Object Detection using Selective Search & EdgeBoxes algorithm in OpenCV

By Agnish Sahu, Shambhavi Rai

Objective

This project aims to experiment with two methods (Selective Search and EdgeBoxes) to generate object proposals. We will use OpenCV library to test these methods on some sample color images.

Environment Setup

Python Version = 3.6

OpenCV Version = 4.1.1

Code Execution

Selective Search

To run the code, download it and execute using the following command in terminal/command prompt:

```
python selective_search.py <input_image_path> <annotated_image_path> <strategy>
```

- input_image_path: Enter the image file name including path
- annotated_image_path: Enter the annotated image file name including path
- strategy: Enter the strategy – 'color' for color strategy, 'all' for all strategies

Eg: python selective_search.py ./JPEGImages/1.jpg ./Annotations/1.xml color

EdgeBoxes

To run the code, download it and execute using the following command in terminal/command prompt:

```
python edgeboxes.py <input_image_path> <annotated_image_path>
```

- input_image_path: Enter the image file name including path
- annotated_image_path: Enter the annotated image file name including path

Eg: python edgeboxes.py ./JPEGImages/1.jpg ./Annotations/1.xml

Data

The sample color images are present in `/JPEGImages/` folder and their corresponding ground truth boxes are present in `/Annotations/` folder. An example of the bounding box coordinates (saved in .xml file) is in the following format:

```
<object> <name>aeroplane</name> <pose>Unspecified</pose> <truncated>0</truncated> <difficult>0</difficult> <bndbox> <xmin>68</xmin> <ymin>76</ymin> <xmax>426</xmax> <ymax>209</ymax> </bndbox> </object>
```

where the object label is “aeroplane”, and the bounding box is described in the format of (xmin, ymin, xmax, ymax) = (68, 76, 426, 209), namely, the left-top and right-bottom points of the box.

Code Information

Selective search

The Selective Search(SS) algorithm is a hierarchical algorithm which is used for object detection. In this project, we experiment with two strategies: color only and all similarities (color, fill, size, texture). We restrict the algorithm to use only the RGB space. More information about Selective Search can be found in the paper (https://www.researchgate.net/publication/262270555_Selective_Search_for_Object_Recognition)

The steps followed in implementing the code are as follows:

1. Read the image specified in the input_image_path (By default OpenCV reads it in BGR format)
2. Create a SS object by calling createSelectiveSearchSegmentation()
3. Convert the image to RGB format and add it using addImage()
4. Create a graph based segmentor object (gs) using createGraphSegmentation().Set the sigma parameter and K parameter values of this object.
5. Add the graph segmentor object to the ss object using addGraphSegmentation()
6. Based on the strategy parameter in the input command, use either the color strategy or all strategies (colour, texture, size, fill)
7. Add the required strategy using addStrategy() command
8. Get all the proposed output bounding boxes using the process() command
9. Select only the top 100 proposed bounding boxes
10. Each of the proposed bounding boxes obtained from the process command() are of the form (x, y, w, h) where (x, y) are the coordinates of the top left corner and (w, h) are the width and height of the bounding box. Transform the (x, y, w, h) parameters of each bounding box into (x, y, x+w, y+h) parameters where (x, y) indicate the coordinates of the top left corner and (x+w, y+h) indicate the coordinates of the bottom right corner.
11. Display the image showing the top 100 proposed bounding boxes.
12. Parse the annotated image file (annotated_image_path parameter) provided in xml format to obtain the ground truths bounding boxes. I have used the ElementTree xml API to parse the xml file. Each annotated image file has the parameters (x1, y1, x2, y2) where

(x1, y1) indicates the coordinates of the top left corner and (x2, y2) indicates the coordinates of the bottom right corner.

13. Fetch all proposed bounding boxes that have IoU > 0.5 with any of the ground truth boxes and display them against the ground truth boxes.
14. Find those bounding box that have the maximum/best overlap for each ground truth box and display them.
15. Calculate the recall for each image.

EdgeBoxes

The Edge Boxes algorithm uses edges to detect objects. The overlying idea of this algorithm is that if a bounding box contains a large number of entire contours, then the bounding box has a high probability to contain an object. More information about EdgeBoxes can be found in the paper (https://www.researchgate.net/publication/319770284_Edge_Boxes_Locating_Object_Proposals_from_Edges)

The steps followed in implementing the code are as follows:

1. Read the image specified in the input_image_path (By default OpenCV reads it in BGR format)
2. Download the edge detection model from the below link: https://github.com/opencv/opencv_extra/blob/master/testdata/cv/ximgproc/model.yml.gz or use the model (model.yml.gz) available along with the code.
3. Create a edge detection object using createStructuredEdgeDetection()
4. Convert the image from BGR format to RGB format
5. Detect the edges in the image and create an orientation map. Suppress the edges using Non-Maxima Suppression.
6. Create an edgeboxes object by calling createEdgeBoxes() . Set the alpha value to 0.5 and beta value to 0.65
7. Fetch the proposed bounding boxes and their corresponding scores using getBoundingBoxes()
8. Select only the top 100 proposed bounding boxes with the highest scores
9. Each of the proposed bounding boxes obtained are of the form (x, y, w, h) where (x, y) are the coordinates of the top left corner and (w, h) are the width and height of the bounding box. Transform the (x, y, w, h) parameters of each bounding box into (x, y, x+w, y+h) parameters where (x, y) indicate the coordinates of the top left corner and (x+w, y+h) indicate the coordinates of the bottom right corner.
10. Display the image showing the top 100 proposed bounding boxes.
11. Parse the annotated image file (annotated_image_path parameter) provided in xml format to obtain the ground

truths bounding boxes. I have used the ElementTree xml API to parse the xml file. Each annotated image file has the parameters (x1, y1, x2, y2) where (x1, y1) indicates the coordinates of the top left corner and (x2, y2) indicates the coordinates of the bottom right corner.

12. Fetch all proposed bounding boxes that have IoU > 0.5 with any of the ground truth boxes and display them against the ground truth boxes.
13. Find those bounding box that have the maximum/best overlap for each ground truth box and display them.
14. Calculate the recall for each image.
15. Experiment with different values of alpha and beta.

Evaluation Metric

Intersection over Union

We evaluate the proposed bounding boxes generated by Selective Search and EdgeBoxes algorithms using Intersection over Union (IoU) metric which is defined as:

```
Intersection over Union (IoU) = Area of overlap / Area of union
```

Here, Area of overlap is the area of intersection between the ground truth box and the proposed bounding box. Area of Union is the total area encompassed by both the ground truth box and the proposed bounding box. If the ground truth box and the proposed bounding box do not overlap, then the IoU is 0.

Recall

We calculate the "recall" of ground truth bounding boxes by computing what fraction of ground truth boxes are overlapped with at least one proposal box with Intersection over Union (IoU) > 0.5.

Analysis

My analysis on these algorithms are listed below for one of the images (Image 5 in the *Results.pdf* file)

Selective Search:

The test image shown in Appendix A - Image 5 has been annotated with 13 ground truth boxes. The results obtained using color strategy only have a total of 334 proposed bounding boxes. The number of proposal boxes found to have an IoU greater than 0.5 with any of the ground truth boxes is six. Out of these six proposal boxes, only four ground truth boxes have an overlap with atleast one proposal box. The recall for this image using only color strategy is 0.307.

The results obtained using all strategies i.e. color, fill, texture and size contain a total of 354 proposed bounding boxes. The number of proposal boxes found to have an IoU greater than 0.5 with any of the ground truth boxes is eight. Out of these eight proposal boxes, only seven ground truth boxes have an overlap with aleast one proposal box. The recall

for this image using all strategies is 0.5384.

In general, Selective Search using all strategies had a better performance (better recall) when compared to just the color strategy. In our particular image (Image 5) that we considered, we can notice that the image has a variety of colors. Since our code utilizes only the top 100 proposal boxes of relevance, the color strategy is not effectively able to detect all objects of interest (detects 4), where as when the algorithm uses all strategies, we are able to get significantly better results (detects 7). If we expand the number of proposal boxes from 100 to a larger number (say 1000 or 2000), we are able to get a better performance/recall (0.92/1.0) for both strategies. In general, utilizing selective search algorithm with all strategies would give a better performance than using just the color strategy. Additionally, sometimes the results using color algorithm could be good as well depending upon the image we are trying to run the algorithm on.

EdgeBoxes:

The test image shown above in Appendix B - Image 5 has been annotated with 13 ground truth boxes. On selecting the proposed bounding boxes with the top 100 scores, we found that the number of proposal boxes that have an IoU greater than 0.5 with any of the ground truth boxes is 12. Out of these 12 proposal boxes, only four ground truth boxes have an overlap with atleast one proposal box. The recall for this image using edgeboxes algorithm is 0.307.

Following are a few cases of experimenting with different values of alpha and beta:

Case 1: Low alpha and low beta values (< 0.2) gave the worst performance with no proposal bounding boxes being found that had an IoU > 0.5.

Case 2: Setting alpha and beta as 0.5 gave the performance in terms of recall. The algorithm found slightly lesser bounding boxes that had an IoU > 0.5 (detected between 7-10) but the number of ground truth boxes that had an overlap with atleast one proposal box was much higher (detected 6).

Case 3: Setting alpha and beta as 0.8 had a poorer recall than case 2 , but it generally had more proposal boxes that had an IoU > 0.5 (detected 13 approximately). The number of ground truth boxes that had an overlap with atleast one proposal box was around 3.

Setting alpha as 0.5 and beta as 0.5 seemed to give me the best performance in terms of recall.

Comparison between EdgeBoxes and Selective Search:

EdgeBoxes for this particular image (Image 5) did not give a significantly improved performance than selective search. However looking at all the images in appendix A and appendix B, edgeboxes in general gave a better performance than selective search. I

noticed that the proposal boxes generated using edgeboxes were larger in size than the proposal boxes using selective search algorithm because the edgeboxes algorithm tried to detect those boxes which have a lot of contours inside them and scored them highly. So the top 100 scores assigned by the edgeboxes algorithm were generally the larger bounding boxes and thus they had a higher probability of having an IoU > 0.5 with any of the ground truth boxes.

Experimenting with selective search algorithm, I noticed that the top 100 proposed bounding boxes returned by ss.process() command were generally small in size, and thus they had a lower probability of having an IoU > 0.5 with the ground truth boxes, and thus a lower recall. However if I considered the bottom 100 bounding boxes returned by ss.process(), they gave better bounding boxes and a much higher recall than the top 100 bounding boxes.