A Stochastic Approach for Selective Search Algorithms

Author: J.C. Gaiser (4007891)

Supervisor: Dr. L.J.P. van der Maaten

December 27, 2014
1 Preface

The topic of this thesis is that of finding a fast and reliable method for employing selective search with as goal to generate valid object proposals. This thesis has been conducted under the supervision of Dr. L.J.P. van der Maaten at the Delft University of Technology at the Pattern Recognition & Bioinformatics Group.
2 Introduction

There is an ever increasing amount of computer vision being deployed on a daily basis. Computer vision is becoming a widely used application in many industrial tasks, ranging from recognizing postal codes to pick and place procedures. However we keep seeing more and more applications of computer vision in non-industrial tasks. Most famous is probably the facial recognition software used in modern cameras or smartphones, however many more examples can be considered, QR codes, image searches, augmented reality, etc.

In order to detect an object in an image, that object needs to be localized first. Generally this means coupling an object localization algorithm with an object classifier. Object classifiers take as input an image and classify that as belonging to some class. Object localization algorithms creates subimages and proposes these subimages to the object classifier to determine the contents. In such a way, an object that is part of a larger image can be localized and classified, in other words: detected. A commonly used object localization method is the sliding window technique. As its name suggests, this method slides a window with some size and aspect ratio over the image, creating object proposals at each step. An example of the sliding window approach can be seen in figure 1. However objects are generally not of the same size and aspect ratio throughout images, therefore sliding just one window over the image is often not enough. That is why often different windows with multiple sizes and aspect ratios are used to create object proposals. In such a manner, the sliding window technique samples a subspace from the space of every possible subimage that can be created from an image. The idea is that the object of interest will most likely be found in that subspace. For example when detecting a face, faces generally have the same aspect ratio but can vary in size depending on the distance to the camera. In such an application it is probably useful to slide multiple sized windows over the image and just a few with different aspect ratio.

In the example of figure 1, the boats can be further away or closer to the camera, partially occluded or fully in sight, rotated in any direction. Therefore both the size of a boat and its aspect ratio is completely unknown to the algorithm that proposes objects. Consider the scenario where we want to detect an object at five different scales and five different aspect ratios in an image of 512×512 pixels, this already results in over 400,000 1 object proposals that need to be processed by the classifier. In order to overcome

\[ \text{Approximately } \frac{512}{4} \times \frac{512}{4} \times 5 \times 5 = 409,600 \]
this large number of object proposals, classifiers have been designed to use fast and easy to compute features[13, 5, 11]. If the object proposal algorithm could be improved to generate fewer proposals, each proposal could be checked more carefully, which could increase the detection rate of such classifiers. An example of such a computationally more expensive feature is the bag of words feature. In some situations it might provide better results than other computationally cheaper features, however since the sliding window technique generates this large amount of proposals, bag of words features becomes computationally unfeasible.

Other algorithms[9, 14] use the appearance model to guide the sliding window technique. Using a branch and bound technique, [9] drastically reduces the search space. Although reducing the number of proposals generated, [1] argues that for non-linear classifiers, [9] still visits over 100,000 windows per image. FI-SPOT[14] uses the most important features in the appearance model to quickly prune many of the proposed windows, after which the remaining windows will be checked using the full appearance model. Although an improvement over the regular sliding window approach, these algorithms still process a large amount of windows.

Another more recent approach[2, 4] uses a measure of ‘objectness’ to determine the likelihood that a subimage contains an object of any class. [4] can very efficiently (reportedly running at 300fps) determine this objectness value for an object proposal and only those proposals with a high score need to be processed by the object classifier. The assumption is that all objects look alike when looking at the normed gradient of the image. Using an SVM classifier and some computational tricks they managed to get speed up their algorithm. These type algorithms can be seen as a sort of prefilter.
step for the object classifier. The increased speed comes at a price however, the resulting windows are often not that accurate at covering the object.

Object proposal algorithms such as, or based on, the sliding window technique can be seen as exhaustive search algorithms. They attempt to exhaust many of the possible locations where an object of interest can be found, at the price of generating a large amount of proposals. Another type of object proposal algorithms exists which uses the contents of the image to create the object proposals: selective search algorithms. Generally these algorithms work much alike, they segment the image using some segmentation algorithm and propose multiple combinations of segments as object proposals. The main advantage of this type of object proposal algorithms is that they generate far fewer proposals, mostly in the hundreds to low thousands. Usually a disadvantage is that many of these methods are very slow at generating proposals, while an important reason to use these algorithms is to increase efficiency.

Within the category of selective search algorithms another distinction becomes clear; deterministic versus stochastic. Deterministic selective search algorithms\cite{12, 3} generally create a segmentation hierarchy. Each pair of neighboring segments gets scored based on their similarity. Then in a greedy fashion, the most similar pair of segments get merged. This process repeats until there is only one segment left, which is the entire image. Each step generates a new segment and thus a new object proposal. Stochastic selective search algorithms\cite{10} take a different approach, instead of greedily selecting the most similar segments, they sample pairs of neighboring segments based on their similarity. In this way, a pair of segments with a high similarity has a high chance of being merged, but this does not necessarily have to happen. A summary of all object proposal algorithms discussed so far can be seen in figure 2.

This thesis investigates selective search algorithms and proposes an algorithm that is both competitive with other selective search algorithms in the quality of the proposals while still being computationally efficient. Section 3 describes the algorithms that have been looked at, section 4 describes the proposed algorithm and section 5 evaluates the proposed algorithm, comparing it to the current state of the art.

3 Research

This section will discuss the algorithms that have been investigated for this thesis. To compare algorithms with one another we use three different val-
Figure 2: A visualization of the different types of categories for creating object proposals. Subspace[9, 14], prefilter[2, 4], deterministic[12, 3] and stochastic[10].

ues: the intersection-over-union (IoU) score to describe the quality of the proposals, the time used to generate the proposals and the number of proposals generated. In this way, we evaluate both quality and efficiency, as we argue that an object proposal algorithm should be optimized in both regards. The IoU score is often used to describe the quality of an algorithm and it can be computed as follows:

\[
O(a, b) = \frac{R_a \cap R_b}{R_a \cup R_b}
\]  

Where \( O \) is the intersection over union score for segments \( a \) and \( b \), \( R_s \) is the region covered by segment \( s \), \( R_a \cap R_b \) is the joint region covered by both segments \( a \) and \( b \) in number of pixels and \( R_a \cup R_b \) is the region covered by either \( a \) or \( b \) in number of pixels.

By computing the maximum IoU score for a ground truth region with every generated proposal we can compute the best IoU for that object. If we do this for every object of some class in a dataset, we can compute the average best IoU score for that class. By computing the mean of the average best IoU scores for each class we can compute the mean average best overlap (MABO) score, which is one value representing the quality of the algorithm. While optimizing parameters we will use this MABO score. To compare algorithms with eachother, we compute the best IoU score for all objects in the VOC2007 and create IoU detection rate graphs. These graphs show on the horizontal axis the threshold in IoU value and on the vertical axis
the percentage of detections that have a higher IoU score. These algorithms can be compared to one another through the ‘area under graph’ (AUC) value, as is done in [8]. We use the VOC2007 dataset for evaluation since many algorithms[8] are also compared using that dataset. This allows us to easily compare with other algorithms. The VOC2007 dataset contains 4952 images with one or more annotations per image, a few examples can be seen in figure 3. In total there are 20 classes in the VOC2007 dataset, ranging from dogs to trains.

3.1 Contour Detection and Hierarchical Image Segmentation

One popular algorithm that was worth investigating, was that of \( gPb \)[3]. Although it is reportedly very slow (\( \frac{1}{240} \)fps according to [6]), they achieve interesting results; specifically the segmentation and edge detector. The \( gPb \) algorithm first creates an edge probability map on multiple different channels of the image, eventually resulting in a global edge probability map which, for each pixel, determines the probability of an edge in one of eight possible directions. By using a watershed algorithm on this edge probability map, the algorithm generates a segmentation. Then a graph \( G(V, E) \) is constructed with each segment a node in \( V \) and each pair of neighboring segments is connected by an edge in \( E \). The weight on these edges that
connect segments is determined by the mean of the probability of an edge in the image at the shared border of the two segments. The algorithm then creates a segmentation hierarchy by greedily selecting the weakest edge and removing it from the graph $G$. Pseudo code for this algorithm can be seen below.

### Algorithm 1: $gPb$

```plaintext
1 // mPb
2 Convert the image to CIE Lab colorspace
3 Compute texture channel
4 Compute gradients for all four channels
5 // sPb
6 Compute eigenvectors of the image
7 Compute the spectral edge probability map
8 // gPb
9 Construct the global edge probability map from mPb and sPb
10 // segmentation
11 Segment the image using a watershed algorithm
12 Create a graph $G = (V, E)$ based on the segmented image
13 while $|V| ≠ 1$ do
14     Select $e ∈ E$ such that $e$ has minimum weight
15     Remove $e$ from $E$
16 end
```

The segmentation algorithm makes use of the edge detector, which consists of two parts called $mPb$ (multiscale) and $sPb$ (spectral) for estimating the probability of the presence of an edge at a certain location. The $mPb$ measure computes an oriented gradient of the image in four separate channels; the first three channels are of the CIE Lab colorspace and the last is an additional texture channel. For each of these channels, an oriented gradient is computed by computing the histogram for each pixel in two halves of an area around that pixel. An example of this approach can be seen in figure 4.

The value of the gradient magnitude $G$ at location $(x, y)$ with angle $\theta$ is then determined by the $\chi^2$ distance:

$$\chi^2(g_\theta, h_\theta) = \frac{1}{2} \sum_i \frac{(g_\theta(i) - h_\theta(i))^2}{g_\theta(i) + h_\theta(i)}$$

(2)

With $g$ and $h$ the two histograms around $(x, y)$ at some angle $\theta$. $mPb$ is
then computed as a linear combination of these cues as follows:

\[ m_{Pb}(x, y, \theta) = \sum_s \sum_i \alpha_{i,s} G_{(i,\sigma)}(x, y, \theta) \]  

(3)

To detect structures at different scales, three different scales (\(\sigma\) values) are used. For color and texture channels these are \(\sigma \in \{5, 10, 20\}\) and for the brightness channel these are \(\sigma \in \{2.5, 5, 10\}\). \(s\) indexes all values for \(\sigma\), \(i\) indexes the channels and \(\alpha_{i,s}\) is computed through gradient descent by optimizing the score of the BSDS dataset, a dataset used by [3] to evaluate their edge probability algorithm.

The spectral part of the edge detector \((s_{Pb})\) uses the eigenvectors obtained from spectral partitioning. In practice, \(n = 16\) eigenvectors are computed for one image. These are then convolved using Gaussian directional derivative filters at multiple orientations. The final spectral probability value then becomes:

\[ s_{Pb}(x, y, \theta) = \sum_{k=1}^{n} \frac{1}{\sqrt{\lambda_k}} \nabla_\theta v_k(x, y) \]  

(4)

Where \(\lambda_k\) is the eigenvalue for eigenvector \(k\) and \(v_k\) eigenvector \(k\). The value for the resulting \(g_{Pb}\) is then given as a linear combination of \(m_{Pb}\) and \(s_{Pb}\) as follows:

\[ g_{Pb}(x, y, \theta) = \sum_s \sum_i \beta_{(i,s)} G_{(i,\sigma)}(x, y, \theta) + \gamma s_{Pb}(x, y, \theta) \]  

(5)

Where \(\beta_{(i,s)}\) and \(\gamma\) are computed through gradient descent based on the score on the BSDS dataset. An example of the resulting \(g_{Pb}\) values can be seen in figure 5.
Using this global probability for the contour, a watershed segmentation is computed with seeds at the local minima of the edge probability map. Then, a graph $G(V, E)$ is constructed where each segment is a node in $V$ and each neighboring segment is connected through an edge in $E$. These edges receive weights according to the probability of an edge at the shared contour of the two segments. The strength of this edge is computed by using a complex system to approximate all borders with linear lines that have one of eight possible orientations. The benefit of this translation is that the strength of a border between two segments (and thus a connection between segments) can be derived from the oriented edge probability map produced by $gPb$. An example of what the original $gPb$ algorithm tries to overcome can be seen in figure 6. Edge probability maps are not necessarily precise; an edge can span multiple pixels in the orthogonal direction of the edge, depending on how sharp it is. The strength of the vertical borders in figure 6 can therefore ‘overflow’ to the weaker horizontal border. This causes outliers in the edge values of the horizontal border. When the connection is computed as the mean of these values, it is erroneously larger than expected. By computing the strength of a connection based on the oriented edge probability map, rather than one single edge probability map (described in [3] as the maximum of all oriented edge probability maps), the effects of this issue is drastically reduced. $gPb$ can then take the orientation of the edge into account, therefore when computing the similarity of the segments separated by the horizontal border, the edge probability in the vertical direction are ignored.

### 3.2 Improvements

As was mentioned before, $gPb$ is cited often and appears to give good results, but at the cost of a lot of computational power. The algorithm consists
Figure 6: Example of why \( gPb \) approximates edges with straight lines and uses the edge probability for a certain orientation to compute the strength of a connection. Displayed is an example of three borders with their corresponding edge probability from an edge map. In the original edge map, the vertical borders most likely span multiple columns. The horizontal border therefore becomes erroneously more strong (specifically because of the two pixels outlined in red).

of three main parts; the edge probability algorithm, the segmentation algorithm and creating the hierarchical structure from this segmentation. For this algorithm to be more efficient in terms of time used it would require a vast speedup. This means that the edge probability algorithm should be replaced with a faster algorithm, as it is an inherently slow algorithm. This is however also one of the main strengths of the \( gPb \) algorithm. Calculating the edge probability takes in the order of a minute on images from the VOC2007 dataset. A recent edge detector described in [6] gives comparable results to the edge detector in \( gPb \) but is many times faster (running at 60fps in the fastest setting). Replacing the edge detector in \( gPb \) with that of [6] should provide similar results. There is an additional side effect to changing the edge detector however; the segmentation algorithm in \( gPb \) depends on knowing the edge probability in a few orientations. These are no longer available as [6] provides just one edge probability map, disregarding oriented edge probabilities. Instead of using the segmentation algorithm from \( gPb \) it is possible to use a segmentation algorithm that does not depend on oriented gradients, like the Felz-Hutt segmentation algorithm. This algorithm segments an image in approximately 0.1s on an image of 512 × 512 pixels.

The revised algorithm will use the edge detector from [6] and the segmentation algorithm from [7]. Each pair of segments that has a common border becomes a connection. The solution proposed here however is to use one edge map and compute the strength as the median of the edge values in contrast to the mean which is used in \( gPb \). This reduces the effect of the problem that strong edges will overflow and erroneously strengthen weaker edges as can be seen in figure 6. Using the median has the same effect as that
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MABO</th>
<th>Time (s)</th>
<th>Nr. of proposals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original gPb</td>
<td>0.6426</td>
<td>240</td>
<td>2450</td>
</tr>
<tr>
<td>Modified gPb</td>
<td>0.6305</td>
<td>0.33</td>
<td>1650</td>
</tr>
</tbody>
</table>

Table 1: Comparison of the original gPb algorithm and the modified version of the algorithm. Running time for the original algorithm has been derived from [6].

described in gPb, that is to be more robust to outliers, while maintaining efficiency. A pseudo code version of the algorithm can be seen below.

```
Algorithm 2: gPb improved
1 Create an edge probability map using [6]
2 Segment the image using [7]
3 Create a graph G = (V, E) based on the segmented image
4 while |V| ≠ 1 do
5     Select e ∈ E such that e has minimum weight
6     Remove e from E
7 end
```

Running the original gPb on the VOC2007 dataset gives a MABO score of 0.643, while running the modified gPb algorithm gives a score of 0.631. This is a mere difference of 1.2%, however the time speedup is in the order of 720 times faster. A summary of these tests can be seen in table 1.

The modified version of the gPb algorithm requires more parameters than the original because of the use of the Felz-Hutt segmentation algorithm. This segmentation algorithm requires a parameter $k$, which determines the size of the segments, and a parameter $\sigma$, which is used to smooth the image before segmentation. Running this algorithm on the VOC2007 dataset while varying these parameters gives us the results from figure 7. It can be seen in figure 7a there that a small value for $k$ (and thus smaller segments) is beneficial for the MABO score. A larger value for $k$ most likely means that some objects have been under segmented, which means that some segments ‘leak’ away from the object to the background. In addition, higher $k$ values mean fewer segments to merge and since each merge results in an object proposal, there are fewer objects proposed. This will also mean that the computation time reduces for larger $k$.

Varying the $\sigma$ value has less effect on the MABO score. A higher $\sigma$ value
Figure 7: Each row shows the results on the VOC2007 dataset when varying one of the parameters for the (modified) \( gPb \) algorithm. The left column shows the ABO for a certain setting, the middle column shows the time it took on average per image and the right column shows the number of bounding boxes generated. (a) Shows the results for varying \( k \) values, (b) shows the results for varying \( \sigma \) values and (c) shows the result for different color spaces.

seems favorable as can be seen in figure 7b, but it seems to behave somewhat erratically. An increased \( \sigma \) value leads to an increasingly smoothed image, causing segments to overlap more easily. In other words, a higher \( \sigma \) means fewer segments, which is what we can see in columns 2 and 3 of row 2 in figure 7.

In addition, we can change the color space in which we process the images. Some color spaces are more invariant to lighting conditions, such as the Hue channel in the HSV color space. Changing the color space can significantly change the score of the resulting algorithm, as can be seen in figure 7c.
The most promising setup of this modified version of $gPb$ appears to be $k = 40, \sigma = 0.2$ and using the RGB color space. It is interesting to see that the grayscale images perform just mildly worse from the RGB color space. Grayscale images have just one third of the data of the RGB color space and should theoretically be much faster.

### 3.3 Selective Search for Object Recognition

Another very promising research result is that of van de Sande et al. [12], which achieves great results on the VOC2007 dataset. In essence, it is similar to the concept of $gPb$; it segments an image, computes features to quantify the similarity between two segments and merges those with the highest similarity first. The segmentation of the image in this algorithm is done by the Felz-Hutt[7] segmentation algorithm. It is many times faster but also less precise[3] than the segmentation of $gPb$. Once the segmentation is done, the algorithm computes a feature vector for each segment. These features were selected to make sure they contain information about the segment, while being easy to compute and easy to propogate. This last constraint is important, as at each merge step the features of both segments needs to be merged as well. In addition, all features have values in the range of $[0, 1]$ so that they can be added up to form a single similarity value. The following features are used in van de Sande:

1. **Color histogram.** A color histogram is computed for each segment. Multiple color spaces are tested, but the approach is generally the same. Compute the histogram for each channel in 25 bins and concatenate the bins. Resulting in 75 bins in the case of the RGB space. The similarity measure for this feature is calculated with the histogram intersection, which works as follows:

$$s_{\text{color}}(i, j) = \sum_{k=1}^{n} \min(c_i^k, c_j^k)$$

Where $s$ is the similarity score, $i$ and $j$ are the indices of two segments, $c_i^k$ is the value of the histogram of segment at bin $k$.

2. **Texture histogram.** Similar to the color histogram, a texture histogram is computed using Gaussian derivatives in eight directions for each color channel. With three channels, eight directions and a bin size of 10 this leads to a histogram of size 240. The similarity score for this feature is computed in the same manner as the color histogram:
$$s_{\text{texture}}(i, j) = \sum_{k=1}^{n} \min(t^k_i, t^k_j)$$

(7)

Where in this case, $t^k_i$ is the value of the histogram of segment $i$ at bin $k$.

3. **Size.** This measurement encourages segments of smaller sizes to be merged first. It is quantified as the number of pixels in the segment divided by the total number of pixels in the image. The similarity measure is computed as:

$$s_{\text{size}}(i, j) = 1 - \frac{\text{size}(i) + \text{size}(j)}{\text{size}(im)}$$

(8)

Where size($i$) is the number of pixels of segment $i$ and size($im$) is the total number of pixels in the image.

4. **Fill.** This measurement helps to fill gaps in segments. The idea is that when a segment is contained in the bounding box of another segment, it is more likely to be merged. The similarity measure is computed as follows:

$$s_{\text{fill}}(i, j) = 1 - \frac{\text{size}(BB_{ij}) - \text{size}(i) - \text{size}(j)}{\text{size}(im)}$$

(9)

Where $BB_{ij}$ is the bounding box of the combination of segments $i$ and $j$.

[12] shows motivation for three of the four similarity scores, these can be seen in figure 8. Figure 8a shows that object localization is a hierarchical task since for example the spoon and fork are in a salad bowl and the salad bowl in turn is located on a table. As another example, the term ‘table’ here might refer to the entire object with everything on top of it or just the wood of the table itself. 8b shows the necessity of the color similarity. In this case, the two cats cannot be separated by texture, however their color is very different. 8c shows a similar argument but for texture instead of color, but in this case it is most likely better to merge segments based on texture, since all objects in the image are more or less green. Lastly 8d shows that object that are enclosed often share a similarity as well. In this case, the wheels of the car are enclosed by its chassis, [12] argues that in
general objects that are enclosed by other objects share similarity. Through these motivations, [12] argues that a solution for segmentation using a single strategy might not exist at all, since there are many conflicting reasons why a region should or should not be grouped together. For that reason, they use a combination of multiple features together.

The similarity measures (or a subset of similarity measures) are summed up to one score, for which the assumption is that the higher the score, the more likely the segments belong to the same object. The algorithm described in [12] computes these similarities for all neighboring segments. It then selects the connection of two segments with the highest similarity score and merges these two segments. It will repeat this process until all segments have been merged and there is only one segment remaining. This results in a hierarchical segmentation tree, similarly as what was done for $gPb$ algorithm. Each node in this tree represents one or multiple segments and is used as an object proposal. Pseudo code for this algorithm can be
Algorithm 3: Selective Search

1. Segment the image using [7]
2. Create a graph $G = (V, E)$ based on the segmented image
3. Compute features for each $v \in V$
4. Compute weights for every $e \in E$ based on calculated features
5. while $|V| \neq 1$ do
   6. Select $e \in E$ such that $e$ has minimum weight
   7. Remove $e$ from $E$
6. end

These steps are then repeated for multiple color spaces, different combinations of the features described above and also multiple values for the parameter $k$ in the Felz-Hutt segmentation algorithm, which determines the size of the segments. The most ‘expensive’ version of their algorithm uses five different color scales, four different feature combinations and four different parameters for the Felzenszwalb’s algorithm. This means a total of 80 strategies.

The algorithm described in [12] is similar to that of gPb. Both algorithms compute a segmentation of the image, both algorithms compute features for segments and their neighbors and greedily merge them one at a time, creating a segmentation hierarchy. The main difference between these two algorithms is that [12] computes different features and uses multiple color scales and parameters (referred to as strategies) to create more diversity. [12] describes three versions of their algorithm; single strategy, fast and quality. Each having increasingly more strategies and thus more windows to check. The single strategy version uses one segmentation, one set of features and in one colorspace. This method reportedly achieves a mean average best overlap (MABO) of 0.693. The fast version uses two colorspace (meaning the entire process is repeated in both colorspace), two different combinations of features and two segmentations. This results in a total of 8 combinations of parameters (hence 8 strategies) and scores an MABO of 0.799. The quality version uses even more combinations; 5 colorspace, 4 different combinations of features and 4 different segmentation levels, resulting in a total of 80 strategies. This version achieves an MABO of 0.878, which is a very good result, but this is achieved at the cost of the number of windows (over 10,000) to process and thus the time (more than 17s per image). An overview of their versions can be found in table 2.
Table 2: Overview of selective search versions as described in [12]. Original

<table>
<thead>
<tr>
<th>Version</th>
<th>Strategies</th>
<th>MABO</th>
<th># win</th>
<th># strat</th>
<th>time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Strategy</td>
<td>HSV, C+T+S+F, k = 100</td>
<td>0.693</td>
<td>362</td>
<td>1</td>
<td>0.71</td>
</tr>
<tr>
<td>Selective Search Fast</td>
<td>HSV,Lab, C+T+S+F, T+S+F, k = 50, 100</td>
<td>0.799</td>
<td>2147</td>
<td>8</td>
<td>3.79</td>
</tr>
<tr>
<td>Selective Search Quality</td>
<td>HSV,Lab,rgI,H,I, C+T+S+F, T+S+F, F,S, k = 50, 100, 150, 300</td>
<td>0.878</td>
<td>10,108</td>
<td>80</td>
<td>17.15</td>
</tr>
</tbody>
</table>

With this many possibilities to choose from, it seems interesting to figure out what the effect of each component is on the MABO score. To do this, we can take a single strategy version of the algorithm and vary one of the parameters to see how this affects the MABO score. Results for these tests can be seen in figure 9. Note that this is an algorithm that is reproduced based on [12], not their actual algorithm.

Similarly as with gPb, smaller values of \( k \) is beneficial for the MABO score. Note however that the MABO score is approximately 0.655 here at its highest point, compared to 0.6305 of gPb. The computation time however has also increased, this is due to the features that need to be computed and maintained for each segment. The \( \sigma \) value causes some erratic results on the MABO score, however as with gPb it is clear that an increased \( \sigma \) leads to less segments and thus less proposals. Using different color spaces does not seem to have much effect, with the exception of the hue channel in the HSV color space. This seems to be the worst performing color space. Interestingly, Lab and RGB have a very similar MABO score, however Lab is slightly faster and produces nearly 200 fewer proposals on average. Lastly, using different combinations of features seems to have a large impact. Using all features
Figure 9: Each row shows the results on the VOC2007 dataset when varying one of the parameters for the (reproduced) van de Sande algorithm. The left column shows the ABO for a certain setting, the middle column shows the time it took on average per image and the right column shows the number of bounding boxes generated. (a) Shows the results for varying $k$ values, (b) shows the results for varying $\sigma$ values, (c) shows the results for different color spaces and (d) shows the MABO score for different combinations of features used, (C)olour, (T)exture, (S)ize and (F)ill (the number of segments remain the same with each setting and is thus omitted).
produces the best result, while using only one feature (the texture feature specifically) seems to produce the worst results, which is also reported in [12].

To further increase the MABO score, the authors of [12] introduced the term ‘strategies’. So far we have only examined single strategies settings, but [12] theorizes that there is no single strategy that can find all objects. It makes sense to search for objects in multiple ways; different sizes, texture versus color, etc. Using a second strategy creates an entirely new segmentation hierarchy and produces many different object proposals. Setting up a new strategy takes time however, the segmentation often needs to be recomputed, features need to be calculated again, etc. A different approach introduced here, inspired by [10], is to use the connections strengths as weights and randomly sample one connection to merge based on these weights, in contrast to greedily merging the two most similar segments. The hypothesis here is that creating diverse segmentation hierarchies is much faster than running multiple strategies, while still being beneficial to the MABO score. It is then possible to run the merging part of the algorithm a few iterations, on the same initial segmentation and features, and get more diversity in the proposals. The results of this modification can be found in figure 10.

3.4 Prime Object Proposals with Randomized Prim’s Algorithm

The work described in [10] resembles that of [12] in many ways. The key difference is that instead of greedily selecting the most similar connection, it samples a random connection based on their similarity. The higher the similarity, the more likely that connection is chosen to be added to the current proposal. In addition, this algorithm does not return a proposal at each step, instead it adds a random number of segments to some randomly
chosen starting segment and then returns the result as a proposal.

The features per segment are slightly different. [10] uses similar features as [12] for the color similarity and size in number of pixels, but no size expressed in bounding box and no texture similarity. Instead it uses a ‘common border ratio’ similarity, for which the similarity is computed as follows:

\[
s_{\text{border}} = \max\left(\frac{l_{n|m}}{l_n}, \frac{l_{n|m}}{l_m}\right)
\]

Where \(l_n\) and \(l_m\) are the perimeters of segments \(n\) and \(m\) respectively and \(l_{n|m}\) is the perimeter of the shared border of \(n\) and \(m\).

Similarly to [12], these similarities get added to one value. However in this case the authors trained a logistic function to learn the weights for each feature to model the probability that \(n\) and \(m\) are superpixels of the same object. The model looks as follows:

\[
\rho_{n,m} = \sigma(w^\top \Phi_{nm} + b)
\]

\[
\sigma(x) = \left(1 + \exp(-x)\right)^{-1}
\]

With \(\rho\) the logistic function, \(\Phi\) the vector of features, \(b\) a bias term and \(\sigma\) the sigmoid function. The values for the weights can be found in table 3.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Color</th>
<th>Border</th>
<th>Size</th>
<th>Bias</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight</td>
<td>2.69</td>
<td>1.00</td>
<td>2.36</td>
<td>-3.0</td>
</tr>
</tbody>
</table>

Table 3: Results from training a logistic function on the weights for the similarity score.

Positive and negative examples have been computed from the VOC2007 dataset using the set of segmented images. A positive example is one in which a pair of neighboring segments have at least 60% overlap with the object in the image. A negative example is any pair of neighboring segments do not belong to the same object.

The process of [10] can be described in three steps, which are visualized in figure 11. In the first step, a random segment is selected as starting point. Then, one of its neighbors is randomly selected, using the similarity between the starting segment and its neighbor as weight. This segment then gets added to the result (current object proposal), after which some features need to be recomputed. The algorithm then checks whether it is
time to stop the process of adding more segments and outputs the current combinations of segments as an object proposal. The stopping criterion is somewhat randomized. When selecting a starting segment, a threshold in the range of $[0,1]$ is uniformly generated. Then each time a segment is added, the following equation is evaluated:

$$
\xi(T_k, e_k) = \frac{1 - \rho_{n,m} + \alpha(T_k)}{2}
$$

(13)

Where $T_k$ is the set of segments at iteration $k$, $e_k$ is the edge that was added in iteration $k$, $\rho_{n,m}$ the weight of $e_k$ (which can be considered a probability that the edge contains segments of the same object) and $\alpha$ as the fraction of objects in the training data with area smaller than $T_k$. This process can be repeated as many times as is desired, allowing full control over the number of proposals generated, unlike the previously discussed algorithms. This allows this algorithm to be used in a “detection-on-a-budget” system, where the user determines a certain budget in number of proposals or processing time for the object proposal algorithm. Pseudo code of this algorithm can be seen below.

**Algorithm 4: Randomized Prim’s**

1. Segment the image using [7]
2. Create a graph $G = (V, E)$ based on the segmented image
3. Compute features for each $v \in V$
4. Compute weights for every $e \in E$ based on calculated features
5. for $i$ in range $[0, n]$ do
   6. $s \leftarrow \text{Uniform}(V)$
   7. $\xi_0 \leftarrow \text{Uniform}([0,1])$
   8. $k \leftarrow 0$
   9. $T_k \leftarrow \{s\}$
   10. repeat
       11. $n \leftarrow \text{Uniform}(N(s))$ $k \leftarrow k + 1$
       12. $T_k \leftarrow T_{k-1} \cup n$
       13. $\xi(T_k, e_k) \leftarrow \frac{1 - \rho_{n,m} + \alpha(T_k)}{2}$
   14. until $\xi(T_k, e_k) > \xi_0$
15. end

As a result of the stochastic way in which this algorithm functions, every run of merging segments can be different. Instead of choosing a variety of different strategies as is done in [12], you can choose one strategy and
create many different segmentation combinations from that strategy. That way, if two segments are wrongfully merged in one run, it can be corrected in another run. Sampling these segments based on their similarity and creating different combinations of segments can be evaluated much faster than evaluating many different strategies, since these require to recompute the segmentation again and thus recompute the features.

Another minor difference compared to [12] is the use of a logistic function to compute optimal weights for the features. Evaluating the advantage of using weights found using a logistic function over static weights of 1 (as used in van de Sande) shows there is only a slight difference in MABO score, as can be seen in figure 12. This raises some doubts about the validity that features increase results of the object proposal algorithm.

3.5 BING: Binarized Normed Gradients for Objectness

[4] works in a very different manner than those discussed before. It does not depend on segmenting an image into superpixels, instead it depends on a sliding window technique. However instead of classifying each window
Figure 13: (a) shows an input image with red windows as objects and green windows as non-objects. (b) shows these same windows but rescaled to a $8 \times 8$ image and in the normed gradients feature space. (c) shows the normed gradient features for the input image for a number of scales. (d) shows the weights of the SVM classifier, these weights determine what the objectness score of a window is.

to be some object or not, each window is reduced to only 64 dimensions (an $8 \times 8$ normed gradient image). This window is then multiplied in an efficient manner with weights obtained through a SVM. The result is a measure for the ‘Objectness’ of that window. This measure can be computed extremely fast, at reportedly 300fps according to [4]. This algorithm works in a comparable fashion as the regular sliding window approach, meaning it also suffers from the same negative aspects; the algorithm will have to evaluate an exponential number of windows when different ratios, sizes and orientations have to be checked. The benefit here is that evaluating a window can be done very fast. In effect filtering the set of windows before proposing possible results to the often more computational intensive classifier. This allows the classifier to spend more time per window and thus check them more thoroughly.

As can be seen in figure 13, objects share a high similarity in the normed
4 Stochastic Selective Search

Keeping in mind the results obtained from evaluating the related work, the best performing algorithm appears to be the randomized Prim’s algorithm. It not only outperforms the other algorithms in terms of time and MABO score, but it also generates fewer proposals. Since the randomized Prim’s algorithm seems to keep efficiency as well as quality into account, which is a goal for the algorithm introduced in this thesis, it makes sense to start investigating improvements over this algorithm.

As was previously discussed, the randomized Prim’s algorithm exists of three steps; choosing a start segment, adding segments and determining when to stop the process. Each of these steps can be checked if there are improvements that can be made. Starting with the first step, selecting a starting segment. As [10] describes this process, it selects a segment uniformly from all segments, but perhaps it is possible to add some prior knowledge to this sampling. One obvious method could be to favour segments that are more located towards the center of the image. One solution to do this would be to weight each segment using a Gaussian distribution with as mean location the center of the image and some parameter $\sigma$. This parameter determines how spread out the Gaussian distribution is, setting $\sigma$ to infinity results in a uniform selection, whereas a lower value leads to a more spikey distribution. Another ultimately similar solution is to turn to the VOC2012 segmentation data. This dataset contains 2913 annotations of segmented images. An example can be seen in figure 14.

By creating a new image and summing up the mask of all the segmented annotations, we can also create a sort of prior probability for where objects are mostly located in images. Intuitively this will result in a Gaussian like image as well. The result of this step can be seen in figure 15.

Another method is to compute a prior based on the possibility of an object being present at some location; ie. an objectness score. One method of doing this is by using the BING algorithm to compute a likelihood over the entire image. Since BING reportedly runs at 300fps, this should be a fast and easy preprocessing step. BING however outputs windows with scores which represent the objectness of that window. The likelihood for an entire image can then be computed by adding each of these windows together, using their weight as values. A visualization of what this selection prior looks like can be seen in figure 16.
Figure 14: An example of the segmented annotation of the VOC2012 dataset.

Figure 15: The result of creating a selection prior based on annotated data from the VOC2012 dataset.
Figure 16: A visualization of what the objectness selection prior looks like.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MABO</th>
<th>Time (s)</th>
<th>Nr. of proposals</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSS (uniform)</td>
<td>0.693</td>
<td><strong>0.174</strong></td>
<td>477</td>
</tr>
<tr>
<td>SSS (location)</td>
<td>0.694</td>
<td><strong>0.174</strong></td>
<td>490</td>
</tr>
<tr>
<td>SSS (Objectness slow)</td>
<td>0.696</td>
<td>0.251</td>
<td>485</td>
</tr>
<tr>
<td>SSS (Objectness fast)</td>
<td><strong>0.698</strong></td>
<td>0.219</td>
<td>490</td>
</tr>
</tbody>
</table>

Table 4: Comparison of the different starting segment selection procedures.

A slightly faster way is to check objectness the other way around; for each segment, retrieve the windows from BING which it overlaps with and calculate the mean of those scores. Both will be compared here, the former being referred to as ‘Objectness (slow)’ while the slightly faster version will be referred to as ‘Objectness (fast)’. Results for these different selection priors can be seen in figure 17, for which the remaining parameters have been kept the same. Fastest of the four variations is the uniform and location based sampling, at approximately 0.17s per image. Objectness (fast) is slightly slower than these, at 0.22s per image, while Objectness (slow) is a bit slower than that, at 0.25s per image. A summary of the results can be seen in table 4. Surprisingly, none of the selection methods have any significant impact on the result of the algorithm, which allows us to pick the fastest of the bunch, which is either location based or uniform selection. We select location based sampling as it gives just a slightly better result based on our tests.

The second step of the algorithm uses features from each segment to sample a neighboring segment and add that to the current set of segments. The original randomized Prim’s algorithm describes three features; color, common border and size. Van de Sande describes four features; color, texture, size and bounding box. This results in a total of five possible features; color, texture, size, bounding box and common border. Which of these features or
combination of features give the best result? Luckily, since we have only five dimensions, we can do a bruteforce search to find the optimal combination of features. Results for this test can be found in figure 18. A comparison of the best set of features, the same set of features but with weights obtained from a linear SVM model and using no weights at all can be seen in figure 19.

Lastly the termination step can be changed to allow the algorithm to stop merging at a different time. One possibility is to set a threshold in the range \([0, 1]\) and at each iteration of the process to generate a number in the range \([0, 1]\) and check if it exceeds the threshold or not. In effect this will mean that each iteration has a chance, equal to the threshold, of continuing the process. A test that varies this parameter can be seen in figure (TODO). Another possibility is to uniformly generate a random number \(l\) in the range \([0, ms]\) when selecting a starting segment, where \(ms\) is the maximum number of segments that can be added. Then the algorithm adds at most \(l\) number of segments to the initial starting segment. A test that varies the number \(n\) can be seen in figure 20. A comparison of the best results of these two stopping possibilities can be seen in figure (TODO).

Interestingly, the combination of uniformly selecting a starting segment, uniformly selecting a neighboring segment and choosing a random stopping criterion does not perform all that bad. It is performing only slightly worse than the best combination that achieves the best ABO score, however it is a lot faster. Because it is this much faster, it can even be run with two strategies in the same time as the best performing algorithm.
Figure 18: MABO score for different combinations of similarities. Other parameters have been kept the same ($k = 130$, $\sigma = 1.2$, $ms = 26$ using the maximum number of segments to add stopping criterion).

Figure 19: IoU detection rate graph for using weighted features (WF-SSS), features with weights 1 (F-SSS) and no features at all (SSS).

Figure 20: Results for varying the parameter $ms$ for the MABO score, time to process and number of bounding boxes generated.
5 Evaluation

This thesis focuses on finding a fast, yet reliable selective search method. Two quality factors play a role in determining what is a ‘better’ algorithm; accuracy and speed. Speed can be measured in two ways, the time it takes for the selective search algorithm to propose objects and secondly, the number of proposals it generates. An algorithm can be very fast to produce proposals, but if there are too many proposals being generated, there is no benefit in using selective search. The average best overlap (or ABO) is often used to compute the quality of object proposals with respect to the annotated ground truth object location. The overlap is computed as follows:

\[
O(a, b) = \frac{R_a \cap R_b}{R_a \cup R_b}
\]

where \(O(a, b)\) is the overlap score of region \(a\) with region \(b\). \(R_a\), \(R_b\) the region covered by \(a\) and \(b\) respectively. \(R_a \cap R_b\) is the number of pixels \(R_a\) and \(R_b\) have in common, while \(R_a \cup R_b\) is the number of pixels \(R_a\) and \(R_b\) cover in total. The average best overlap on a dataset can be computed by, for each image, computing the maximum score over all object proposals and then compute the average of these best overlap scores.

It is important that all three measures are optimized for selective search to be a useful technique in object detection or object tracking. The sliding window technique is only efficient in the time it takes to propose objects and, for many cases, it also achieves a good ABO score. It is however not efficient in the number of proposals that it generates, this can easily go up to a few hundred thousands with some variety in parameters used. The goal of the selective search methods that have been researched in this thesis try to optimize all three measures.

6 Discussion

discussion of the results, and recommendations and suggestions for future research.
References


