Selective Search

In this section we detail our selective search algorithm for object recognition and present a variety of diversification strategies to deal with as many image conditions as possible. A selective search algorithm is subject to the following design considerations:

- **Capture All Scales.** Objects can occur at any scale within the image. Furthermore, some objects have less clear boundaries then other objects. Therefore, in selective search all object scales have to be taken into account, as illustrated in Figure 2. This is most naturally achieved by using an hierarchical algorithm.
- **Diversification.** There is no single optimal strategy to group regions together. As observed earlier in Figure 1, regions may form an object because of only colour, only texture, or because parts are enclosed. Furthermore, lighting conditions such as shading and the colour of the light may influence how regions form an object. Therefore instead of a single strategy which works well in most cases, we want to have a diverse set of strategies to deal with all cases.
- **Fast to Compute.** The goal of selective search is to yield a set of possible object locations for use in a practical object recognition framework. The creation of this set should not become a computational bottleneck, hence our algorithm should be reasonably fast.

1. Selective Search by Hierarchical Grouping

We take a hierarchical grouping algorithm to form the basis of our selective search. Bottom-up grouping is a popular approach to segmentation [6, 13], hence we adapt it for selective search. Because the process of grouping itself is hierarchical, we can naturally generate locations at all scales by continuing the grouping process until the whole image becomes a single region. This satisfies the condition of capturing all scales. As regions can yield richer information than pixels, we want to use region-based features whenever possible. To get a set of small starting regions which ideally do not span multiple objects, we use the fast method of Felzenszwalb and Huttenlocher [13], which [3] found well-suited for such purpose.

Our grouping procedure now works as follows. **We first use [13] to create initial regions. Then we use a greedy algorithm to iteratively group regions together**: First the similarities between all neighbouring regions are calculated. The two most similar regions are grouped together, and new similarities are calculated between the resulting region and its neighbours. The process of grouping the most similar regions is repeated until the whole image becomes a single region. The general method is detailed in Algorithm 1.

Algorithm 1: Hierarchical Grouping Algorithm

```
Input: (colour) image
Output: Set of object location hypotheses $L$

Obtain initial regions $R = \{r_1, \cdots, r_n\}$ using [13]
Initialise similarity set $S = \emptyset$
foreach Neighbouring region pair $(r_i, r_j)$ do
    Calculate similarity $s(r_i, r_j)$
    $S = S \cup s(r_i, r_j)$

while $S \neq \emptyset$ do
    Get highest similarity $s(r_i, r_j) = \max(S)$
    Merge corresponding regions $r_t = r_i \cup r_j$
    Remove similarities regarding $r_i : S = S \setminus s(r_i, r_*)$
    Remove similarities regarding $r_j : S = S \setminus s(r_*, r_j)$
    Calculate similarity set $S_t$ between $r_t$ and its neighbours
    $S = S \cup S_t$
    $R = R \cup r_t$

Extract object location boxes $L$ from all regions in $R$
```

For the similarity $s(r_i, r_j)$ between region r_i and r_j we want a variety of complementary measures under the constraint that they are fast to compute. In effect, this means that the similarities should be based on features that can be propagated through the hierarchy, i.e. when merging region r_i and r_j into r_t, the features of region r_t need to be calculated from the features of r_i and r_j without accessing the image pixels.