

Improved Object Marker

Gunawan Herman. 04/05/06.

Introduction

Training object classifiers requires the set of positive and negative data. While a negative data can be easily obtained, collecting positive training data requires considerably much more efforts. Negative data are just those that do not contain objects of interest. For instance, for the case of training a (human) face detector, a set of animal images (that do not contain human faces) with their variable size and background, perfectly qualifies as the negative data. On the other hand, positive data usually contains as much as possible the area of objects of interest, and as little as possible the area of other objects. This means that in collecting the positive data, we have to locate and crop the objects of interest out of the 'raw' data. Manually locating and cropping the objects is impractical. A tool which can minimize the required efforts is therefore of great importance.

This Improved Object Marker program is an improved version of the Object Marker program by Adolf Florian which is freely available from <http://inflomatik.com>. The original program lacks the flexibility and control in localizing the objects in images, and this is the area where major improvements have been made. Improved Object Marker is written using Microsoft Visual Studio. However, as this program only uses the standard C++ and OpenCV libraries, porting it to Linux should not be much a problem.

This program expects that all the 'raw' images are put inside a directory called 'rawdata' which is located in the same directory where the ObjectMarker.exe is executed. The expected format of the raw images is hard-coded in the source code to JPEG format. When user marks the location of objects of interest in the 'raw' images, the program writes those annotations to a text file called 'annotation.txt' which is also located in the directory where ObjectMarker.exe resides. If the file 'annotation.txt' already exists, then the program will simply append the new annotations to the existing file. Apart from the annotations, the program also writes to the external file the date and time it is executed. Here is what the produced external file may look like:

```
#####  
Wed May 03 11:18:47 2006  
#####  
rawdata/71502865_098e872b43.jpg 1 161 122 38 19  
rawdata/71502896_7fd8fcd58d.jpg 2 215 102 24 12 123 98 25 12  
rawdata/71502923_c19a41dbec.jpg 1 198 198 53 26  
rawdata/71502997_e67e0ed332.jpg 4 131 92 24 12 210 96 24 12 273 82 27  
13 346 96 21 10  
rawdata/71503002_62a9f0551b.jpg 5 83 85 25 12 178 84 25 12 265 70 26  
13 385 83 23 11 342 100 22 11
```

Each line in produced external file is the record of how many and where the objects of interest are located in the 'raw' image. The first column contains the path of the raw image. The second column tells how many objects are marked from the raw image. The next four columns records information about the position of a marked object: the x-value of top-left corner of the bounding box, the y-value of top-left corner of the

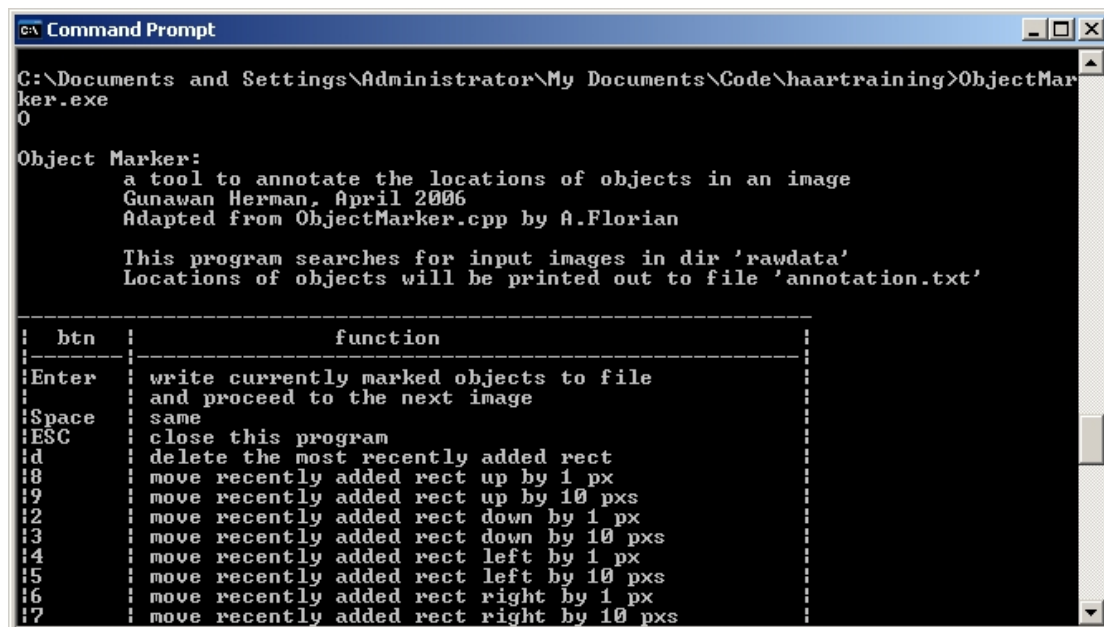
Improved Object Marker

bounding box, the width, and the height of the bounding box. For instance, from the segment of 'annotation.txt' above, we can see that there are two objects marked from 'rawdata/71502896_7fd8fcd58d.jpg', where the top-left coordinate of the first object is (215,102) and the size is 24x12, while the top-left coordinate of the second object is (123,98) and the size is 25x12.

Notice that the format of the produced external file is the same as the format of the input files that createsamples.exe expects. This is designed this way because the produced file is to be used as an input to createsamples.exe program. It is important to remove the lines other than the annotations (including the date, comments, and extra empty lines) before passing the file to createsamples.exe, or otherwise createsamples.exe will fail to parse 'annotation.txt'. Createsamples.exe is a program that comes with OpenCV distribution, which is used to create and view a .vec file. A .vec file itself is an input to the haartraining.exe program, which encodes all the positive training data.

How to use

This program is mainly a console-based application. The inputs and outputs are passed through the Command Prompt (or Terminal on Linux), except the visualisation of images which are done through the use of a separate display window. The console prompt is depicted in figure-1, while the display window is shown in figure-2, 3. The commands accepted by this application are listed in table-1. Most of the commands are straight-forward and require no detailed instructions. When invoking a command, the display window must retain the focus.



```

C:\Documents and Settings\Administrator\My Documents\Code\haartraining>ObjectMarker.exe
0

Object Marker:
  a tool to annotate the locations of objects in an image
  Gunawan Herman, April 2006
  Adapted from ObjectMarker.cpp by A.Florian

  This program searches for input images in dir 'rawdata'
  Locations of objects will be printed out to file 'annotation.txt'

-----
| btn | | function |
-----
|Enter| | write currently marked objects to file |
|      | | and proceed to the next image |
|Space| | same |
|ESC  | | close this program |
|d     | | delete the most recently added rect |
|8     | | move recently added rect up by 1 px |
|9     | | move recently added rect up by 10 pxs |
|2     | | move recently added rect down by 1 px |
|3     | | move recently added rect down by 10 pxs |
|4     | | move recently added rect left by 1 px |
|5     | | move recently added rect left by 10 pxs |
|6     | | move recently added rect right by 1 px |
|7     | | move recently added rect right by 10 pxs |
-----
```

Figure-1: console prompt

Key	Function
Enter or Space	Record objects marked on current 'raw' image to 'annotation.txt' and proceed to the next 'raw' image
ESC	Quit the program
d	Delete the most recently marked object (represented by a bounding box). Doing this many times will delete one by one from the most to the least recently added objects.
8	Move last added rounding box up by 1 pixel
9	Same as above, but by 10 pixels
2	Move last added rounding box down by 1 pixel
3	Same as above, but by 10 pixels
4	Move last added rounding box left by 1 pixel
5	Same as above, but by 10 pixels
6	Move last added rounding box right by 1 pixel
7	Same as above, but by 10 pixels
w	Increase width of last added rounding box by 1 pixel
W	Reduce width of last added rounding box by 1 pixel
h	Increase height of last added rounding box by 1 pixel
H	Reduce height of last added rounding box by 1 pixel
z	Increase size of last added rounding box by 2%
Z	Reduce size of last added rounding box by 2%
m	Switch between free-size and fixed-scale mode, or to lock-in current width-height ratio
s	Input the scale ratio for fixed-scale mode
p	Enable marking objects by points instead of bounding boxes
t	Print usage instruction
j	Jump to 'raw' image with the specified index

Table-1: list of recognized commands.

An object is usually marked by putting a bounding-box around it. There are two possible modes to create a bounding box: *free-size* and *fixed-scale* modes. Free-size mode is when the width-height ratio of the created bounding-box is not forced to follow a certain predetermined ratio, as it is in fixed-scale mode. It is sometimes desirable to mark objects with free width-height ratio, but for object detection training, usually the width and height of the bounding-box follows a predetermined ratio. For instance, in training face detector, we use 24x24 positive training data, and to get those data, we may put a rounding box whose width:height ratio is 24:24 (i.e. a square). This object marking mechanism is shown in figure-2. The width-height ratio can be defined in two ways. User can either specify the width-height ratio directly or by mimicking the width-height ratio of a bounding-box that the user creates.

Apart from bounding the object of interest with a rectangle, there is another mechanism that we can use to mark an object. We can place *points* to mark certain location of interest, and then use the location of the points for further post-processing. User has control over the number, the location, and the order of the points placed in the raw image, as well as the post-processing of the records. Post-processing is usually carried out through a separate, small program which crops the objects of

interest based on the recorded points. Figure-3 shows how each face is marked by two points: one just above the eyebrows, another one in the mid-way between lower-lip and chin. We use this mechanism to gather positive training data for our face detector training. For more information, please see our report '*Training_Face_Detector_Using_OpenCV*'.



Figure-2: objects marked by bounding boxes

Choices on how to mark the objects of interest are important. In some cases user may find that *bounding-box* mechanism is more desirable, while in some other cases the *points* mechanism is preferred. From my experience, points mechanism is a better choice when some rules regarding the cropped objects need to be imposed. For instance, it might be required that the eyes are located in the centre along the horizontal axis, the top of the bounding-box must be above the eyebrows, and the bottom of the bounding-box must be below the lower-lip, etc. Bounding-box mechanism is a preferred choice when the objects of interest are too small to be marked accurately by points, for instance, when we want to crop human eyes from the raw images. In most cases, points mechanism is more efficient than bounding-box mechanism.

User is given various controls while placing a bounding-box around the object of interest. A bounding box can be navigated and resized around the object of interest so that it is located in the most appropriate position and covers the desired area. A bounding-box can be maintained at the same width-height ratio while increasing or reducing its size by zooming it in or out. The width and the height can also be increased explicitly.



Figure-3: objects marked by points

This application, apart from being used to mark objects of interest, can also be used to crop the objects of interest based on the information in the file 'annotation.txt'. When the program is invoked without any parameters, the program functions as an object marker, but when the program is invoked with a parameter (any parameter at all, for instance "ObjectMarker.exe -"), the program crops the objects out of the raw images based on the information in the file 'annotation.txt' and place the cropped images inside directory 'cropped' which resides in the same directory where ObjectMarker.exe is invoked. If the directory 'cropped' does not exist, user must create one manually. The produced images are in JPEG format and the size is hard-coded in the source code.

Limitations and possible improvements

There are a number of improvements that can be made to this program. The most important one is probably to remove the hard-coded constraints (location and format of raw images, location and name of produced annotation file, location, format, and size of produced cropped images) and enable the user to specify them during runtime. The second possible improvement is to add a graphical user interface to the program in order to enhance the program's usability. Another improvement can be made by making the command-key bindings more intuitive.