Object Detection

Modules

C API

Classes

class	**cv::BaseCascadeClassifier**
class	**cv::CascadeClassifier** Cascade classifier class for object detection. More...
struct	**cv::DefaultDeleter< CvHaarClassifierCascade >**
class	**cv::DetectionBasedTracker**
struct	**cv::DetectionROI** struct for detection region of interest (ROI) More...
struct	**cv::HOGDescriptor** Implementation of HOG (Histogram of Oriented Gradients) descriptor and object detector. More...
class	**cv::QRCodeDetector**
class	**cv::SimilarRects**

Enumerations

```
enum  {
        cv::CASCADE_DO_CANNY_PRUNING = 1,
        cv::CASCADE_SCALE_IMAGE = 2,
        cv::CASCADE_FIND_BIGGEST_OBJECT = 4,
        cv::CASCADE_DO_ROUGH_SEARCH = 8
      }
```

Functions

Ptr< **BaseCascadeClassifier::MaskGenerator** >	**cv::createFaceDetectionMaskGenerator** ()
void	**cv::groupRectangles** (std::vector< **Rect** > &rectList, int groupThreshold, double eps=0.2) Groups the object candidate rectangles. More...
void	**cv::groupRectangles** (std::vector< **Rect** > &rectList, std::vector< int > &weights, int groupThreshold, double eps=0.2)
void	**cv::groupRectangles** (std::vector< **Rect** > &rectList, int groupThreshold, double eps, std::vector< int > *weights, std::vector< double > *levelWeights)
void	**cv::groupRectangles** (std::vector< **Rect** > &rectList, std::vector< int > &rejectLevels, std::vector< double > &levelWeights, int groupThreshold, double eps=0.2)
void	**cv::groupRectangles_meanshift** (std::vector< **Rect** > &rectList, std::vector< double > &foundWeights, std::vector< double > &foundScales, double detectThreshold=0.0, **Size** winDetSize=**Size**(64, 128))

Detailed Description

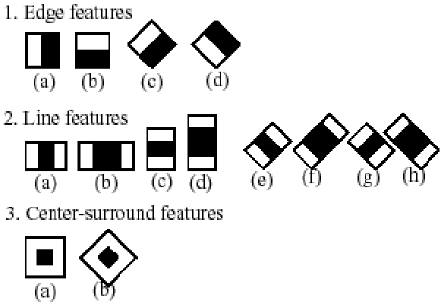
Haar Feature-based Cascade Classifier for Object Detection

The object detector described below has been initially proposed by Paul Viola **[260]** and improved by Rainer Lienhart **[148]** .

First, a classifier (namely a *cascade of boosted classifiers working with haar-like features*) is trained with a few hundred sample views of a particular object (i.e., a face or a car), called positive examples, that are scaled to the same size (say, 20x20), and negative examples - arbitrary images of the same size.

After a classifier is trained, it can be applied to a region of interest (of the same size as used during the training) in an input image. The classifier outputs a "1" if the region is likely to show the object (i.e., face/car), and "0" otherwise. To search for the object in the whole image one can move the search window across the image and check every location using the classifier. The classifier is designed so that it can be easily "resized" in order to be able to find the objects of interest at different sizes, which is more efficient than resizing the image itself. So, to find an object of an unknown size in the image the scan procedure should be done several times at different scales.

The word "cascade" in the classifier name means that the resultant classifier consists of several simpler classifiers (*stages*) that are applied subsequently to a region of interest until at some stage the candidate is rejected or all the stages are passed. The word "boosted" means that the classifiers at every stage of the cascade are complex themselves and they are built out of basic classifiers using one of four different boosting techniques (weighted voting). Currently Discrete Adaboost, Real Adaboost, Gentle Adaboost and Logitboost are supported. The basic classifiers are decision-tree classifiers with at least 2 leaves. Haar-like features are the input to the basic classifiers, and are calculated as described below. The current algorithm uses the following Haar-like features:



image

The feature used in a particular classifier is specified by its shape (1a, 2b etc.), position within the region of interest and the scale (this scale is not the same as the scale used at the detection stage, though these two scales are multiplied). For example, in the case of the third line feature (2c) the response is calculated as the difference between the sum of image pixels under the rectangle covering the whole feature (including the two white stripes and the black stripe in the middle) and the sum of the image pixels under the black stripe multiplied by 3 in order to compensate for the differences in the size of areas. The sums of pixel values over a rectangular regions are calculated rapidly using integral images (see below and the integral description).

To see the object detector at work, have a look at the facedetect demo: https://github.com/opencv/opencv/tree/master/samples/cpp/dbt_face_detection.cpp

The following reference is for the detection part only. There is a separate application called opencv_traincascade that can train a cascade of boosted classifiers from a set of samples.

Note

> In the new C++ interface it is also possible to use LBP (local binary pattern) features in addition to Haar-like features. .. [Viola01] Paul Viola and Michael J. Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. IEEE CVPR, 2001. The paper is available online at http://research.microsoft.com/en-us/um/people/viola/Pubs/Detect/violaJones_CVPR2001.pdf

Enumeration Type Documentation

◆ anonymous enum

anonymous enum

```
#include <opencv2/objdetect.hpp>
```

Enumerator	
CASCADE_DO_CANNY_PRUNING Python: cv.CASCADE_DO_CANNY_PRUNING	
CASCADE_SCALE_IMAGE Python: cv.CASCADE_SCALE_IMAGE	
CASCADE_FIND_BIGGEST_OBJECT Python: cv.CASCADE_FIND_BIGGEST_OBJECT	
CASCADE_DO_ROUGH_SEARCH Python: cv.CASCADE_DO_ROUGH_SEARCH	

Function Documentation

◆ createFaceDetectionMaskGenerator()

```
Ptr<BaseCascadeClassifier::MaskGenerator> cv::createFaceDetectionMaskGenerator ( )
```

```
#include <opencv2/objdetect.hpp>
```

◆ groupRectangles() [1/4]

```
void cv::groupRectangles ( std::vector< Rect > &  rectList,
                           int                    groupThreshold,
                           double                 eps = 0.2
                         )
```

Python:

```
cv.groupRectangles( rectList, groupThreshold[, eps] ) -> rectList, weights
```

```
#include <opencv2/objdetect.hpp>
```

Groups the object candidate rectangles.

Parameters

rectList Input/output vector of rectangles. Output vector includes retained and grouped rectangles. (The Python list is not modified in place.)

groupThreshold Minimum possible number of rectangles minus 1. The threshold is used in a group of rectangles to retain it.

eps Relative difference between sides of the rectangles to merge them into a group.

The function is a wrapper for the generic function partition . It clusters all the input rectangles using the rectangle equivalence criteria that combines rectangles with similar sizes and similar locations. The similarity is defined by eps. When eps=0 , no clustering is done at all. If $\texttt{eps}\rightarrow +\inf$, all the rectangles are put in one cluster. Then, the small clusters containing less than or equal to groupThreshold rectangles are rejected. In each other cluster, the average rectangle is computed and put into the output rectangle list.

◆ groupRectangles() [2/4]

```
void cv::groupRectangles ( std::vector< Rect > &  rectList,
                           std::vector< int > &   weights,
                           int                    groupThreshold,
                           double                 eps = 0.2
                         )
```

Python:

```
cv.groupRectangles( rectList, groupThreshold[, eps] ) -> rectList, weights
```

```
#include <opencv2/objdetect.hpp>
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

◆ groupRectangles() [3/4]

```
void cv::groupRectangles ( std::vector< Rect > &    rectList,
                           int                      groupThreshold,
                           double                   eps,
                           std::vector< int > *     weights,
                           std::vector< double > *  levelWeights
                         )
```

Python:

cv.groupRectangles(rectList, groupThreshold[, eps]) -> rectList, weights

```
#include <opencv2/objdetect.hpp>
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

◆ groupRectangles() [4/4]

```
void cv::groupRectangles ( std::vector< Rect > &    rectList,
                           std::vector< int > &     rejectLevels,
                           std::vector< double > &  levelWeights,
                           int                      groupThreshold,
                           double                   eps = 0.2
                         )
```

Python:

cv.groupRectangles(rectList, groupThreshold[, eps]) -> rectList, weights

```
#include <opencv2/objdetect.hpp>
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

◆ groupRectangles_meanshift()

```
void cv::groupRectangles_meanshift ( std::vector< Rect > &    rectList,
                                     std::vector< double > &  foundWeights,
                                     std::vector< double > &  foundScales,
                                     double                   detectThreshold = 0.0 ,
                                     Size                     winDetSize = Size(64, 128)
                                   )
```

```
#include <opencv2/objdetect.hpp>
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Generated on Mon Jul 5 2021 14:38:35 for OpenCV by doxygen 1.8.13