Multilevel Object Closure Detection by Superpixel Grouping

CSC2523 Research Project, Winter 2018

1 Abstract

The closure of any object in an image sets the scope for the pixels for the object, and those pixels within the closure can be used for further application, for example recognition tasks by neural network models. Finding a proper closure for an object thus can be thought as a preprocessing to focus the recognition on a specific object instead of finding a main object among a few possible objects. The definition of objects can be vague and abstract, but if we treat objects in different level, we will be able to have the priority in detecting objects by their closures in an image. The definition of level can be defined as that if an object's closure contains a few other objects' closures, that object will be in the parent level for the other objects within the parent object. Our project was based on the work of Levinshtein et. al. [1], and our goal was to develop a mechanism to recursively detect the closures of objects at different levels. This could be a very challenging and complicated task, and there are a few critical concepts involved in our project. For example, the Pb edge detector, superpixels, superpixel grouping, and the recursive mechanism. Therefore, these concepts will also be discussed in this report. Though we have not yet achieved ideal results, the progress in this project showed that our method is feasible to be applied for detecting multiple objects by their closures to an image.

2 Introduction

2.1 Background and Literature Review

Image segmentation is the process of separating an image into multiple parts of interest. This process can also be regarded as grouping adjacent pixels based on their similarity, the edges in the image, and some other prior information. Properly segmenting an image is essential for many computer vision tasks, including object recognition, detection, and tracking. Performing an appropriate segmentation on images has been an active area of research for the last few decades. The ongoing goal is to get segments of an image that are as close as possible to how human beings perceive and understand. Obtaining the contour closure provides a reasonable method towards achieving this goal. This procedure runs by first getting the contours and then linking some contours together to get the closure, which at the same time, segments the area of interest [1], [2]. However, correctly linking contours together to form a closure still remains an open question.

According to [1] and [3], early work attempted to incorporate the prior information, like shape features and geometric descriptions, to constrain the contour grouping process, but prior information is difficult to encode quantitatively. As Shi and Malik [4] pointed out, the prior information can also be subjective for different human users. Therefore, there is no standard about how to use the prior information for the contour closures. Moreover, there were a few researchers, who tried to use shape features to group contours, utilizing geometrical features like parallelism and symmetry to accomplish this [1]. However, geometrical features also include too many complexities. For example, there are other features like collinearity, orthogonal, and other shape change patterns. Thus, the difficulty embedded in this kind of method is not a feasible option for the problem. Later, other research work attempted to simplify the shape features into weak shape priors, such as continuity and proximity; and they proposed a concept named boundary gap, which is a measurement of missing edges along a possible closed contour [1]. Taking advantage of probability theory, Elder and Zucker [5] tried to compute the probability of a link between any two adjacent contour fragments and found contour cycles by using a shortest path algorithm. Wang et al. [6] applied the ratio cut method to optimize a measure of average gap. However, all the above methods were too complicated to cope with all possible challenges in real applications of detecting contour closure. But, Levinshtein et al. [1], [3] took advantage of superpixels and made collections of them to overcome these computational complexity limitations. Thus, Levinshtein's work became a starting point for our project.

We also want to briefly introduce superpixels here. A superpixel can be thought as a group of adjacent pixels, and it can serve as the higher-level abstraction of pixels. The researchers in the superpixel field [1], [2], [3], [4] have showed that superpixels are convenient for applications such as depth estimation, image segmentation, skeletonization, body model estimation and object location. There are many methods to generate superpixels for an image, and they generally use different algorithms and have various efficiencies. According to Achanta et. al. [2], there are two categories of superpixels: graph-based and gradient-ascent-based. In our project, we chose one graph-based method (Ncuts) and two gradient-ascent-based methods (SLIC and TurboPixels). Their details will be introduced in section 3. After getting the superpixels for an image, Levinshtein et al. [1] reformulated the grouping of superpixels as a mathematical optimization problem, and the outer boundary of the selected superpixels will be the contour closure for an object. Based on this result, we further the research by implementing it recursively for all possible objects' closures in multiple levels. We also intended to build a tree structure to show the relationships among the objects in an image, and we found that this idea was quite similar to some motivations of building a hierarchical structure for objects in an image once proposed by Arbelaez et. al. [7] and Shi and Malic [4]. In the following section, we will introduce the basic framework of our project.

2.2 Project Overview and Structure

This research project was done based on the work and code from [1]. There were a few critical steps involved, applying Pb edge detection, superpixels, superpixel grouping, and recursive object-closure detection. Following are brief descriptions, and the details are included in susequent sections.

The first step was to compute the Pb edges for the image. The Pb edge detector was first proposed by Martin et. al. [8] to combine brightness, color, and texture as local features to detect edges for natural images. The result of the Pb edge detector was used to help use determine if a contour has enough edge support. The more edge support we have, the more confident we have for the selected contour closure.

The second step was to compute superpixels, which are a higher level abstraction than the raw pixels in an image. Since we intended to use a group of adjacent superpixels to represent an object and use their outer boundary to approximate the closure, generating superpixels and labelling them for an image were necessary. There are a few different algorithms to compute superpixels, and we chose SLIC, NCuts, and TurboPixels as the main candidates. We also intended to compare the results by using these three different superpixel algorithms.
The third step was to formulate the problem and group multiple adjacent superpixels. According to [1], the grouping of multiple superpixels to get their outer boundary as an approximation of the closure of an

object and can be formulated as an optimization problem. This problem can then be reformulated as an energy flow problem and be solved by parametric maxflow.

The last step is to wrap the above three steps as a function and use it to detect the objects' closures on multiple levels. For the same level object's closure detection, we intend to search all valid objects's closures with the criteria of reaching a certain size. For the next level objects of a parent object, we intend to search all their closures within the parent object's closure on the parent level. By recursively repeating these two steps, we intend to find all objects' closures and construct a tree-structure, which represents the relationships among all objects in an image. We refer to this as the recursive object-closure detection. Further details will be discussed in the next section.

3 Methodology

In this section, we will introduce the major techniques involved in this research project.

3.1 Pb Edge Detector

In our project, the Pb edge detector was used in the mathematical optimization of the superpixel grouping process in the section 3.3, and it was first introduced and implemented by Martin et. al. [8]. It was utilized by them to solve the edge detection difficulties, which could not be solved by more traditional algorithms like the Canny edge detector [9]. Martin et. al. [8] found that the Canny edge detector did not perform well, when there were high contrast edges inside a texture area or boundaries among different objects with subtle difference in brightness. Therefore, they [8] proposed the idea to combine brightness, color, and texture to calculate a posterior probability of whether or not each pixel is on a boundary. For calculating features, they used local discontinuities for each feature, over multiple ranges and scales. To implement this, they treated brightness using the oriented energy (OE) approach [10]. According to [8], the OE is reproduced as the following:

$$\mathrm{O}E_{\theta,\sigma} = (I * f^{e}_{\theta,\sigma})^2 + (I * f^{o}_{\theta,\sigma})^2$$

Here, I is the intensity at the pixel; $f^{e}_{\theta,\sigma}$ and $f^{o}_{\theta,\sigma}$ are functions of even and odd symmetric filters at orientation θ and scale σ. According to Martin et. al. [8], the even-symmetric filter is a Gaussian second-derivative, while the corresponding odd one is its Hilbert transform. For more details, please check [8] and [10]. In short, this equation is used to detect and localize composite edges [11].

The Pb edge detector also incorporates gradient-based features, which are used to treat color, texture, and also brightness. Martin et. al. [8] used the following formula to calculate the local gradients for each of these features:

$$\chi^2(g,h) = \frac{1}{2}\sum_{i} \frac{(g_i - h_i)^2}{g_i + h_i}$$

The motivation is to use a circle with two half equal-sized discs and center the circle at each pixel. Then, a diameter at a certain orientation will be the edge between the two discs. The user can then set the number of bins in each disc and calculate the values for each pair of bins from the two discs. A bin can be thought as a vertical bar in a half-circle disc, and it is used to calculate the feature of the bin by considering all pixels within it. The g_i and h_i are the values of the ith bin from the two discs. The resulting value will summarize the local feature for that pixel. For each pixel, the above equation will then be used to individually calculate color gradient (CG), texture gradient (TG), and brightness gradient (BG) based on the color, texture, and brightness features, respectively.

To combine all the above equations (OE, CG, TG, and BG), Martin el. al. [8] considered that using linear weights could be enough, and they proposed to use supervised learning approaches to learn the weights. They tested multiple models, including density estimation, classification trees, logistic regression, mixture of experts, and SVM. As a result, they found the weights produced by even simple classifiers could perform well. The benefits of using the Pb edge detector for detecting edges are to avoid relying too much on the brightness feature by traditional algorithms. Later, Arbelaez et. al. [7], based on the work of Martin et. al. [8], proposed a more advanced Pb edge detector named Global Pb edge detector, which also take into consideration the global constrains in the edge computation. Therefore, incorporating the Pb edge detector instead of a traditional edge detector in our project helped prevent treating an edge inside an object as a a part of its closure in the later steps.

3.2 Superpixels

We chose three superpixel algorithms as candidates to compute superpixels. They are Simple Linear Iterative Clustering (SLIC), Normalized Cuts (Ncuts), and TurboPixels. All three have open source packages available for use in MATLAB.

3.2.1 SLIC

SLIC was first proposed by Achanta et. al. [2], and their motivation was to find an algorithm to compute nearly equal-sized superpixels efficiently for an image. Intuitively, for all pixels, the algorithm performs local clustering of a 5-D space data defined by L, a, b, values of the CIEAB color standard and the x, y coordinates. Suppose an image has N total pixels, and a user sets K as the number of total superpixels to have in the image. Then, there will be around N/K pixels for each superpixel. If our goal is to make each superpixel's shape similar to a perfect square, then the length of its side will be $\sqrt{N/K}$. Thus, this value will be a good estimate for the interval distance between each pair of initial superpixel centers on the image. After that, we can use $C_k = [l_k, a_k, b_k, x_k, y_k]^T$ to denote each superpixel center, with k as the index from 1 to K. Here, Achanta et. al. [2] made an assumption that the search area for the pixels associated with a superpixel center will be within a $2S \times 2S$ area around the center, since the area of each superpixel is about S^2.

Then according to [2], the L, a, b values are used to calculate the color distances from neighboring superpixel centers, while the x, y coordinates are used to estimate the Euclidean distances from the center. The following formulas from [2] were used to combine both color and geometric distances for each pixel from a nearby superpixel center:

$$d_{lab} = \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2}$$
$$d_{xy} = \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2}$$
$$D_s = d_{lab} + \frac{m}{S} d_{xy}$$

Here, d_{lab} and d_{xy} are the Euclidean distances for L, a, b values and x, y coordinates, respectively. D_s is a linear combination of d_{lab} and d_{xy}. Specifically, $S = \sqrt{N/K}$, and the m value is chosen by the user.

Since S will be fixed from the fixed N and K values, the m value will determine how much weight we put on the distance for x, y coordinates.

Then, a minor adjustment is normally required for the locations of initial superpixel centers to prevent them from sitting on any edges [2]. We chose to ignore this minor detail in this work because it did not have a significant impact on our results. The interested reader may refer to the original paper. Finally, the formal algorithm proposed by [2] will be to first initialize the superpixel centers on the grid of an image by the interval distance of $\sqrt{N/K}$. Then, each pixel in the image will be assigned to its nearest superpixel center. After this step, a new superpixel center will be calculated by averaging all the $C_i = [l_i, a_i, b_i, x_i, y_i]^T$ of each pixel belonging to that center. Then, it will repeat the above two steps, until the results of all suerpixel centers become stable.

The advantage of using SLIC to compute superpixels is that its efficiency is outstanding. According to [2], its efficiency is $O(N)$, where N is the total number of pixels in an image. However, since the SLIC algorithm is a special case of the K-means algorithm in the unsupervised learning field, it may inherit some of the disadvantages from the K-means algorithm family.

3.2.2 Ncuts Algorithm

Ncuts was first introduced by Shi and Malik [4], whose intention was to use a graph partition method to consider global features in image segmentation. Instead of focusing on the local features like the SLIC superpixels, Ncuts attempts to get the global sense of an image as how a human would perceive an image. In this algorithm, they [4] modeled an image as a weighted undirected graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$. Each pixel in the image corresponds to a node in $\boldsymbol{V}$ in a graph, and the edges $\boldsymbol{E}$ have the weights that are the similarities between each pixel and its neighboring pixels. Then, segmenting the image becomes a similar exercise to producing a graph cut. Suppose we want to cut the whole graph into two sub-graphs A and B, and then let $w(u, v)$ denote the edge weight between the node u and v in the graph. Now, the graph cut cost can be defined as the following [4]:

$$cut(A, B) = \sum_{u \in A, v \in B} w(u, v)$$

According to Shi and Malik [4], by considering the dissimilarities among different groups and the similarities within each group in the model, Ncuts avoids the tendency to separate an individual pixel as a group out from all other pixels. The disassociation measurement between different sub-graphs can then be written as the following, where we use two different sub-graphs as an example [4]:

$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)}$$

Here, $assoc(A, V)$ is defined as the total edge weights from the nodes of group A to all other nodes, and a similar definition applies to $assoc(B, V)$. As we discussed above, instead of simply using $cut(A, B)$ as a measurement, by putting the $assoc(A, V)$ and $assoc(B, V)$ in the denominators, the cost is normalized. This normalization is what gives the algorithm its name. Shi and Malik [4] proved that minimization of the disassociation among different sub-graphs is equivalent to maximization of the association within each sub-graph. Therefore, we just need to use the above $Ncut(A, B)$ formula for the next step.

In this paper, we only discuss the core mathematical concepts, and leave the discussion of the optimizations to the original paper. Let $\boldsymbol{d}(i) = \sum_j w(i, j)$ be the weights of all edges incidental to the node

i; let $\boldsymbol{D}$ be an $N \times N$ diagonal matrix, where N is the total number of nodes, and $\boldsymbol{d}(i)$ of each node is on the diagonal of the matrix. Moreover, let $\boldsymbol{W}$ be an edge weight matrix with the size of $N \times N$, and each element $W_{i,j} = w(i,j)$ is the edge weight between node i and node j. We also need to define an N dimensional vector $\boldsymbol{x}$ to indicate which nodes are in the segmented-out sub-graph and which nodes are among the other sub-graph. After defining these variables, the problem is reduced to the following eigenvector calculation [11]:

$$(\boldsymbol{D} - \boldsymbol{W})\boldsymbol{x} = \lambda \boldsymbol{D}\boldsymbol{x}$$

Here, $\boldsymbol{x}$ is the eigenvector for the graph cut, and λ is its corresponding eigenvalue.

The Ncuts algorithm sets up the graph model $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ for an image and calculates the weight for each edge for each pair of neighboring pixel nodes. Then, solve the above eigenvector calculation and use the corresponding eigenvector as the solution for the sub-graph to be segmented out from the original graph. The above steps are repeated recursively until no more necessary sub-graphs are left from the original whole graph. According to both [2] and [4], the efficiency of the Ncuts algorithm is $O(N^{\frac{3}{2}})$, which is generally slower than the SLIC superpixels. But it considers global features in the calculation and uses normalizations to prevent extreme cases.

3.2.3 TurboPixels

TurboPixels was first introduced by Levinstein et. al. [12] to compute compact superpixels with high efficiency by using geometric flows. They found that oversegmentation will be better than undersegmentation, since merging superpixels is easier than splitting them. This becomes one of the motivations for TurboPixels. Formally speaking, there are five principles behind TurboPixels; they are uniform size and coverage, connectivity, compactness, smooth and edge-preserving flow, and no superpixel overlap [12]. Since this superpixel algorithm is very mathematical heavy, we will not list all mathematical details here but instead show the intuitions of the design. If you want to check the mathematical details involved, please check the original paper. According to [12], the basic idea behind the design is to first have a user place a number of small circle “seeds” on the original image and distribute them on the grid of the image with roughly the same distances among the neighboring seeds. Then, the algorithm will try to expand the boundaries of all seeds repeatedly until they cannot evolve further. With the final seeds’ circles, the algorithm can infer the corresponding lattices that contain them, respectively. As a result, these lattices are the desired superpixels. Compared with Ncuts, the efficiency of TurboPixels is preferable, and it is roughly $O(N)$, where N is the total number of pixels in the original image [12]. Different from Ncuts, which uses graph-cut technique, TurboPixels utilizes the geometric flows to “grow superpixels from the circle seeds”. However, TurboPixels does not consider global features as Ncuts does.

3.3 Superpixel Grouping

After getting superpixels, the next goal will be grouping them together and using their outer boundary as the closure for an object. Based on the method proposed by Levinshtein et. al. [1], finding the optimal grouping of some adjacent superpixels can be reduced to a mathematical optimization problem. Intuitively, we try to pick out several superpixels, whose boundaries have strong edge support. Because an object of interest is usually composed of a bunch of adjacent superpixels, we place some emphasis on their spatial coherence. Levinshtein et. al. proposed a cost function, which is a ratio of a novel learned boundary gap measure to the area of the selected superpixels to solve this problem. Additionally, using parametric maxflow, we can get a few possible candidates for optimal groupings of superpixels. Following is a more detailed discussion of the algorithm.

First, let X_i be a binary variable for the *ith* superpixel, with the value 1 indicating that it belongs to the grouping of selected superpixels and 0 indicating that it does not. The superpixels of the whole image are represented by vector $\boldsymbol{X}$. Based on Stahl and Wang [13], Levinshtein et. al. [1] defined the cost function to be $C(\boldsymbol{X}) = \frac{G(\boldsymbol{X})}{A(\boldsymbol{X})}$. The numerator $G(\boldsymbol{X})$ measures the cost of the gap measurement around the outer boundary of the group of selected superpixels, while the denominator is the area of all the selected superpixels. More specifically, $G(\boldsymbol{X}) = P(\boldsymbol{X}) - E(\boldsymbol{X})$, where $P(\boldsymbol{X})$ measures the total number of pixels on the outer boundary of selected superpixels, and $E(\boldsymbol{X})$ measures the total number of pixels on both the outer boundary and an edge detected by previous steps. The more boundary pixels on an edge, the smaller the value of $G(\boldsymbol{X})$. This makes sense, because the grouping of the selected superpixels is a better estimation of a contour closure for an object, if the boundary of them are supported by the edges detected by the Pb edge detector. Levinshtein et. al. also showed a few methods to determine if a pixel is considered on an edge. Here we will explain the intuition behind their approach.

The next step is to minimize the above cost function. In fact, for binary variables, we can think of ratio minimization problem as a parametric maxflow problem. Kolmogorov et al. [14] showed that under certain constraints on the ratio $C(\boldsymbol{X})$, the energy $C(\boldsymbol{X}, \lambda)$ can be submodular and can thus be optimized globally in polynomial time by the min-cuts algorithm. The method from [14] can not only optimize the ratio $C(\boldsymbol{X})$, but also find all intervals of λ (and the corresponding $\boldsymbol{X}$). The smallest breakpoint λ_0 will correspond to the optimal ratio $C(\boldsymbol{X_0})$, and consecutively larger breakpoints λ_1, λ_2, . . . will correspond to other ratio optimization for other intervals [14]. Therefore, with the parametric maxflow method applied to optimize the closure cost function, we can obtain a set of optimal closure solutions, and usually the first solution will be the optimal grouping of some superpixels.

3.4 Recursive Object-Closure Detection

The benefit of using closures as a means to detect objects is that it not only gets a closure for an object, but also tells a scope for other objects. Therefore, within a closure of a parent object, we can potentially find the objects' closures in the next level. Outside the closure of the object, we can find other objects at the same level. Therefore, because of these two mechanisms, we can potentially find all possible objects' closures in an image. Furthermore, when the process finishes, a tree-structure can also be built to indicate the relationships among the objects. The following is the pseudo code for our algorithm of searching for all objects' closures in an image. Due to time constraints, a full implementation of the tree structure is not yet complete. However, the following psudo code will show our design intention.

```
=========================================
function extract_objects (image, mask, tree_node):
   if (mask's size is more than a certain threshold size):
         return;
   end;
   masks_holder = find_objects_same_level(image, mask)
   while (masks_holder is not empty)
         its_mask = get_first(masks_holder);
         add its_mask as a child_node of the tree_node;
```

```
        extract_objects(image, its_mask, child_node):
        save its_mask to a file;
    end;
end;

function find_objects_same_level (image, mask):
    initialize m_holder for masks;
    a_mask = find_an_object(image, mask);
    while (a_mask > certain size threshold)
        add a_mask to m_holder;
        mask = mask & not(a_mask);
        a_mask = find_an_object(image, mask);
    end;
    return m_holder;
end;
=========================================
```

Explaining the above algorithm, the main function is the *extract_objects* function, which is to extract all objects by their closures from an image. It contains a recursive call to itself, because our design is that the algorithm will recursively search objects for the next level. All objects will be represented by their masks, which will be saved into a file and the tree structure. The second function, *find_objects_same_level*, is to find all objects at the same level. It calls another function, *find_an_object*, which contains all steps in 3.1, 3.2, and 3.3 to use an optimal superpixel grouping to find a single object's closure in an image within the scope set by a mask. The variable mask is almost everywhere in the pseudocode, because it not only shows an object, but also gives a scope. For example, if we have a mask for a mountain, within the mask's scope, we can find the all objects in the next level for the mountain object. On the other hand, if we negate the mountain's mask, we can exclude the mountain from the search and continue searching for other objects at the same level as the mountain in the image.

We first intended to actually crop an object's closure out from the original image, but this idea was not quite feasible for MATLAB, especially when the object does not have an aligned rectangular shape. Therefore, instead, we used masks to represent objects from the original image. A mask is defined as a 2-dimensional binary array of the same dimension as the original image. Different from the original image, each element in a mask is an indicator to show if this pixel is on (by number 1) or off (by number 0). By element-wise multiplying the original image and the mask, we will be able to get the specific object. Therefore, each object in an image will have its own mask.

For the tree structure generated by the algorithm, it is expected to show the relationships among the objects either at the same level or different levels. For example, consider the following example image.

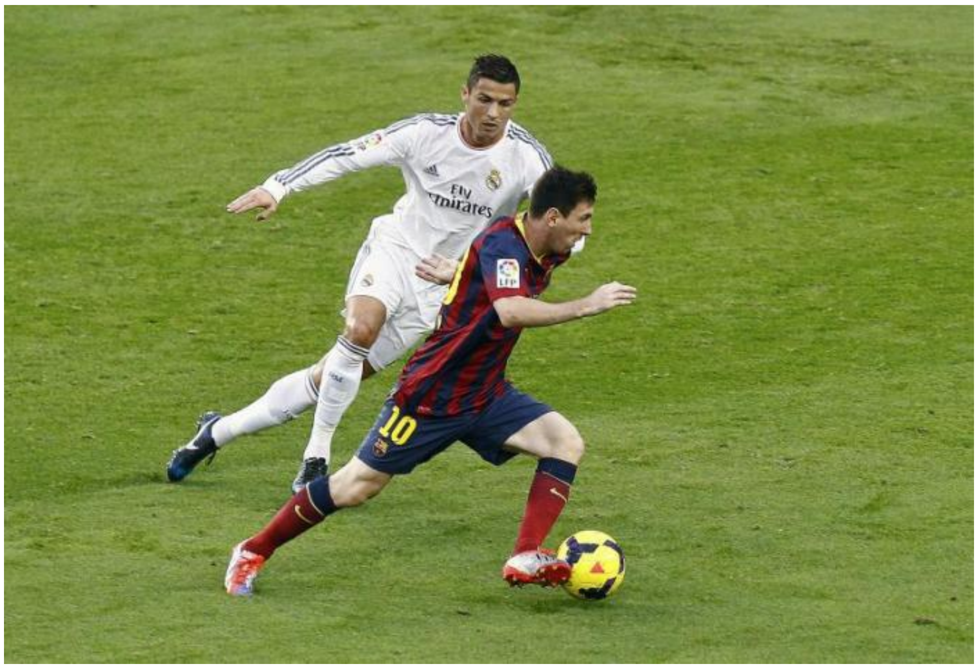

Figure 1. An example image to demonstrate the relationships among objects in a tree structure.

Theoretically, if the algorithm works as expected and detects most major objects from this image, we will have the following tree structure. I put names in the tree nodes to indicate the relationships that will make sense for human beings. This algorithm will not do the recognition part, but it will just generate the following tree structure. Each node contains at least the corresponding mask for the object.

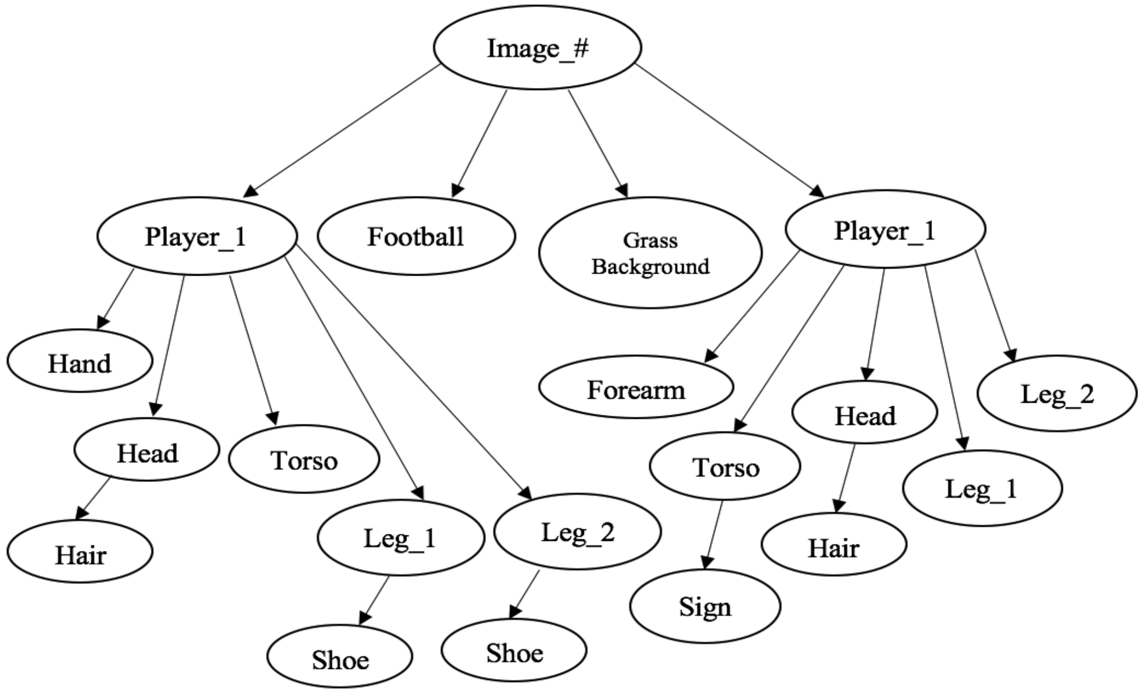

Figure 2. An ideal tree as an example for the relationships among the possible objects in the Figure 1.

In practice, for the sake of exhibition convenience, we label each figure with three numbers to show the tree structure. Since each figure is composed of a batch of son images which are the closure detection results on the father image, we actually need three parameters to locate an image: the level of the image, the batch where the image is in, the rank of the image in its batch. Therefore, we name the figures with key-number in the form “x_y_z”. The number “x” denotes the segmentation level of this image. To show father-and-son relation, the number “y” and “z” cooperate to show the position of this image’s father-image. The number “y” denotes the position of batch where father image belongs to in the higher level. The number “z” denotes the rank of father_image in the batch where it belongs to.

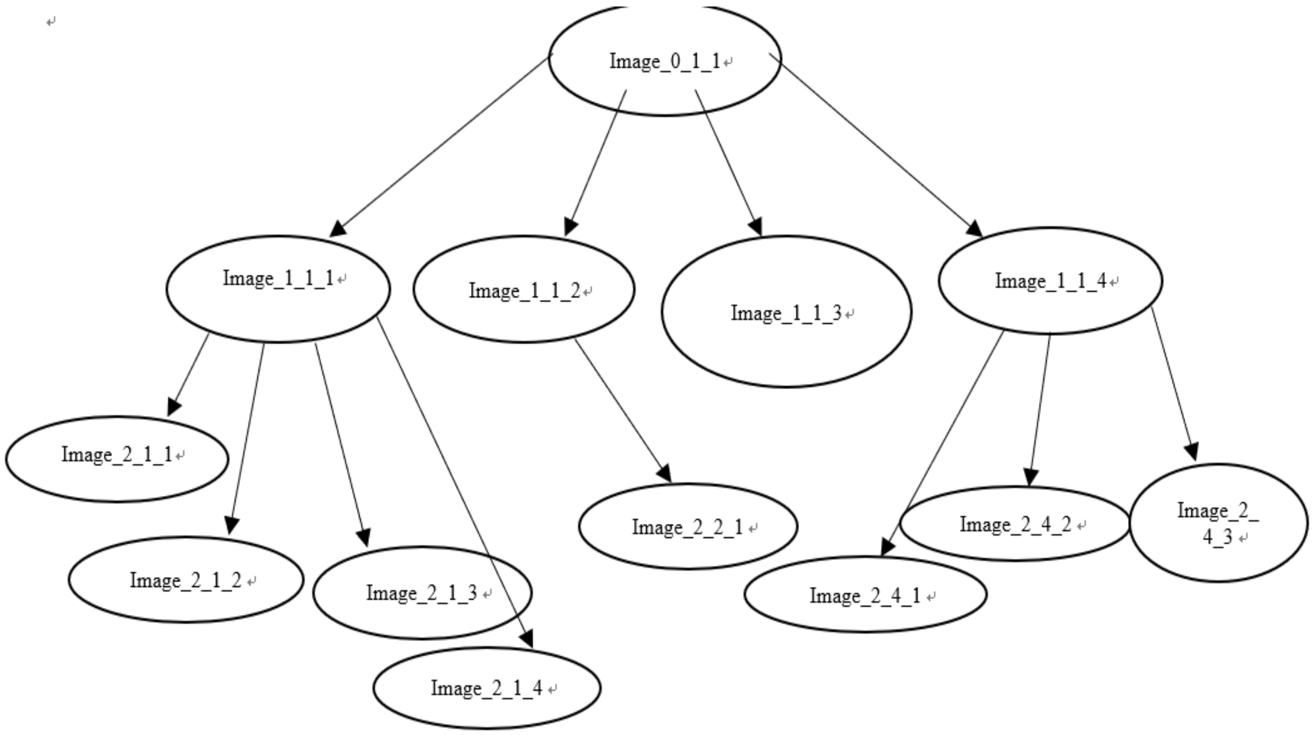

Figure 3. An practical tree as an example for the relationships among the possible objects in the Figure 1

4 Results

4.1 Pictorial results

We implemented our project on MATLAB. It can find all obvious objects in the ground level, second level and third level,etc. Considering the computing cost and semantic information, we generally use 100 to 200 superpixels. At this resolution ratio, our implementation could generally find closures of objects in the ground level, second level and third level. The following are some result examples (the original image followed by detected objects).

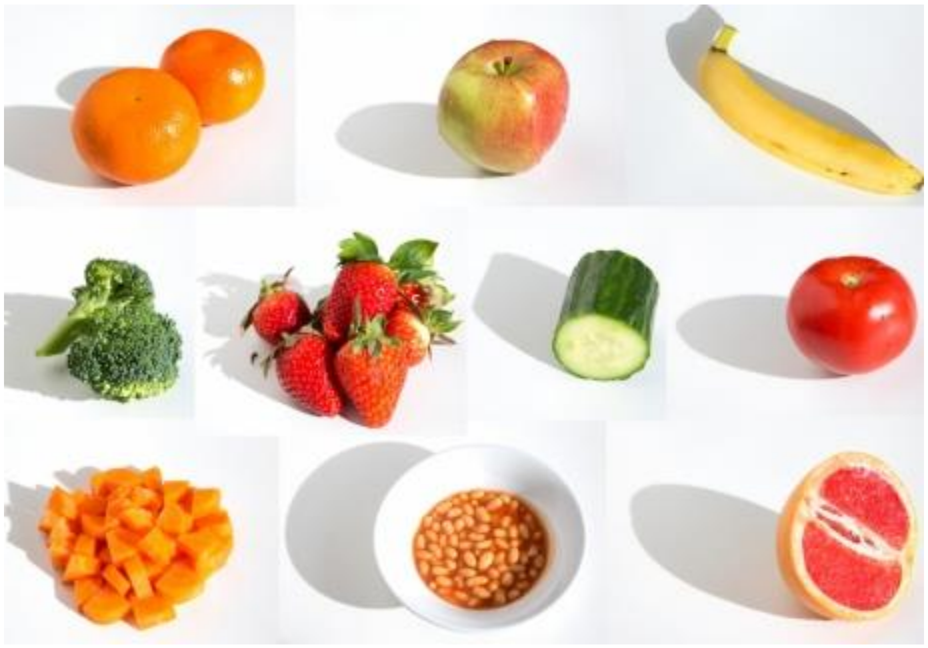
Figure 4. Fruits example image to show the detection of objects’s closures.

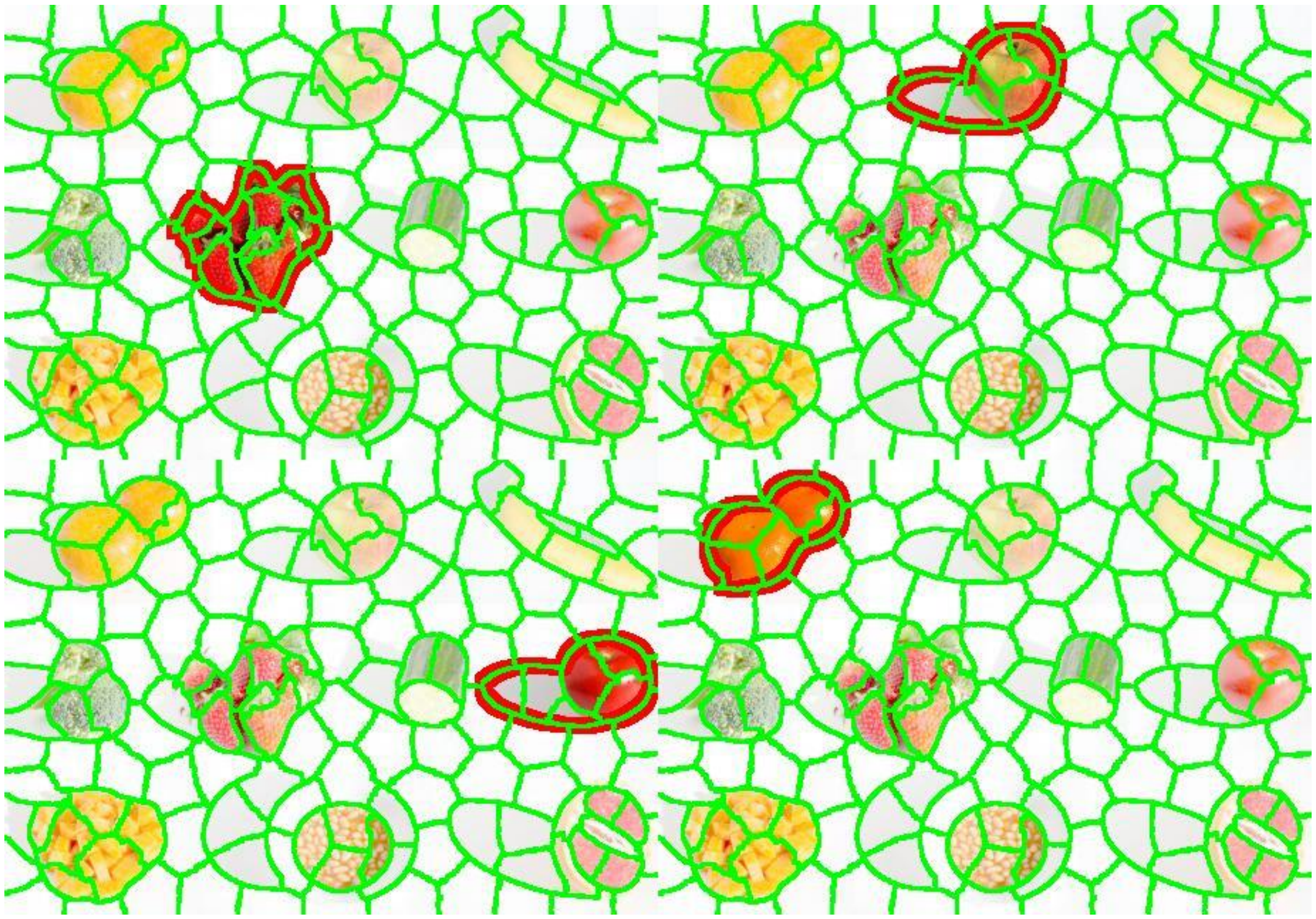
Figure 5. “fruits_1_1_1” image with 150 superpixels at the ground level. There are four pictures which show four closure detection results of original image--Figure 4.

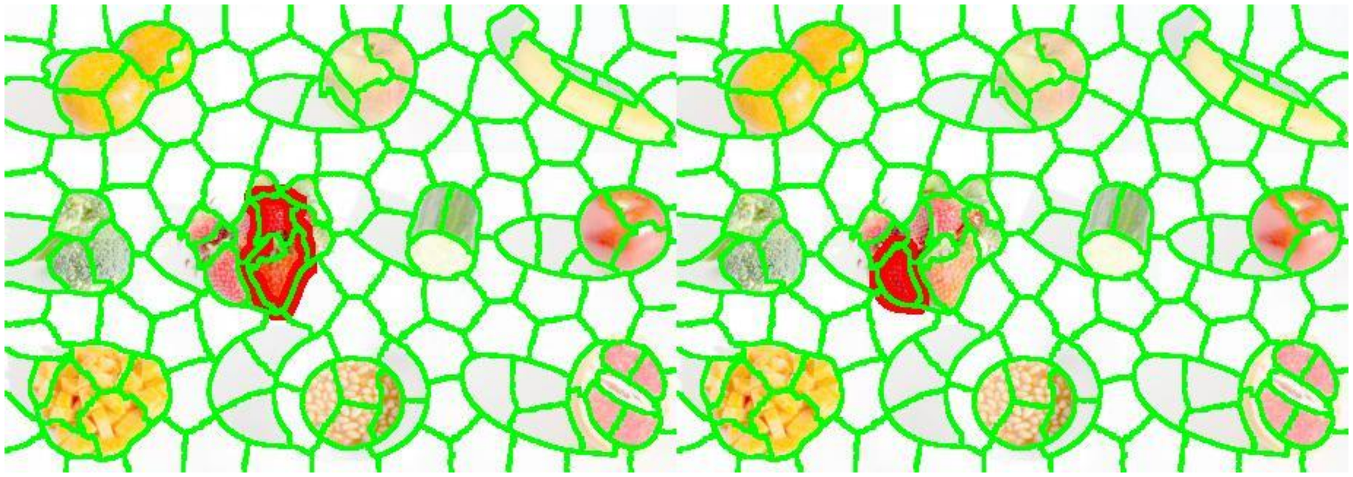
Figure 6. “fruits_2_1_1” image at the second level. There are two pictures in this image which show the closure detection results of objects in the first image in Figure 5.

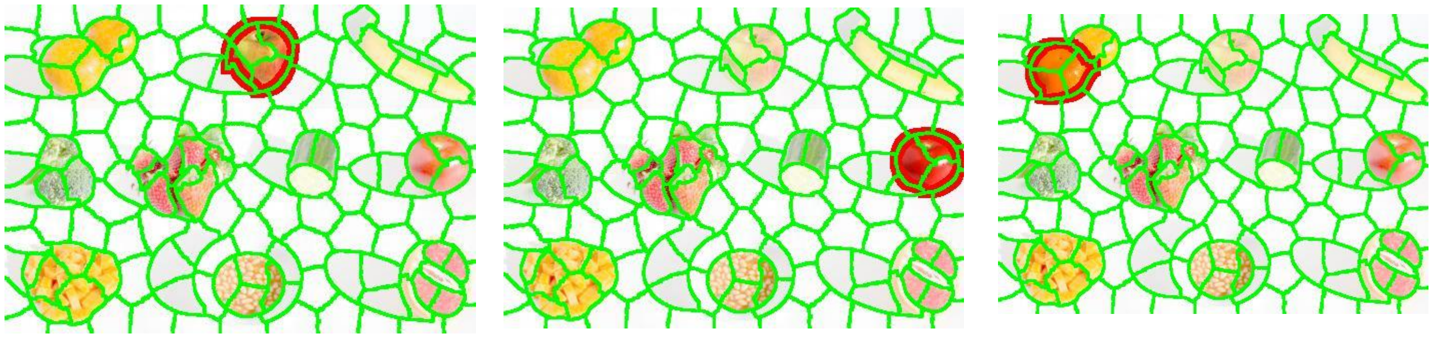
Figure 7. “fruits_2_1_2” image, “fruits_2_1_3” image and “fruits_2_1_3” image at the second level for the the second, third and forth picture in Figure 5.

As comparison, we also used 200 superpixels to detect the object contour of Figure 4. The result is shown below.

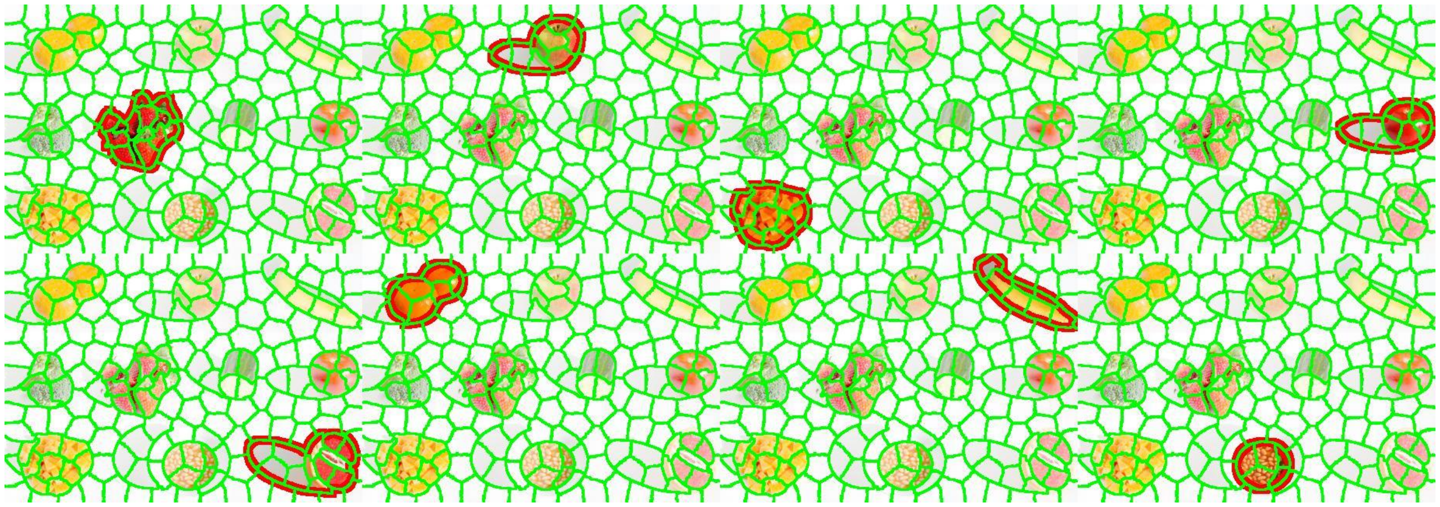
Figure 8. “fruits_1_1_1” image with 200 superpixels at the ground level. There are eight pictures which show eight closure detection results of original image--Figure 4.

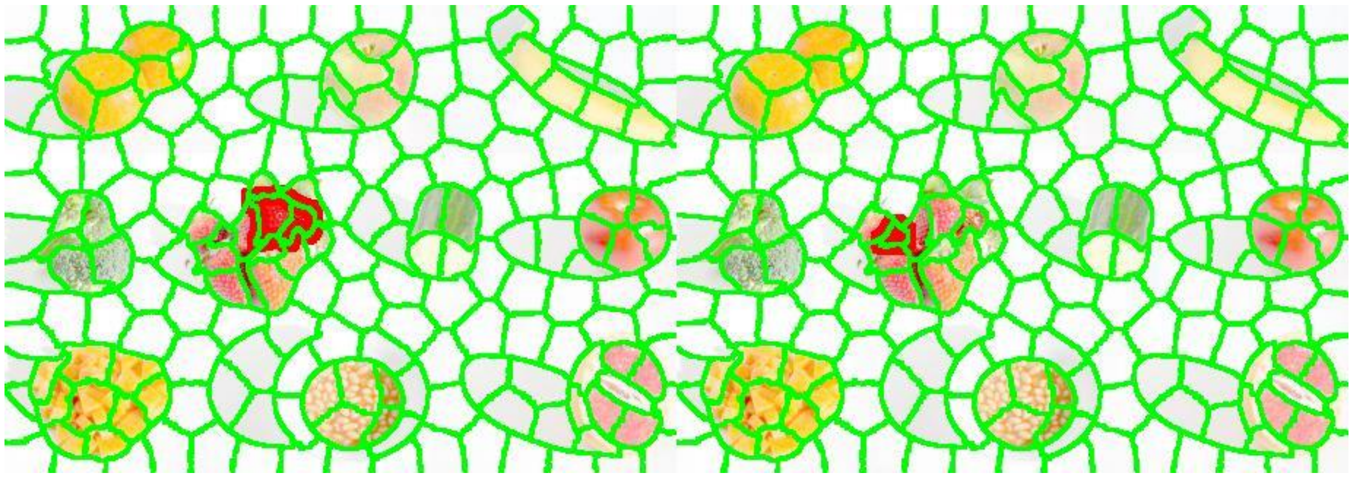
Figure 9. “fruits_2_1_1” image at the second level. There are two pictures in this figure which show the closure detection results of object in the first picture in Figure 8.

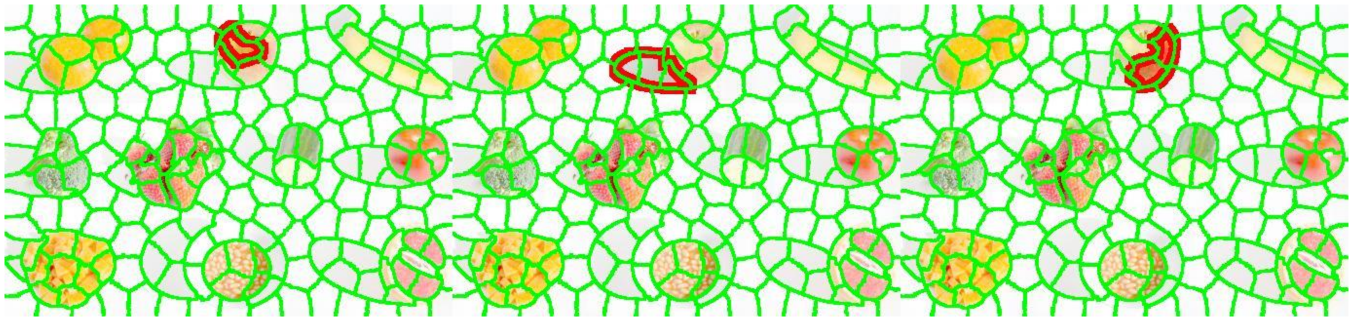
Figure 10. “fruits_2_1_2” image at the second level. There are three pictures in this figure which show the closure detection results of object in the second picture in Figure 8.

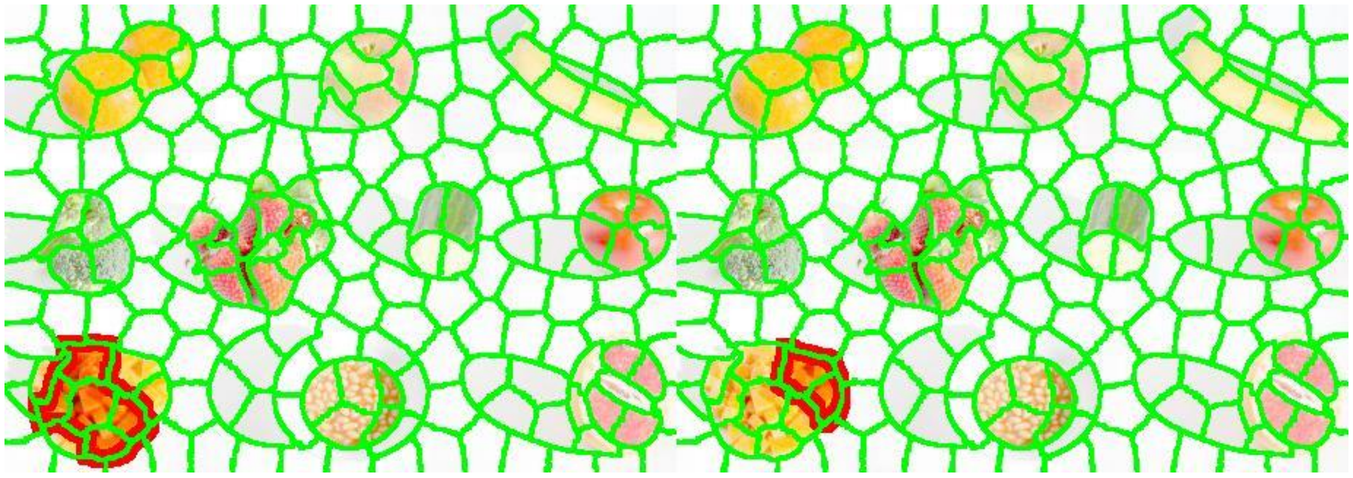
Figure 11. “fruits_2_1_3” image at the second level. There are two pictures in this figure which show two closure detection results of object in the third picture in Figure 8.

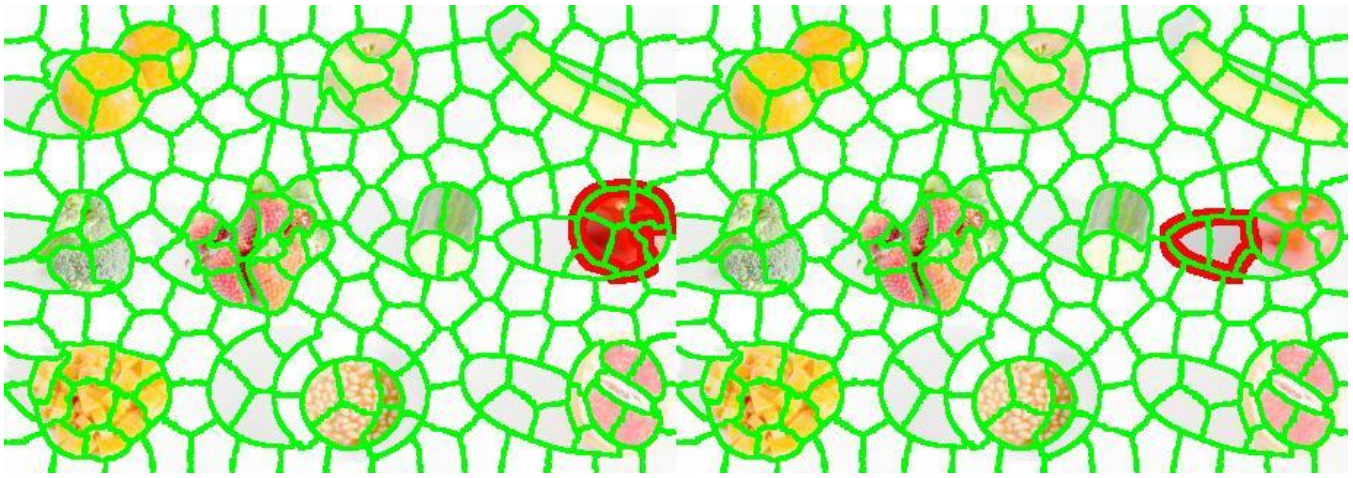
Figure 12. “fruits_2_1_4” image at the second level. There are two pictures in this figure which show two closure detection results of object in the forth picture in Figure 8.

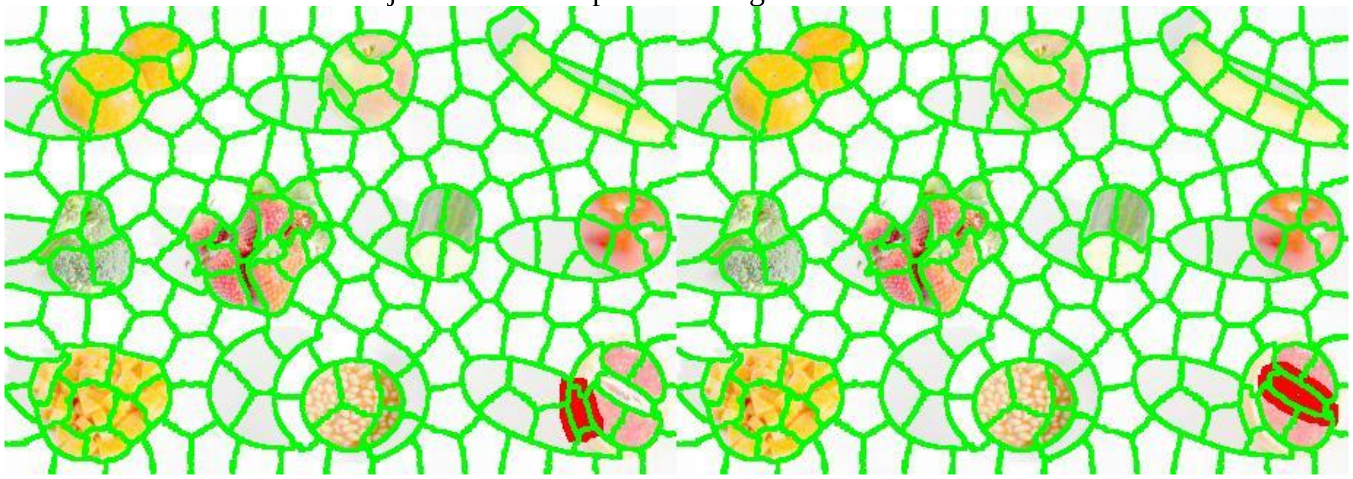
Figure 13. “fruits_2_1_5” image at the second level. There are two pictures in this figure which show the closure detection results of object in the fifth picture in Figure 8.

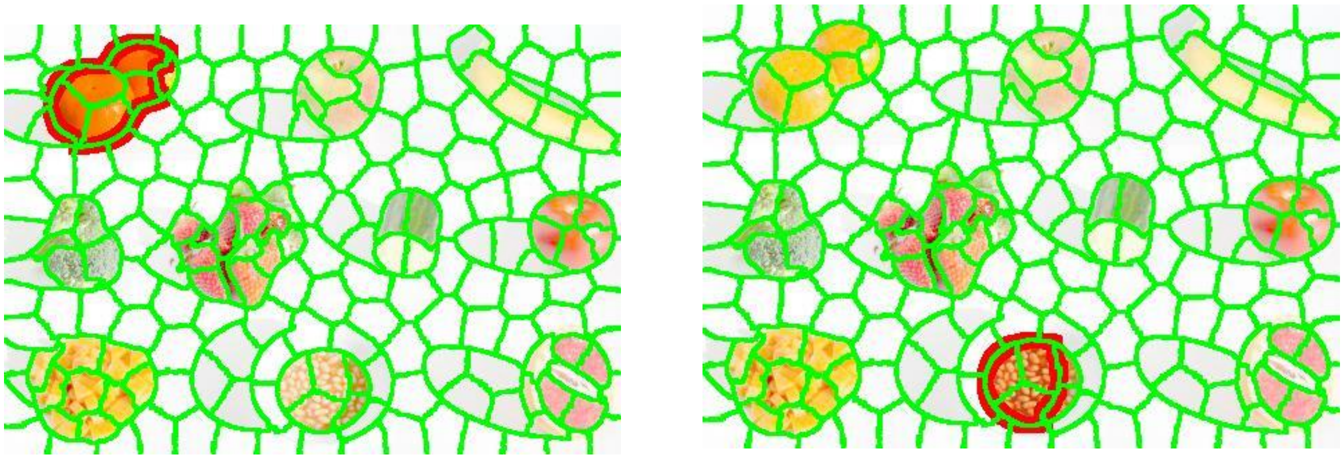
Figure 14. “fruits_2_1_6” image and “fruits_2_1_8” image at the second level. They respectively shows the closure detection results of object in the sixth and eighth picture in Figure 8.

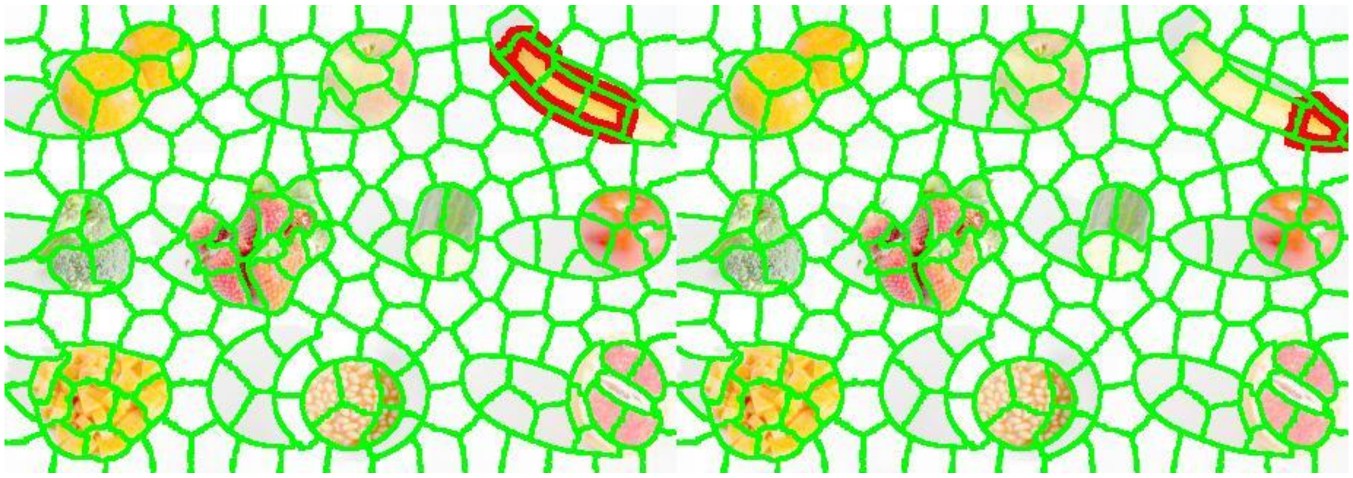
Figure 15. “fruits_2_1_7” image at the second level. There are two pictures in this figure which show two closure detection results of object in the seventh picture in Figure 8.

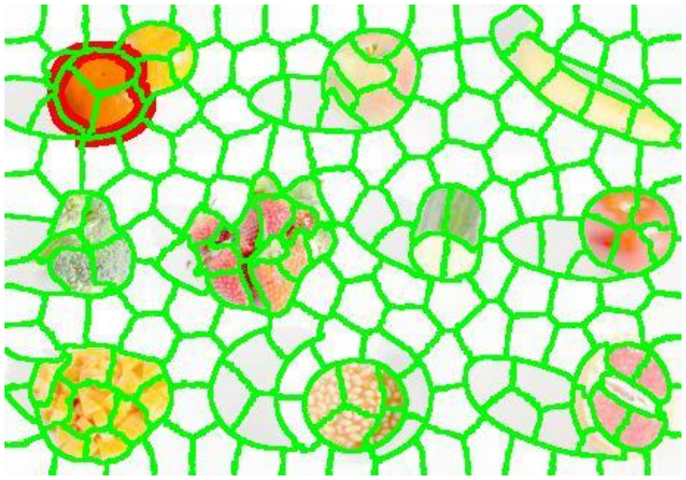
Figure 16. “fruits_3_6_1” image at the third level which shows the closure detection result of object in the sixth image at level two in Figure 14.

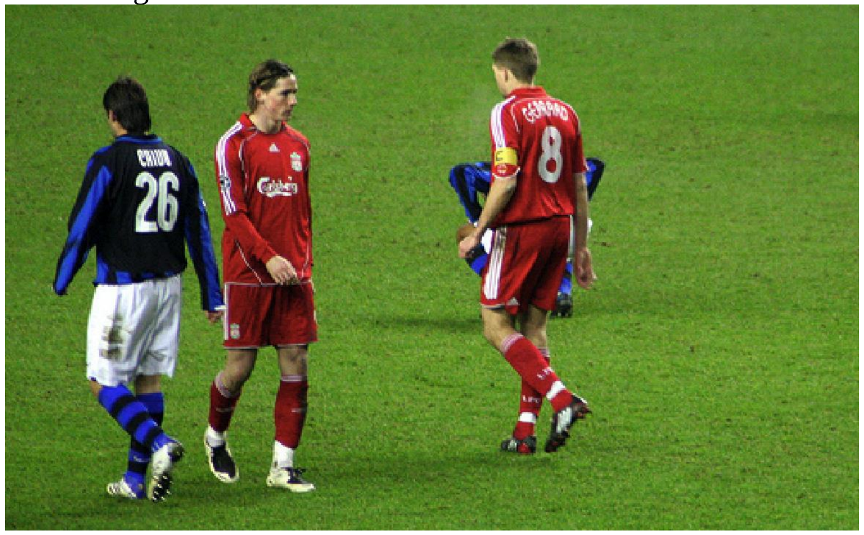

Figure 17. Another image of football players.

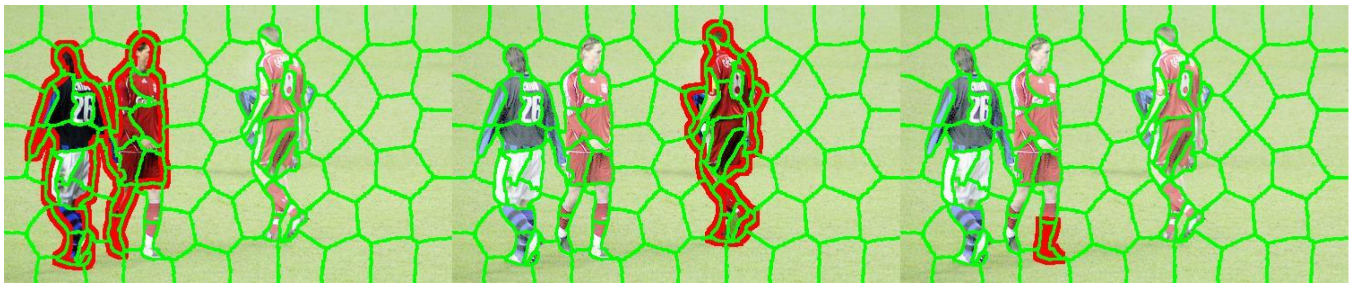
Figure 18. “players_1_1_1” image with 100 superpixels at the ground level. There are three pictures which show eleven closure detection results of original image--Figure 17.

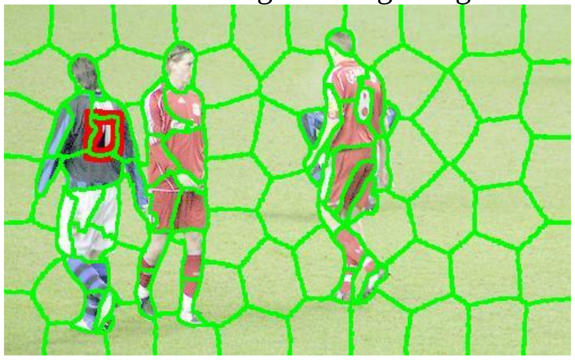
Figure 19. “players_2_1_1” image at the second level which shows one closure detection result of object in the first image at ground level in Figure 18.

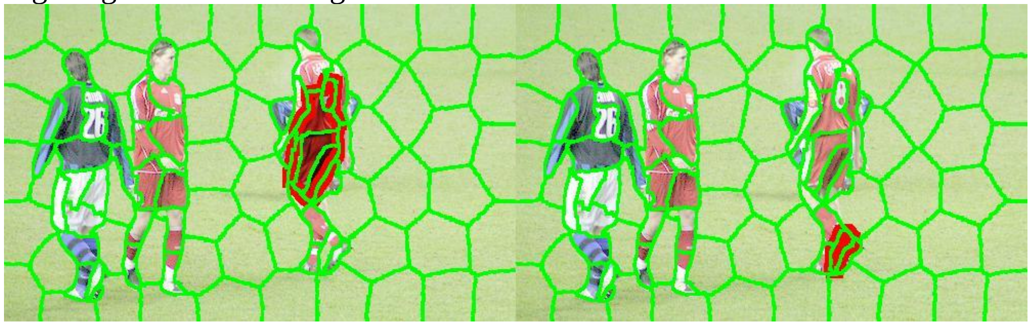
Figure 20. “players_2_1_2” image at the second level which shows two closure detection results of object in the second image at ground level in Figure 18.

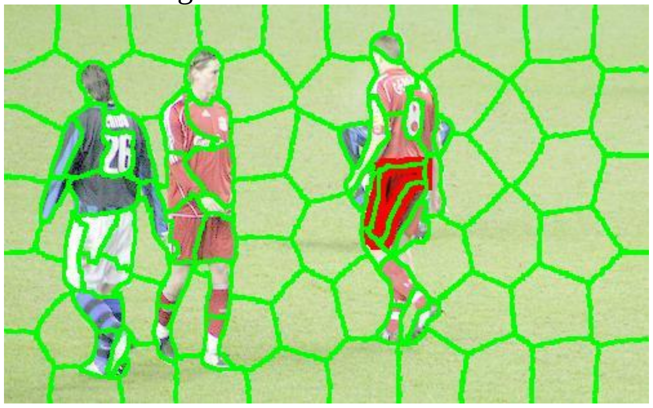
Figure 21. “players_3_2_1” image at the third level which shows one closure detection result of object in the first image in second batch at second level in Figure 20.

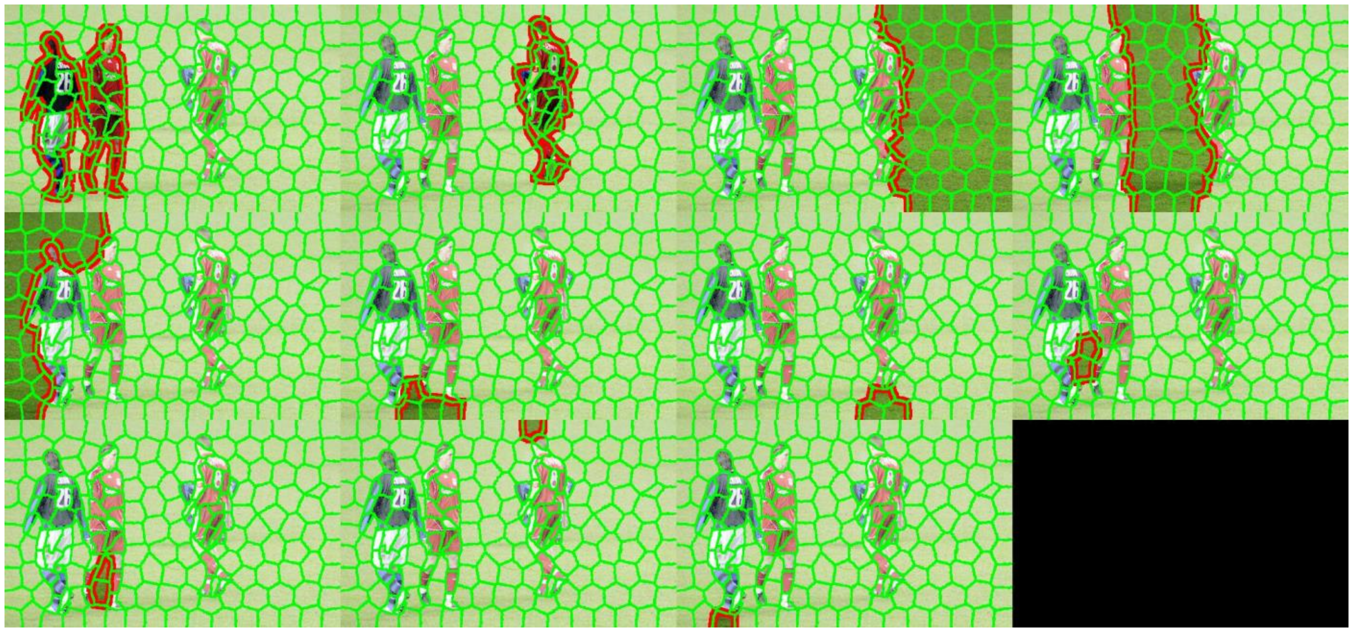
Figure 22. “players_1_1_1” image with 200 superpixels at the ground level. There are eleven pictures which show eleven closure detection results of original image--Figure 17.

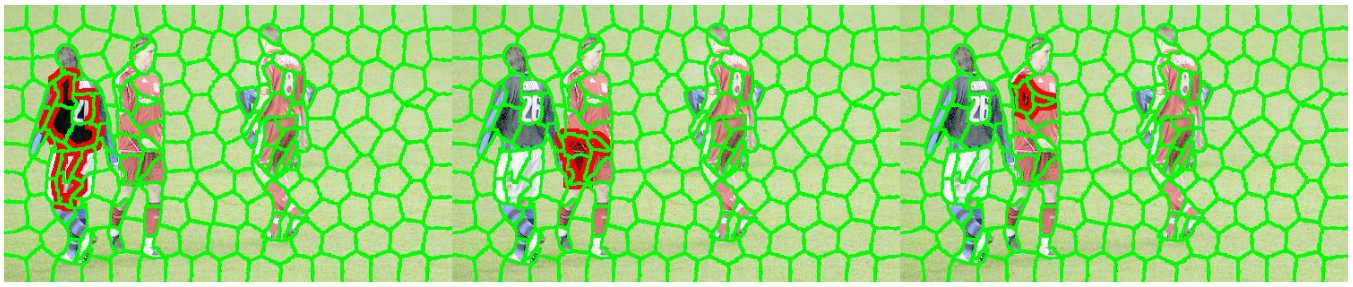
Figure 23. “players_2_1_1” image at the second level which shows three closure detection results of object in the first image at ground level in Figure 22.

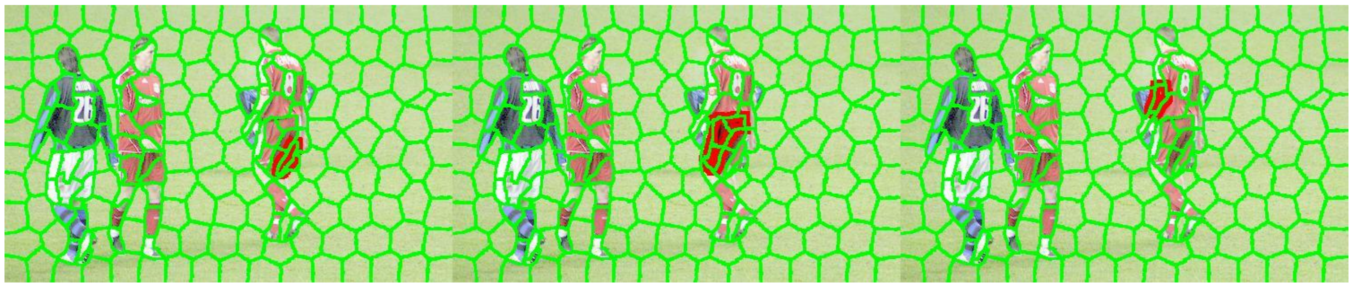
Figure 24. “players_2_1_2” image at the second level which shows three closure detection results of object in the second image at ground level in Figure 22.

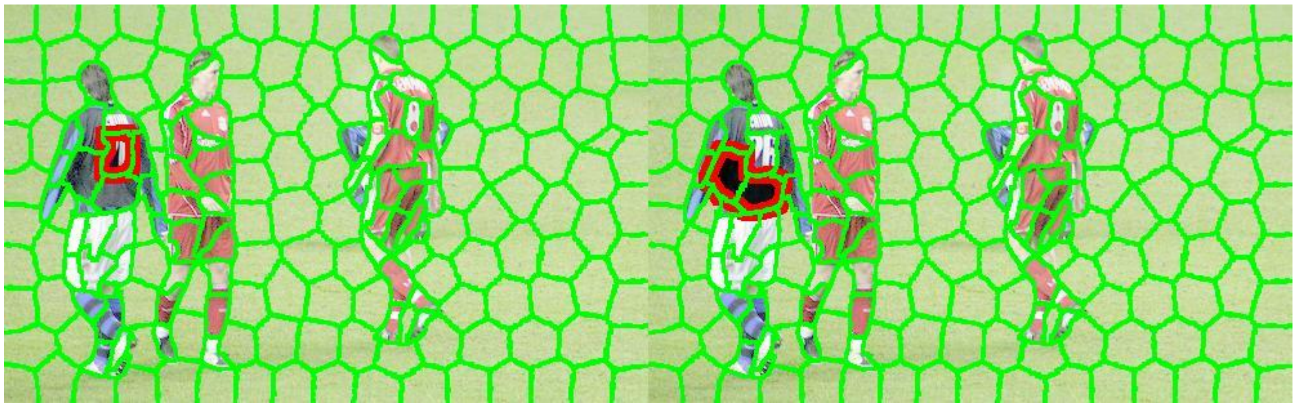
Figure 25. “players_3_1_1” image at the third level which shows two closure detection results of object in the first image at second level in Figure 23.

The implement has met our expectation that it can automatically generate the closures of objects at multiple levels. When the objects’ sizes became smaller and smaller and the number of superpixels in each closure gets fewer and fewer, the program would terminate automatically to avoid over-segmentation.

4.2 Analysis of results

As we can see from the above results, our idea to detect all obvious possible objects’ closures in an single image by using superpixel groupings is feasible in practice. Our implementation on MATLAB could run bug-free and demonstrated a feasible and novel way of detecting objects in the images with considerable color contrasts among objects at different levels.

Totally we show four kinds of implementation—150 superpixels on fruits, 200 superpixels on fruits, 100 superpixels on players and 200 superpixels on players. The above results showed that our work could detect multiple objects well especially when objects can be distinguished by their colors and intensities. For example, Figures 5, 8, 18 and 22, clearly show the most obvious objects’ closures at the first level. At a deeper level, take Figure 25 for instance, it can segment number from the clothes. Take Figure 22 for instance, since the lawn largely has the same color, the 3-11 objects in this figure were not segmented afterwards. Thus, the proposed algorithm can work well with this case.

Meanwhile, at the deeper level, the algorithm’s performance is associated with the balance between resolution ratio and color diversity. In the case of “150 superpixels on fruit”, Figure 6 and 7 show that it can successfully divide strawberries into two parts, separate apples from the background and separate one orange from another. In the case of “200 superpixels on fruit” which has higher resolution ratio, it can detect more objects, including the banana etc. However, as it went to deeper level, it may lead to over-segmentation, like the Figure 15 which shows two parts of a single banana.

However, when different parts of a single object have different colors, it does not handle them well. For instance, in the Figure 18, as the second player has brown leg which distingduish from red shorts, the algorithm sepereate it from the body. For instance, in the Figure 23, the first player have black shirt and white shorts, the result did not segment the first player from the second player successfully. Therefore, heavily relying on the colors of objects might be a weakness in our approach. For the goal of detecting the closures of the objects in the next level, the pictorial examples in second and third levels demonstrate that it is practical. The only concern here is that when the objects' sizes become smaller, they become more difficult to detect.

For the efficiency of our research project, for an image of moderate size (e.g. 500 x 500), it will take about less than a minute to finish detecting the closures for all objects at the first three levels by choosing the SLIC superpixels. It could take longer than a minute by choosing the other two superpixel algorithms. Compared with deep learning approaches, our research project relied on explicit algorithms to detect objects and did not rely on much training before it could work. Therefore, our work could be more efficient. In fact, deep learning approaches can be unstable if the hyperparameters' settings are not well tuned at the beginning, and they can heavily rely on the quality of the training data. On the other hand, the explicit algorithm in our project allows us to fully understand the mathematical reasonings and gives us a more predictable performance. However, this project also has a few disadvantages compared deep learning approaches. We need to manually set a few hyperparameters, which can influence the results significantly. Moreover, when there are errors in the results, the program will not be able to learn from them.

Comparing these three superpixel algorithms used in our project, SLIC is the fastest with the computation complexity $O(N)$. Ncuts is the slowest option with the efficiency $O(N^{\frac{3}{2}})$, but using it could produce the best objects' closures according to human perceptions. TurboPixels is as efficient as SLIC, but our project always returned small closures of objects if using TurboPixels. Since TurboPixels prefers small and compact superpixels, it will let our program choose compact superpixels to estimate objects' closures than the other two superpixel algorithms. In short, Ncuts performed the best for our project's results, because it considers the global features to generate superpixels while the other two do not.

5 Limit and Future Work

Our research project is based on the Pb edge detector, superpixels, superpixel grouping, and recursive object-closure detection algorithm. Therefore, any limit in these four factors will also negatively influence the performance. For Pb edge detector, there are a few hyperparameters to set. For example, when we calculated the gradient-based features for the Pb edge detector, the size of the disc halves will determine the result edges. However, there is no unique size that will work for all images. Using a single size of disc halves will favor some kind of images but show biases to the others. For superpixels, there are multiple algorithms. For instance, we have discussed three algorithms: SLIC, Ncuts, and TurboPixels. They may have different features and efficiencies. Moreover, depending on how to set their hyper parameters, the superpixel results may be different. Take the number of total superpixels as an example. If we have a large number of superpixels, the whole algorithm will be very slow to compute, but the result can be more accurate. On the other hand, if we have a small number of superpixels, each superpixel will be large, and the objects found will be a very rough estimation. Then, for parametric maxflow, it is an NP-hard problem [1], [14], and thus it is not guaranteed to be an efficient way to solve the optimization problem. Furthermore, the parametric maxflow part can produce a few possible grouping results as candidates. The first choice of the group of superpixels might not be the best estimation. Also, letting a computer choose the best one that makes sense to human users can be very difficult. For the recursive object-closure detection algorithm designed by us, there is also a parameter to set, and it is the condition to stop the search. Right now, we used the area size as the condition, but it might not do well for all cases. Finally, the first three factors, Pb edge detector, superpixels, and parametric maxflow, can be heavily influenced by the color intensities of the input images. As a result, if an object is composed of several parts with different colors, then the algorithm may fail to detect the whole object. Or if several objects with similar colors are put together, the algorithm may treat them as a single object. Therefore, this research project still has limitations in real applications.

The above limitations give us some ideas about future work to be performed on this project. For example, machine learning techniques can be used to adjust hyperparameters for different input images. Since setting hyperparameters is a human-level art, we can use supervised learning to teach the program to select the best hyperparameter combination to detect objects' closures, especially when colors are not the dominating factor to determine an object. The second thing that we can do is to improve the efficiency of the algorithm. The parametric maxflow algorithm can be inefficient. We may be able to find other mathematical models to fit the optimization problem involved. We also can build the tree structure as we propsed earlier. We also want to consider expanding the set of applications of our work. For example, suppose the algorithm can now get all the boundaries for all possible objects in an image; we could add labels for all of them, so that these results can be used for object recognition by deep learning. Also, adding labels by human to the tree-structure can also be used for neural networks to learn the relationships among objects for other images. These will be a kind of data augmentation. Therefore, we are optimistic about the possible future work for this project.

6 Conclusion

In this project, we intended to build an algorithm to detect all possible objects and their closures at different levels by using superpixel groupings. Though it was a difficult task to accomplish and still has limitations, we were happy with our approach, and inspired by our results. The project was based on four key techniques: the Pb edge detector, superpixels, superpixel grouping, and recursive object-closure detection mechanism. The first three factors were the foundation to detect a single object's closure in an image, and the last factor designed by us was to detect all possible objects' closures at different levels. In the process of doing this project, we learned all the related research work and how to apply them in this project. We also implemented this project on MATLAB. This project was a novel attempt to combine the above four key factors to detect the objects' closures at multiple levels in order to achieve the data augmentation purpose for further usages.

7 Acknowledge

We sincerely thank Professor Sven Dickinson, who generously provided guidance and help for the progress of this research project. We also thank Alex Levinshtein, who helped us in implementing the project on the MATLAB platform.

8 References

[1] Levinshtein, A., Sminchisescu, C., Dickinson, S. (2010) "Optimal Contour Closure by Superpixel Grouping". *European Conference on Computer Vision*. Springer-Verlag, 480-493.
[2] Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., and Susstrunk, S. (2010) "SLIC Superpixels". *EPFL Technical Report* no. 149300.
[3] Levinshtein, A., Dickinson, S.J., Sminchisescu, C. (2009) "Multiscale Symmetric Part Detection and Grouping". In: ICCV.
[4] Shi, J. & Malik, J. (1997) "Normalized Cuts and Image Segmentation". *Computer Vision and Pattern Recognition, Proceedings*. IEEE Computer Society Conference on. IEEE, 731-737.

[5] Elder, J.H. & Zucker, S.W. (1996) “Computing Contour Closure”. In: Buxton, B.F., Cipolla, R. (eds.) ECCV. LNCS, vol. 1065, pp. 399–412. Springer, Heidelberg.
[6] Wang, S., Kubota, T., Siskind, J.M., and Wang, J. (2005) “Salient Closed Boundary Extraction with Ratio Contour”. PAMI 27, 546–561.
[7] Arbelaez, P., Maire, M., Fowlkes, C. and Malik. J. (2011) “Contour Detection and Hierarchical Image Segmentation”. *IEEE Trans. Pattern Analysis and Machine Intelligence,*33(5):898-916.
[8] Martin, D.R., Fowlkes, C.C., Malik, J. (2004) “Learning to Detect Natural Image Boundaries Using Local Brightness, Color, and Texture Cues”. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 26(5):530-549.
[9] Canny, J. (1986) “A Computational Approach to Edge Detection”, *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 8, pp. 679-698.
[10] Morrone, M. & Burr, D. (1988) “Feature Detection in Human Vision: A Phase Dependent Engergy Model”, Proc. Royal Soc. Of London B, vol. 235, pp. 221-245, 1988.
[11] Perona, P. & Malik, J. (1990) “Detecting and Localizing Edges Composed of Steps, Peaks and Roofs”, Proc. Int’l Conf. Computer Vision, 1990.
[12] Levinshtein, A., Stere, A., Kutulakos, K.N., Fleet, D.J., Dickinson, S.J., and Siddiqi, K. (2009) “TurboPixels: Fast Superpixels Using Geometric Flows”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2290-2297.
[13] Stahl, J. and Wang, S. (2007) “Edge Grouping Combining Boundary and Region Information”. *IEEE Transactions on Image Processing* 16, 2590–2606.
[14] Kolmogorov, V., Boykov, Y., and Rother, C. (2007) “Applications of Parametric Maxflow in Computer Vision”. In: ICCV.

9 Appendix

Besides this project report, the project’s code was implemented on the MATLAB platform. Before running the code, you need to install the MATLAB 2009 (or later versions), proper C++ compilers, and the VLFeat open source package. To run the code properly, follow the README.txt file in the code directory.