Returns **detector** (*OpenCVDetector*) – The frontal face detector.

load_opencv_profile_face_detector

menpodetect.opencv.**load_opencv_profile_face_detector**()
Load the opencv profile face detector: haarcascade_profileface.xml

Returns **detector** (*OpenCVDetector*) – The profile face detector.

load_opencv_eye_detector

menpodetect.opencv.**load_opencv_eye_detector**()
Load the opencv eye detector: haarcascade_eye.xml

Returns **detector** (*OpenCVDetector*) – The eye detector.

References

INDEX

Symbols

D

L

M

O

T