Superpixel Grouping Based Object Localization Algorithm

Serhan Gürsoy
Computer Science
Ihsan Dogramacı Bilkent University
Ankara, Turkey
serhan.gursoy@ug.bilkent.edu.tr

Ege Yosunkaya
Computer Science
Ihsan Dogramacı Bilkent University
Ankara, Turkey
ege.yosunkaya@ug.bilkent.edu.tr

***Abstract*—Computer vision applications commonly using data sets for training object recognition systems, sets that include images which are tightly cropped around objects. This approach decreases the running time and increase recognition accuracy by providing best characteristics of the objects of interests to the learning algorithms. In an effort to create best test set for learning algorithms, we decided to use superpixel grouping based object recognition. Resulting objects can be used for training the learning algorithms. While this approach can be efficient for complex recognition algorithms which will focus only proposed objects from our algorithm rather than calculating the objects by itself, it can also be powerful and efficient for other but rather simple computer vision applications as well.**

Keywords— object proposals, object detection, edge detection, superpixels

I. Introduction

Superpixel algorithms group pixels into a meaningful region which can be processed depending on the goal of computer vision application. Our goal was reshaping the resulted superpixels in a way that we can identify the location of objects of interest and create meaningful location data for learning algorithms.

There are many approaches to generate superpixels, where each have their advantages as well as disadvantages. In this paper, we decided to use *simple linear iterative clustering*(SLIC) algorithm to create our superpixels. Example outputs with different parameters shown in Figure 1. By considering several features such as Color and Texture feature, edge response feature and graph distance of superpixels to each other. Our algorithm merges related superpixels to result a less complicated and processable superpixel sets. Afterwards, our algorithm scores each object proposal based on edge response results and creates bounding boxes for each recognized object. Later, any learning algorithm can use resulted boundary box locations and shift their focus on that area only in order to decrease processing requirement for each image.

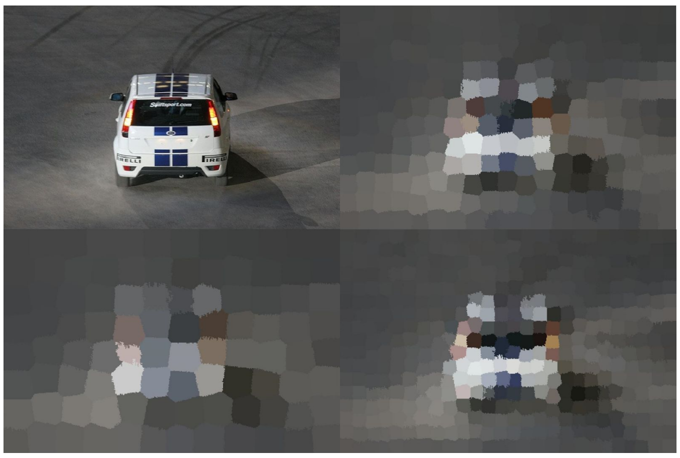


Figure 1. SLIC Results with different pixel limits. (Top left) Input Image, (Lower left) Pixel limit 100 as SLIC parameter, (Top right) Pixel limit 250, (Lower right) Pixel limit 450 as SLIC parameter.

II. Approach

Before diving into implementation of our algorithm, we discussed about the parameters that we will evaluate while we are merging the resulted SLIC superpixels. After concluding that feature selection is dependent to the goal in Computer Vision applications, in order to get best results, we decided to approach this issue by considering same features in *Complexity-Adaptive Distance Metric for Object Proposals Generation* paper [1].

A. Texture and Color Feature Distance

Texture and color features are low-level features for describing regions. Regardless to their low-levelity, texture and color features quite important for our decision algorithm in merging. Similar superpixels shares similar texture and color features, hence, it is important to merge them for better segmentation and classification later on.

As proposed in [1], for construct our color feature distance, we create one-dimensional color histogram with 20 bins per color channel. Furthermore, for texture distance feature, we take Gaussian derivatives in eight orientations forming a normalized texture histogram h_t with 10 bins. These bins are for each color channel per orientation. In order to construct a color and texture distance between two superpixels, we come up with the following *equation 1*,

$$(1)\quad d_{ct}(i,j) = \|h_c(i) - h_c(j)\| + \| h_t(i) - h_t(j)\|$$

where $d_{ct}(i,j)$ simply describes distance, $h_c(i)$ color feature and $h_t(i)$ texture feature.

B. Spatial Distance

Directly relying on spatial distance and local neighboring in superpixel merge operation might not be the best solution. However, once spatial distance and local neighboring used with other decision parameters, such as proposed in this paper, it becomes rather useful and selective in merging. Hence, we decided to use graph distance in order to keep genuine image spatial structure.

From [2], we construct a connectivity graph G based on our initially segmented superpixels. The distance between each pair of nodes $d_g(i,j)$ calculated with Floyd-Warshall algorithm. We also implemented Dijkstra's algorithm however we removed it from actual code because of the time-complexity issue. The Floyd-Warshall all-pairs shortest path runs in $O(n^3)$ time, which is not better than Dijkstra's algorithm asymptotically. However, it runs better in practice because of the loop tightness. Additionally, to speed up processing, rather than processing each superpixel labels, we just considered *border* pixels of superpixels. For large superpixels, this approach saves significant time as well as processing power. Moreover, because this spatial distance matrix should be symmetric according to the spatial properties, we also implemented our function in best optimized manner to eliminate unnecessary calculations. The real graph distance between superpixels defined as D_g in following equation,

$$(2)\quad D_g(m,n) = min\{d_g(i,j) | \, i \in S_m, \, j \in S_n\}$$

Resulting D_g is a symmetric matrix which contains distance data of all possible superpixel pairs. This approach preserves original spatial distances of superpixels. By searching in a larger area beyond a local region, highly-complex object segments with similar areas can be merged easily.

C. Edge Cost

In order to get edge responses, we first used Prewitt's algorithm[3] to obtain our edge map. In most simplistic form, we can define edge cost as the edge responses along the common border of the segments. The edge cost for each neighboring segment is calculated by absolute value of difference between their change. If they are equal, means that either they both on the edge or at non-edge pixel, then difference will give us zero which has no value over our final result. After finding every neighbor, our algorithm increases the common border value which will be used in the normalization. Following equation shows the calculation of edge cost,

$$(3)\quad d_e(i,j) = \begin{cases} \frac{1}{|l_{i,j}|} \sum_{p \in l_{i,j}} E(p) & \text{if } |l_{i,j}| \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Moreover, edge cost between two sets of superpixels can be defined as follows,

$$D_e(m,n) = \begin{cases} \frac{\sum_{i \in S_m, j \in S_n} |l_{i,j}| d_e(i,j)}{\sum_{i \in S_m, j \in S_n} |l_{i,j}|} & \text{if } \sum_{i \in S_m, j \in} l_{i,j} \neq 0 \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

In *equation 4*, we can see how *equation 3* used to determining edge cost between superpixel labels. In homogeneous regions, edge response are expected to be low, however in high-complexity superpixel sets it is expected to be higher.

III. Experiments

A set of 100 images from PASCAL Visual Object Classes used in this paper to obtain experiment results. In PASCAL Visual set, each image contains one or more bounding boxes and their class categories. Such as cat, cow, dog, sheep etc. Bounding boxes and our precision results will be discussed in the evaluation part. In order to observe our merging process more properly, we made some examples about different individual merging processes.

A. Individual Distance Examples

- In figure 2, it is shown that how merging process behaves by looking at graph distance D_g. However, only using graph distance would result a big frame that contains all superpixels because in every single iteration, sets getting close to each others, hence, for this experiment only, we also looked at the color distance feature of the superpixels as well before merging them. In the light of this results, we can clearly state that only looking to the graph distance is not the best approach if it done individually.

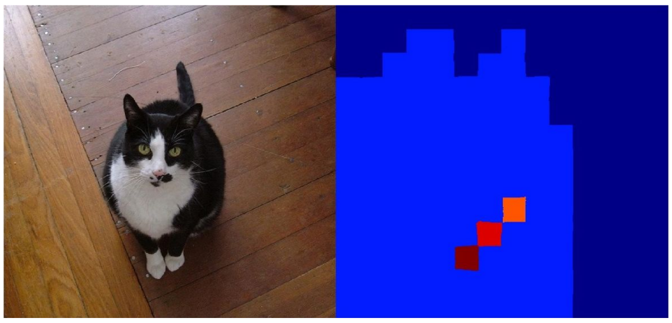


Figure 2. Input image (left), resulted superpixel labels visualization (right) by looking at the Graph Distance feature.

- In figure 3, we merged our superpixel sets only by evaluating the Texture feature. In this example, we can observe the behaviour of our texture distance feature in most simplistic way. As it can also be observed intuitively, car object has some identical, homogeneous texture on its body part. Therefore our merging operation merged them in one single set where ground and sky have some coarse texture pattern which also merged in same set without looking at their color feature or spatial distance. This approach might give close and satisfying results for an average computer vision application but these results are not enough for complex computer vision applications and accuracy of these results are dependent to the content of image, therefore, it should not be used individually.

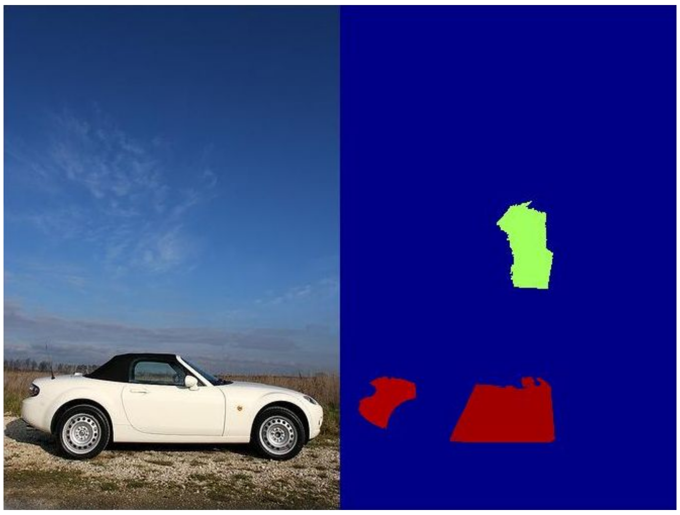


Figure 3. Input image (left), resulted superpixel labels visualization (right) by looking at the Texture feature. Each color represents a superpixel set. Red color represents the smooth texture property while green color represents semi-complex but still fairly smooth texture property and blue color represents the complex texture with coarse

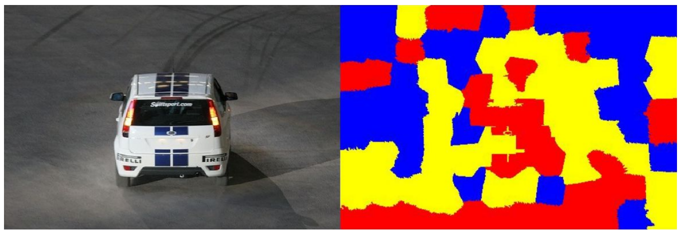


Figure 4. Input image (left), resulted superpixel labels visualization (right) by looking at the Edge response.

- In figure 4, we completed our superpixel set merging operation by considering edge responses only. In order to get our edge responses, we used Prewitt's algorithm [3]. While constructing right side at Figure 4, we did not altered the edge map with morphological operations such as dilation and erosion. However, resulted edge map contains some noise which effects the behaviour of merging process. In figure 5, noise can be observed. Merging process merged unrelated superpixels in same set because of the noisy edge response obtained from the input image.

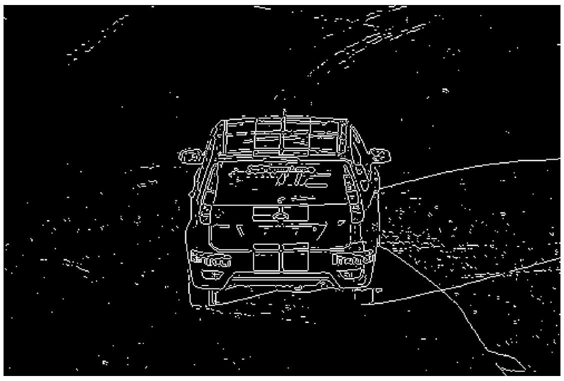


Figure 5. Prewitt edge response result on same input image in Figure 4. Noise in this result causes faulty merge operations which can be seen in Figure 4.

- In figure 6, we completed our superpixel set merging operation with using D_{max} and D_{min} individually. As stated in [1], both D_{max} and D_{min} calculates the texture-color distance between two superpixel sets and while D_{max} takes maximum value, D_{min} takes minimum value. According to the experiments and conclusion in [1], we were also expecting to get better results for D_{min} in complex sets. By comparing our results in Figure 6, we can also conclude that D_{min} also returns more accurate results than D_{max}. However, we can also state that these results are not satisfying for many complex computer vision applications because of their low accuracy.

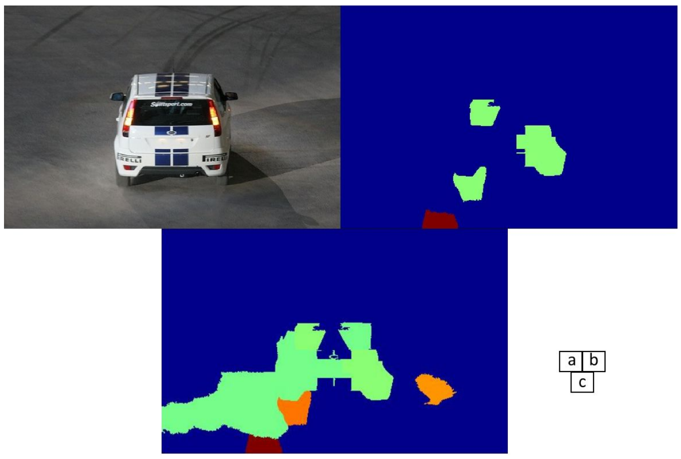


Figure 6. Test results for D_{min} and D_{max}. Input image, with single car object (a), merging based on D_{max} results(b), merging based on D_{min} results.

IV. Results

In this section, we will discuss results we obtained from our algorithm. We will first test our algorithm with the 100 images from PASCAL Visual Object Classes. Then we will compare our results by comparing them with the real/expected results, again, obtained from PASCAL Visual set. We will sub-categorize them according to their belonged class, such as cat, television monitor, dog etc. Red bounding box indicates our detection where green bounding box indicates real/expected result. Images below selected randomly from each class.

A. Testing Algorithm on Cat Class

In this section we will test our algorithm on Cat class with two different input images.

Algorithm working accurate with the cat class, the backgrounds of the images we tested with were plain thus algorithm did not get confused with the edge costs.

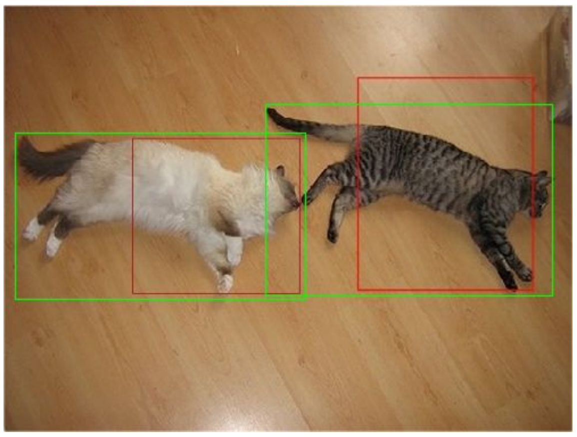


Figure 7. Test result with 2 cats

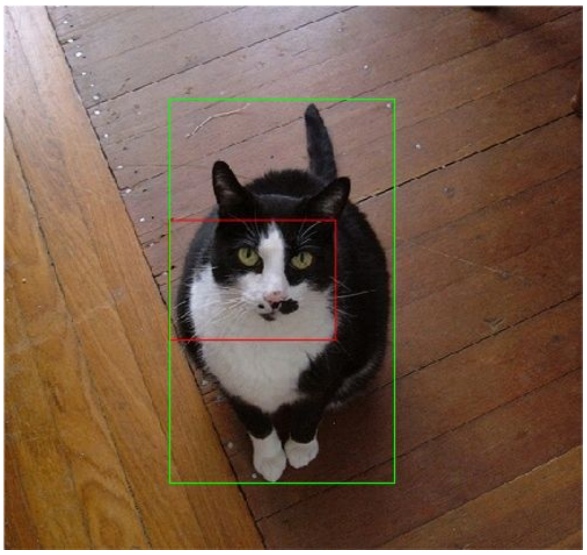


Figure 8. Test result from cat image input with single cat object.

B. Testing Algorithm on Dog Class

In this section we will test our algorithm on Dog class with two different input images.

As we can observe in both Figure 9 and Figure 10, like in cat class, algorithm performed highly accurate results in dog class as well. While knowing that they are similar objects in real life, this result is not surprising in terms of detection.

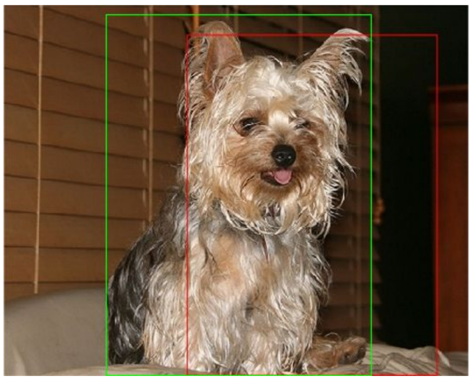


Figure 9. Detection result with single dog object input image.

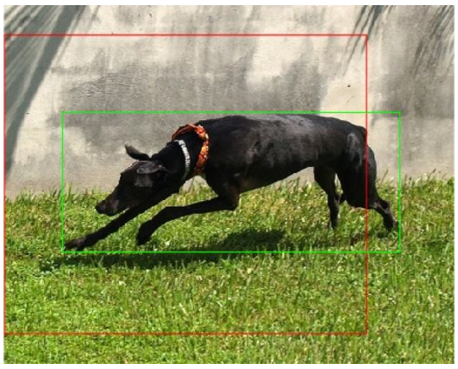


Figure 10. Detection result with single dog object input image.

C. Testing Algorithm on Cow Class

In this section we will test our algorithm on Cow class with two different input images.

In Figure 11 we can observe over-detection, which most probably caused because of the cow's texture pattern that looks similar to the background whereas we can not observe over-detection in Figure 12 because cows in that input image does not share similar properties with their background in terms of texture pattern.

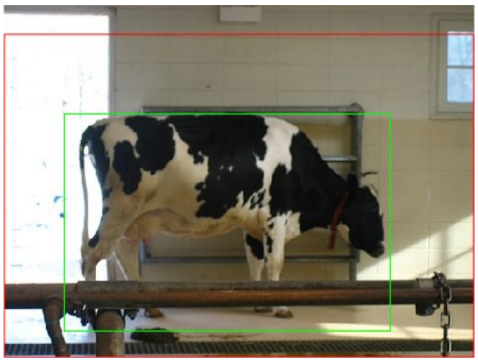


Figure 11. Detection result with input image that contains single cow object.

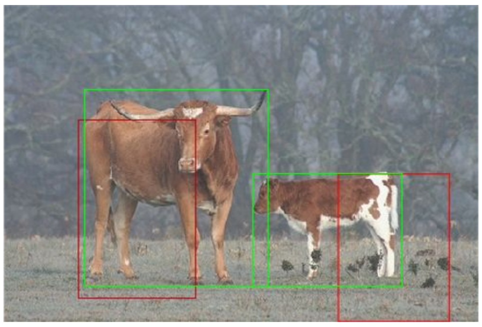


Figure 12. Detection result with input image that contains single two object.

D. Testing Algorithm on Horse Class

In this section we will test our algorithm on Horse class with two different input images.

As we can observe in Figure 13, our algorithm detected rider instead of detecting directly horse itself. Furthermore, we observe that our algorithm also ignored the low-scored horses in the scene as well. This problem occurred because our algorithm is not pre-set to the horses but object of interest. In Figure 14, we see that algorithm again confuses rider and bars as object of interest and over-detection occurs.

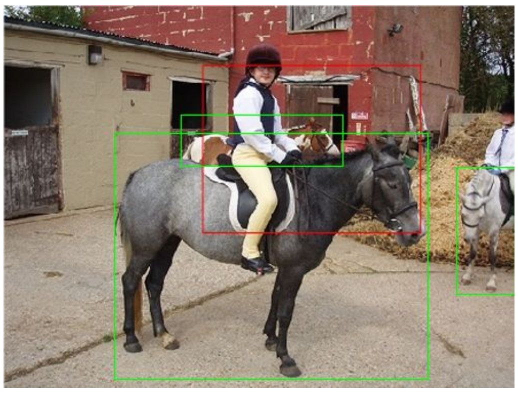


Figure 13. Detection result with input image that contains three horse object.

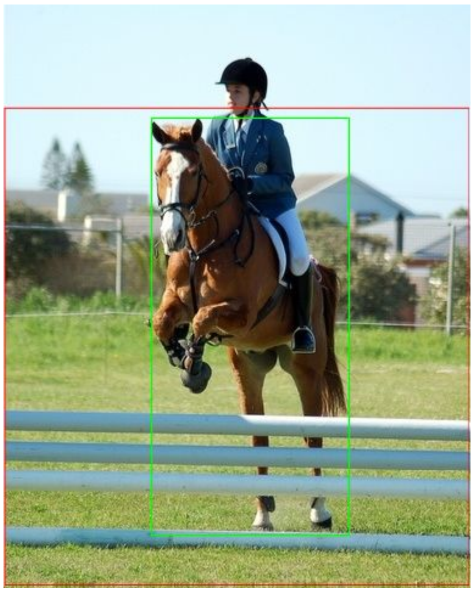


Figure 14. Detection result with input image that contains single horse object

E. Testing Algorithm on Sheep Class

In this section we will test our algorithm on Sheep class with two different input images.

As we can observe in Figure 15 and Figure 16, sheep detection did not resulted as expected. In Figure 15, sheep detection failed most probably because of the texture pattern on input sheep. Moreover, in Figure 16, our algorithm confuses grass with sheep and covers all grassland as object.

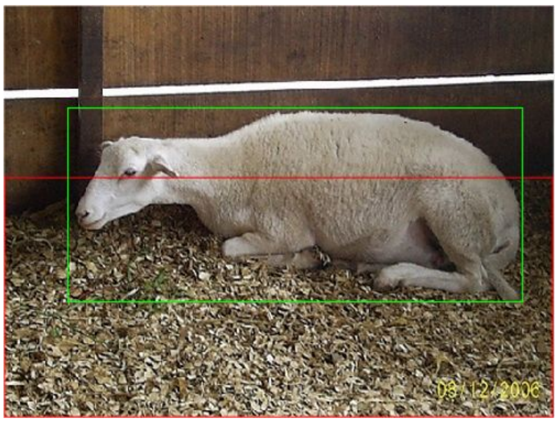


Figure 15. Detection result with input image that contains single sheep object

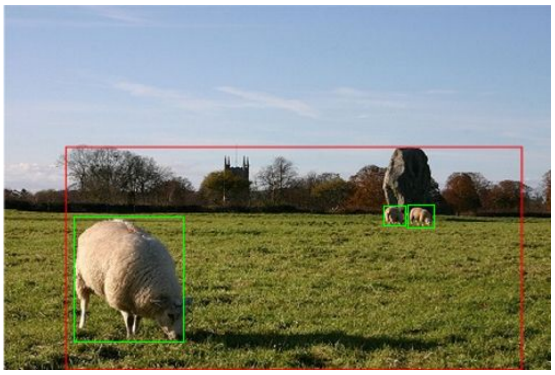


Figure 16. Detection result with input image that contains three sheep object

F. Testing Algorithm on Boat Class

In this section we will test our algorithm on Boat class with two different input images.

In both Figure 17 and Figure 18, we observe fairly good detections compared to expected results. However while in Figure 17 our algorithm does over-detection, in Figure 18 it does under-detection by ignoring 30 percentage of boat.

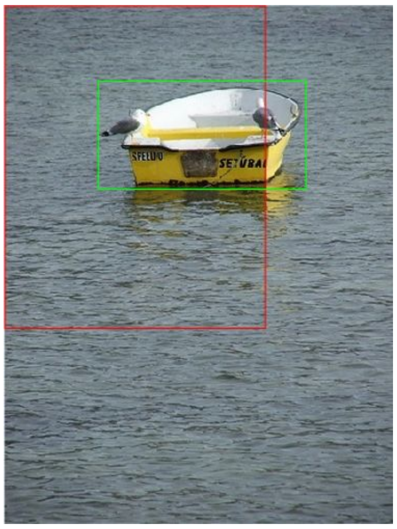


Figure 17. Detection result with input image that contains single boat object

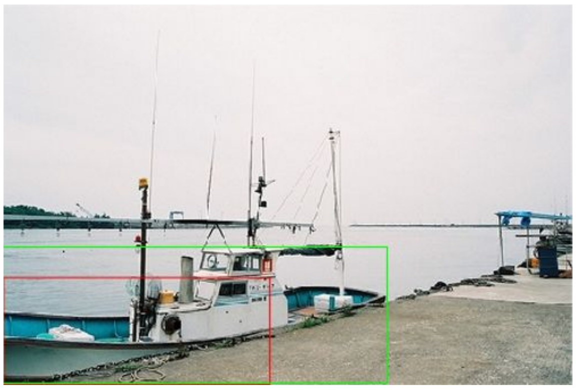


Figure 18. Detection result with input image that contains single boat object

G. Testing Algorithm on Bus Class

In this section we will test our algorithm on Bus class with two different input images.

In Figure 19, we can observe that our algorithm merged car object and bus object into single bounding box but it also identifies fire hydrant as object of interest as well. In Figure 20, we again observe that our algorithm merges all car and bus superpixels and creates a single bounding box to cover it all.

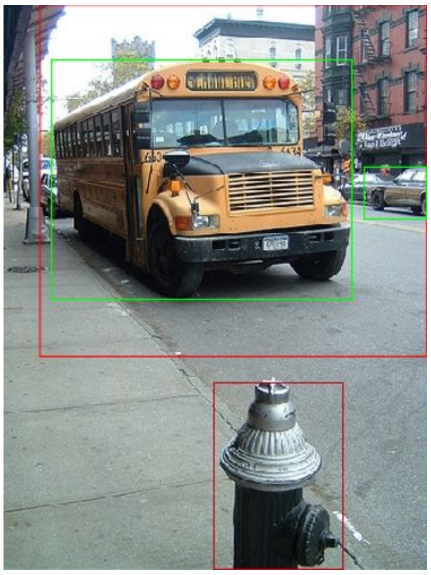


Figure 19. Detection result with input image that contains single bus object and single car object

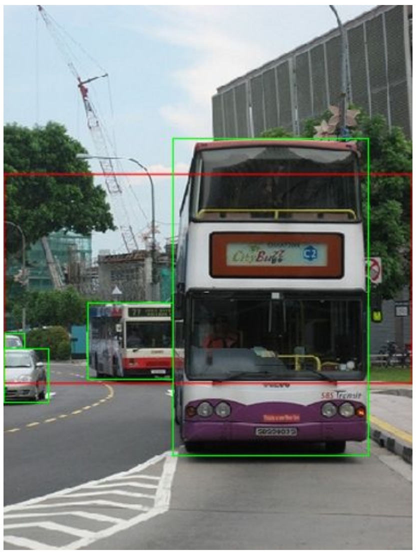


Figure 20. Detection result with input image that contains two bus object and two car object

H. Testing Algorithm on Car Class

In this section we will test our algorithm on Car class with two different input images.

In both Figure 21 and Figure 22 we observe that our algorithm does an accurate job by detecting the both cars' location properly however it was not able to detect their size correctly.

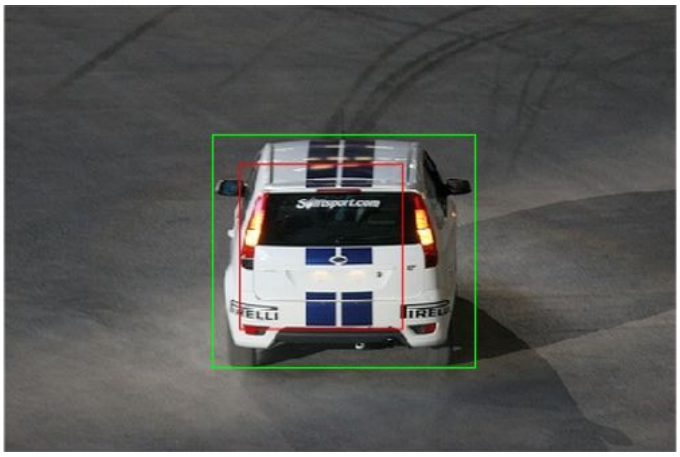


Figure 21. Detection result with input image that contains single car object

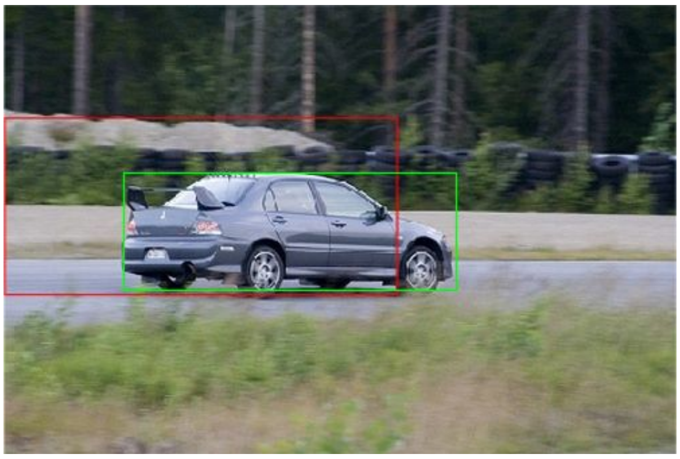


Figure 22. Detection result with input image that contains single car object

I. Testing Algorithm on Train Class

In this section we will test our algorithm on Train class with two different input images.

In Figure 23, we observe that our algorithm correctly identifies one train with a tiny error margin however it fails to find most clear train most probably because of its color feature similarity with the rails. In Figure 24, our algorithm does a poor job and identifies only a small portion of desired train object.

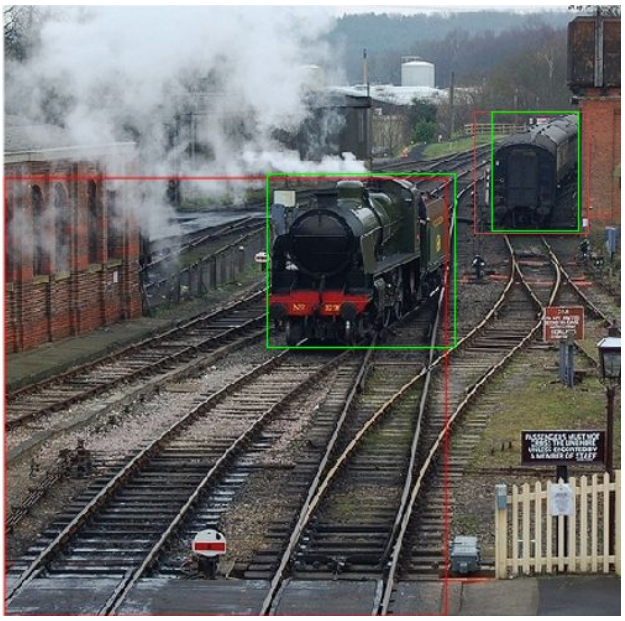


Figure 23. Detection result with input image that contains two train object

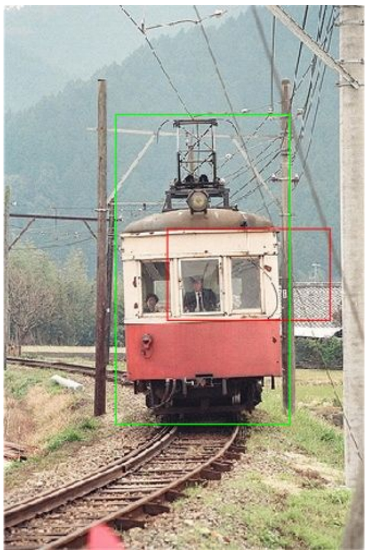


Figure 24. Detection result with input image that contains single train object

J. Testing Algorithm on TV Monitor Class

In this section we will test our algorithm on TV Monitor class with two different input images.

In both Figure 25 and Figure 26 our algorithm does a really fine job by identifying the computer screens. However, in Figure 25, because of the contrast property of one screen, it fails to detect.

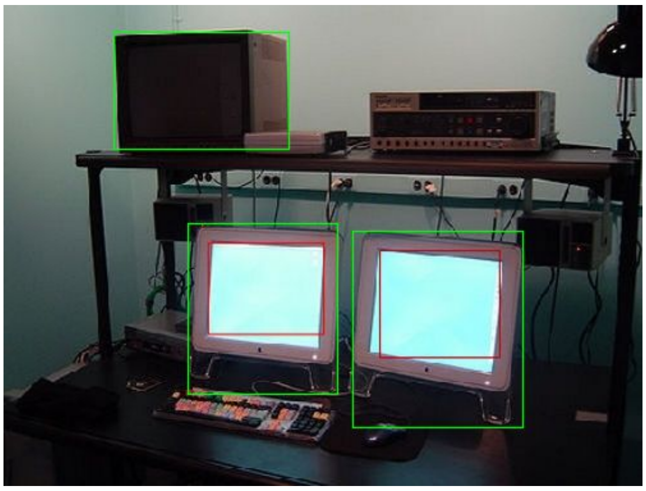


Figure 25. Detection result with input image that contains three monitor object

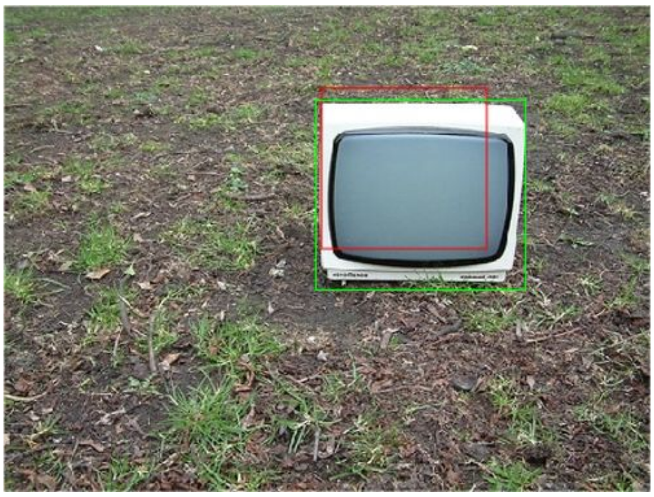


Figure 26. Detection result with input image that contains single monitor object

K. Comparing Results

In this section we will compare results obtained from different classes and we will discuss why some of them are accurate while some are not.

The overall accuracy of the algorithm is mostly affected by the background noise. The crowded backgrounds with edges were the most significant factor that confuses the algorithm. Especially in train class, the rails and the poles near the train made the bounding boxes bigger than it should be.

Moreover in the screen class, the box covering the screen was maximizing the edge cost thus algorithm scored the bounding boxes that covers only the inside with higher values.

The car and cat classes were more successful with respect to the other classes , because the images had plain backgrounds and big objects.

The dog class done better for the plain backgrounds and worse for the moving or small dogs. Moreover, the bus class had less accuracy compared to the car class because the most of the bus images had more than one bus or additional cars in them.

The sheep and cow classes had similar accuracies , the accuracy again depends on the number of objects in the picture and the background. Moreover, in some horse pictures the riders made the detected bounding boxes bigger than real bounding boxes.

L. Precision and Recall

In Figure 27, we present precision results obtained after each class processing with our algorithm. While evaluating set, we used the following formulas for precision and recall.

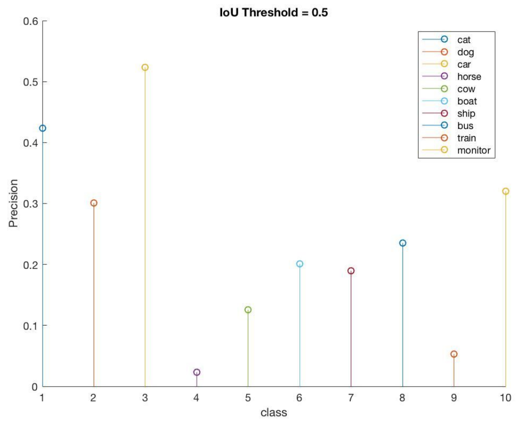
$$(5)\ Precision = \frac{N_{cdo}}{N_{do}}$$

$$(6)\ Recall = \frac{N_{cdo}}{N_{aog}}$$

Where N_{cdo} is number of correctly detected objects, N_{do} number of detected objects, N_{aog} number of all objects in the ground truth. A detection considered true if it's Intersection of Union is greater than 0.5. IoU (Intersection of Union) can be defined as follows,

$$(7)\ IoU = \frac{A_o}{A_u}$$

where A_u is *Area of Union* and A_o is *Area of Overlap,* between two bounding box. In Figure 28, we can examine the effect of threshold over evaluation.



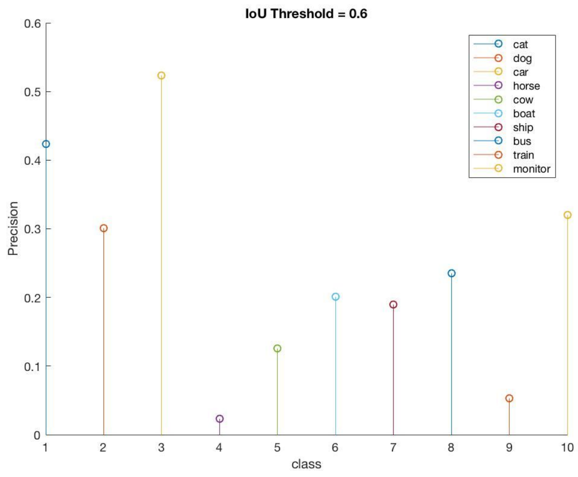


Figure 27. Precision based on class. IoU threshold 0.5 (Top), IoU threshold 0.6 (bottom)

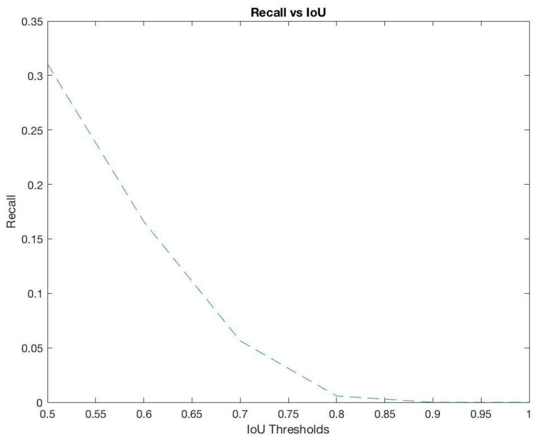


Figure 28. Recall vs IoU curve.

V. Conclusion

Consequently, we implemented the algorithm as described in the given paper [1]. Experimented with and without complexity adaptivity feature, i.e only with the low complexity and both high and low complexity distances. The results were similar what discussed in the research. Low complexity distance feature was unable to detect correct similarities after approximately 4 superpixels merged. Which was stated to be 6 in the paper. However, in the paper they used between 300-500 superpixels but in our segmentation we used 150-200 superpixels because of the running time issues. Thus it makes sense to become complicated after 4 superpixels instead of 6. We set the boundary parameter to 4 because of that reason.

Moreover, we experimented with different parameters to tune our approach but the proposed parameters were performed well and we decided to use them as it is. Furthermore, we experimented with the images only using specific functions to see how they result by their own, such as using only edge cost or color-texture distance. Color-texture distance was the only significant one that resulted in somewhat meaningful way. Graph distance and edge cost were only useful when combined with the color-texture feature.

We tried to use the algorithm without the D_s to see to what extent it was necessary, and the algorithm was tend to merge into one big group rather than merging small groups first.

In proposals ranking we set the parameter κ to 0.8 after we tried 0.5, 0.8 and 1. The parameter was affecting our proposals more than we expected so we tried to pick a number that favors the bigger windows slightly.

After the tuning of the parameters we experimented with 100 images to get the precision and recalls for different IoU thresholds. We used 0.5, 0.6 , 0.7 ,0.8 and 0.9 for IoU thresholds and gather the results. Moreover, we picked 2 of the images from each class randomly to visualize the bounding boxes to show in the report.

We presented the results of the recall vs IoU threshold without class seperation (Figure 28) and the precisions for each class with IoU threshold 0.5 and 0.6 (Figure 27).

Contibutions

Serhan Gürsoy, setted up the working environment in MATLAB as well as in Github to work more professionally and with more solid results. Codes for SLIC and SLICO were not supported by Macintosh operating system and it was problematic for first run without understanding the parameter setup for anyone regardless to their operating system. Hence, Serhan created all boot-files and provided *.mat* files of superpixels which is accessible for anyone who own a legitimate copy of MATLAB software. He implemented the Graph Distance and Edge Cost functions with different variations for different purposes in the research. He also implemented the evaluation functions. Formed the report's final state after evaluating it by considering IEEE standards.

Ege Yosunkaya, setted up the template of research paper by following the IEEE standards. He implemented the Color Distance and Texture Distance functions. He did several optimizations throughout the research in order to make it faster in run-time which was extremely effective for evaluation. He implemented the Low and High complexity calculation functions and Merge functions properly as it proposed in [1]. He also implemented Score function with using Edge Response function for scoring the merge operations. He contributed to Results section of this paper with his results which composed in a visual way.

References

[1] Y. Xiao, C. Lu, E. Tsougenis, Y. Lu, and C.-K. Tang. Complexity-adaptive distance metric for object proposals generation. In CVPR, 2015.

[2] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graphbased image segmentation. International Journal of Computer Vision, 59(2):167–181, 2004

[3] L. Yang, X. Wu, D. Zhao, H. Li and J. Zhai, "An improved Prewitt algorithm for edge detection based on noised image," *2011 4th International Congress on Image and Signal Processing*, Shanghai, 2011, pp. 1197-1200. doi: 10.1109/CISP.2011.6100